Estratégia Concursos

Aula 00

SEFAZ-RJ (Auditor - Área TI) Engenharia de Software

Autor:
Diego Carvalho, Equipe Informática e TI

01 de Fevereiro de 2025

Índice

1) Kanban 5
2) Questões Comentadas - Kanban - Multibancas 9
3) Lista de Questões - Kanban - Multibancas 21
4) TDD 28
5) Questões Comentadas - TDD - Multibancas 36
6) Lista de Questões - TDD - Multibancas 66
7) BDD 82
8) BDD - Questões Comentadas 88
9) BDD - Lista de Questões 90
10) DDD 92
11) MDA 104
12) FDD 106
13) Questões Comentadas - FDD - Multibancas 109
14) Lista de Questões - FDD - Multibancas 118
15) DSDM 125
16) Noções de DevOps 129
17) Noções de DevOps - Questões Comentadas 138
18) Noções de DevOps - Lista de Questões 141
19) Refatoração 143
20) Questões Comentadas - Refatoração - Multibancas 162
21) Lista de Questões - Refatoração - Multibancas 180
22) Integração Contínua 189
23) Questões Comentadas - Integração Contínua - Multibancas 195
24) Lista de Questões - Integração Contínua - Multibancas 201
25) Modelagem Ágil 204
26) Modelagem Ágil - Questões Comentadas 208
27) Modelagem Ágil - Lista de Questões 214
28) MVP 217

Índice

29) MVP - Questões Comentadas 221
30) MVP - Lista de Questões 225
31) Deployment Pipeline 228
32) Agile Think Canvas 230
33) Gestão Ágil de Projetos 231
34) Técnicas de Planejamento de Escopo 237
35) Técnicas de Estimativa de Escopo - Teoria 240
36) Técnicas de Estimativa de Escopo - Questões Comentadas 253
37) Técnicas de Estimativa de Escopo - Lista de Questões 273
38) Especificação Por Exemplo - Teoria 280
39) Débito Técnico 287
40) Ágil Escalado 292
41) Nexus - Teoria 296
42) Nexus - Questões Comentadas 308
43) Nexus - Lista de Questões 316
44) SAFe - Teoria 320
45) SAFe - Questões Comentadas 323
46) SAFe - Lista de Questões 330
47) Práticas Ágeis - Management 3.0 - Teoria 333
48) Práticas Ágeis - Management 3.0 - Questões Comentadas 344
49) Práticas Ágeis - Management 3.0 - Lista de Questões 350
50) Técnicas de Priorização de Backlog - Teoria 353
51) Técnicas de Priorização de Backlog - Questões Comentadas 367
52) Técnicas de Priorização de Backlog - Lista de Questões 377
53) Técnicas de Colaboração de Time Ágil - Teoria 381
54) Técnicas de Colaboração de Time Ágil - Questões Comentadas 398
55) Técnicas de Colaboração de Time Ágil - Lista de Questões 411
56) Disciplined Agile Delivery (DAD) 416

Índice

57) Agile Waterfall 424

KANBAN

Conceitos Básicos

INCIDÊNCIA EM PROVA: MÉDIA

Seus lindos, vamos para outro assunto! *O que é esse tal de Kanban?* Kanban significa cartão ou placa visual, em japonês. **Trata-se de um método para gestão de mudanças** com foco na visualização do trabalho em progresso (também chamado de *Work In Progress* – WIP), identificando oportunidades de melhorias, tornando explícitas as políticas seguidas e os problemas encontrados e, por fim, favorecendo uma cultura de melhoria evolutiva.

Antes de continuar, eu tenho que fazer uma pequena pausa! **Há uma certa polêmica sobre se o Kanban é uma metodologia de desenvolvimento de software ou não!** David J. Anderson – pioneiro do Kanban – acha que não é (conforme podemos ver nas declarações apresentadas abaixo)! No entanto, é bastante comum ver algumas bancas o tratando como uma metodologia de desenvolvimento de software.

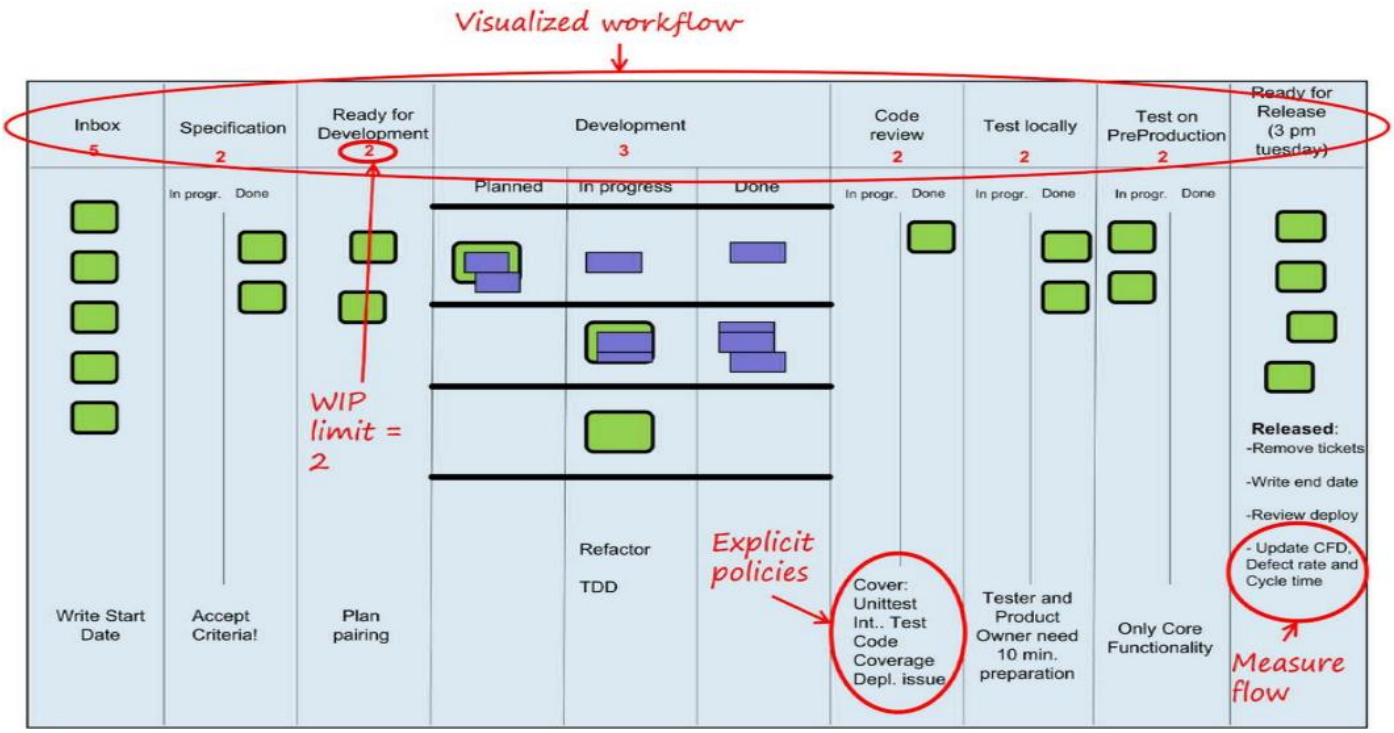
> *"Kanban **is not a software development life cycle or project management methodology**! It is not a way of making software or running projects that make software!" – David J. Anderson*
>
> *"There is **no kanban process for software development**. At least I am not aware of one. I have never published one" – David J. Anderson*
>
> *"It is actually **not possible to develop with only Kanban**. The Kanban Method by itself does not contain practices sufficient to do product development" – David J. Anderson*

O Kanban pode ser visto como um acelerador para a condução de mudanças ou até mesmo um método para implantação de mudanças em uma organização. Ele não prescreve papéis, práticas ou cerimônias específicas (como faz, por exemplo, Scrum). Em vez disso, ele oferece uma série de princípios para otimizar o fluxo e a geração de valor dos sistemas de entrega de software. *Agora o que ele tem a ver com a sua tradução em japonês?*

Galera, ele funciona como uma espécie de cartaz ou placa visual contendo vários post-its – aquele papelzinho colorido para colar lembretes. **Dessa forma, ele permite uma melhor visualização do fluxo de trabalho, favorecendo a transparência para todos os envolvidos.** Além disso, ele permite mudar prioridades facilmente e entregar funcionalidades a qualquer momento. Não há preocupação com estimativas nem em ser iterativo.

Como pode ser visto a partir da imagem apresentada a seguir, todo fluxo de trabalho se torna visível no Kanban. Os limites do Work in Progress são estabelecidos – o fluxo é contínuo sem requerer estimativas. A equipe assume a responsabilidade sobre o processo e se auto-organiza para

otimizá-lo e para ajudar a resolver seus eventuais problemas. O Kanban é construído sobre os conceitos de mudança evolucionária.



Dessa forma, uma possível abordagem é começar a entender como funciona atualmente seu sistema de desenvolvimento de software. Quando conseguir visualizar, medir e gerenciar o fluxo utilizado, melhore-o um passo por vez, aliviando seu maior gargalo, isto é, o processo evoluirá aos poucos. **Isso é muito diferente do que ocorre, por exemplo, no Scrum – em que se inicia definindo papéis, processos e artefatos.**

Isso faz do Kanban um método ideal para utilização em conjunto com outros processos – do Scrum ao Cascata. Ele também é excelente quando estruturas organizacionais inibem mudanças radicais, sendo construído principalmente sob o conceito de melhoria contínua. Ele somente utiliza mudanças radicais em situações especiais, nas quais mudanças estruturais são necessárias ou quando sérias mudanças de desempenho precisam ser feitas.

O modelo é uma boa opção tanto para desenvolvimento de software quanto para operação e manutenção. Kanban e Scrum não são opostos! **Nada impede que se comece a usar o Scrum e se utilize o Kanban para impulsionar mudanças futuras.** Os projetos – apesar de não insistirem no compromisso com iterações planejadas – são muito bem controlados, com cadência fixa, visualização permanente, medição do tempo de ciclo, fluxo de tarefas e ciclos de feedback curtos.

Galera, existem diversas diferenças entre ambos: Scrum requer iterações e o Kanban, não – mas sugere-se que haja uma cadência de entradas e entregas. Quando o Kanban incorpora iterações, ele é tipicamente chamado Scrumban (para diferenciá-lo do Scrum original), uma vez

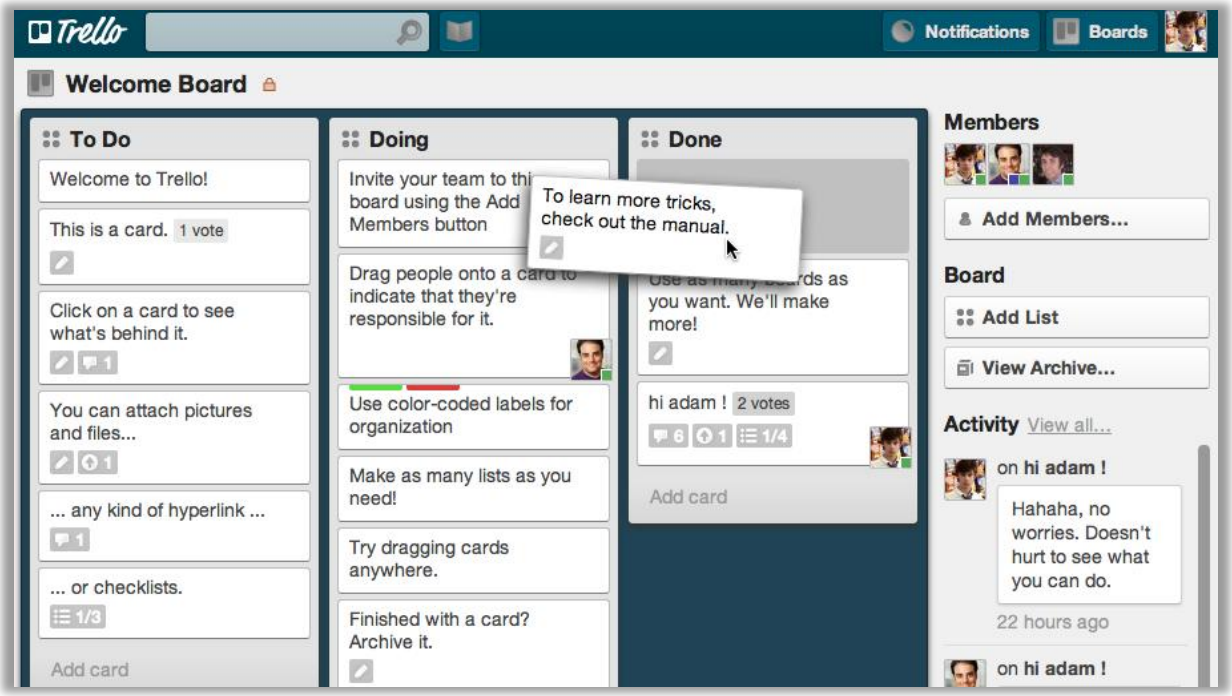
que Scrum não gerencia explicitamente o trabalho em progresso e o Kanban não utiliza iterações por padrão. *Bacana?*

Scrumban é uma metodologia híbrida de gerenciamento de projetos que combina elementos do Scrum e do Kanban. Ele foi criado para aproveitar a estrutura e o planejamento de sprints do Scrum, juntamente com a flexibilidade e o foco no fluxo contínuo de trabalho do Kanban. Scrumban é especialmente útil para equipes que desejam a organização e as cerimônias do Scrum, mas com a liberdade e a adaptabilidade do Kanban. Vamos ver agora os princípios ou restrições:

PRINCÍPIOS OU RESTRIÇÕES DO KANBAN
- Comece com o que você tem hoje;
- Estimule a liderança em todos os níveis da organização;
- Visualize cada passo em sua cadeia de valor, do conceito geral até o software que se possa lançar;
- Limite o Trabalho em Progresso (WIP), restringindo o total de trabalho permitido para cada estágio;
- Torne explícitas as políticas sendo seguidas;
- Meça e gerencie o fluxo, para poder tomar decisões bem embasadas, além de visualizar as consequências;
- Identifique oportunidades de melhorias, na qual a melhoria contínua é responsabilidade de todos.

PRÁTICAS DO KANBAN
- Implemente mecanismos de feedback;
- Gerencie e meça o fluxo de trabalho;
- Visualize o processo;
- Limite o WIP (Work In Progress);
- Torne as políticas dos processos explícitas;
- Melhore colaborativamente e com métodos científicos.

Vamos detalhar o WIP! Trata-se de tarefas que estão em execução em determinado ponto do processo. *Por que devemos limitar o WIP?* **Porque quanto maior o número de tarefas em andamento em determinado ponto do processo, mais tempo a tarefa permanecerá no fluxo.** *Então ele deve ser pequeno?* Não, ele não deve ser muito pequeno nem muito grande. Em geral, se for muito pequeno, qualquer limitação pode parar o processo de desenvolvimento.



E se for muito grande? Nesse caso, muitas tarefas simultâneas levam a grandes perdas e confusões. *Então, qual é o tamanho ideal?* **Bem, isso não existe! Não há um número mágico – é necessário descobrir o tamanho empiricamente de acordo com o contexto da organização.** Ahhh... se você utiliza o Trello, ele é uma forma de apresentar o trabalho sendo realizado, mas também existem outras ferramentas (KanbanFlow, Kanbanery, Leankit, Visual WIP, etc).

QUESTÕES COMENTADAS – DIVERSAS BANCAS

1. **(FGV / TJ-RJ – 2024)** Diversas empresas têm adotado o Kanban como ferramenta de aumento do fluxo e da produtividade no desenvolvimento de software. Considerando que esta tecnologia requer a adesão a algumas práticas fundamentais, avalie se as afirmativas a seguir são verdadeiras (V) ou falsas (F).

( ) Os quadros Kanban incorporam o princípio da visualização do trabalho que se baseia na exibição de cartões que correspondem a itens da lista de pendências do produto.

( ) Empregando o modelo de Pull a equipe puxa para seu fluxo de trabalho as pendências da lista conforme a sua capacidade se torna disponível.

( ) A imposição de limite para o número de tarefas que um time possui atualmente corresponde ao WIP (Work In Progress), e traz o benefício de aumentar o foco e, ao mesmo tempo, reduzir a mudança de contexto.

As afirmativas são, respectivamente,

a) F – V – F.
b) V – F – F.
c) F – F – V.
d) V – V – F.
e) V – V – V.

Comentários:

(V) Os quadros Kanban incorporam o princípio da visualização do trabalho que se baseia na exibição de cartões que correspondem a itens da lista de pendências do produto;

(V) Empregando o modelo de Pull, a equipe puxa para seu fluxo de trabalho as pendências da lista conforme a sua capacidade se torna disponível;

(V) A imposição de limite para o número de tarefas que um time possui atualmente corresponde ao WIP e traz o benefício de aumentar o foco e, ao mesmo tempo, reduzir a mudança de contexto.

Gabarito: Letra E

---

2. **(FGV / AL-SC – 2024)** A prática de Test Driven Development (TDD, ou Desenvolvimento Orientado por Testes) se relaciona com o conceito de verificação e validação e se baseia em um ciclo para garantir a qualidade do código. Entre as características do TDD, é correto o que se afirma em:

a) trata-se de desenvolvimento orientado para os comportamentos, sendo o desenvolvedor responsável por escrever os testes e validá-los de forma que eles funcionem.

b) trata-se de desenvolvimento orientado para os testes, sendo atribuição do programador escrever como o problema deve se comportar.

c) na prática, se refere a escrever um teste automatizado durante o desenvolvimento do código de fato.

d) o ciclo do TDD se inicia com a criação de um teste, para auxiliar a codificação, segue com a realização da codificação para passar no teste, e finaliza com a eliminação das redundâncias, ao se refatorar o código.

e) no desenvolvimento de programas são intercalados testes, elicitação de requisitos e desenvolvimento de código, tendo os testes embutidos em um programa separado que os executa e invoca o sistema que está sendo testado.

Comentários:

(a) Errado. Embora o TDD envolva a escrita de testes pelo desenvolvedor, ele não é especificamente orientado para os comportamentos (Behavior-Driven Development, BDD). O TDD foca na criação de testes para cada pequena funcionalidade do código;

(b) Errado. No TDD, o foco é no desenvolvimento orientado a testes, onde o programador escreve testes antes de implementar a funcionalidade correspondente, mas isso não implica necessariamente em descrever o comportamento do problema;

(c) Errado. No TDD, o teste é escrito antes do código de fato, não durante o desenvolvimento do código;

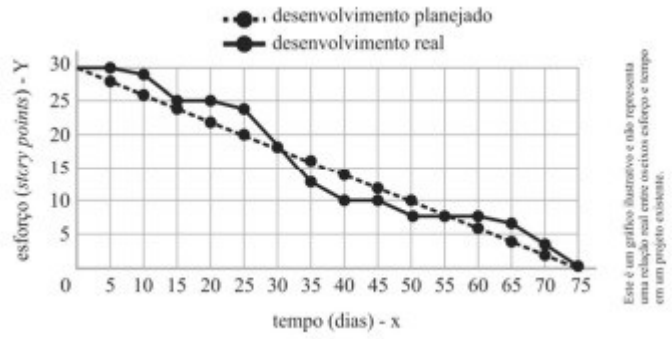
(d) Correto. O ciclo do TDD começa com a criação de um teste que falha inicialmente, segue com a codificação para passar no teste, e finaliza com a refatoração do código para eliminar redundâncias e melhorar a estrutura sem alterar a funcionalidade;

(e) Errado. No TDD, os testes não são apenas intercalados com o desenvolvimento de código, mas são escritos antes do código, guiando o desenvolvimento. A elicitação de requisitos é um processo separado do ciclo de TDD.

Gabarito: Letra D

---

3. **(CESPE / CAU-BR – 2024)** A figura a seguir descreve um Kanban de forma visual, mostrando a produtividade da equipe esforço (eixo-y) até o último dia da sprint (eixo-x), ou seja, esse Kanban liga os dias da sprint com o trabalho restante.



Comentários:

A figura mostrada é um gráfico de Burndown, não um Kanban. O Burndown chart ilustra a quantidade de trabalho restante (em story points) ao longo do tempo (em dias), permitindo acompanhar o progresso do desenvolvimento em relação ao plano inicial. No gráfico, a linha pontilhada representa o desenvolvimento planejado e a linha contínua representa o desenvolvimento real. Um Kanban, por outro lado, é uma ferramenta visual que gerencia o fluxo de trabalho através de cartões em colunas que representam diferentes estágios do processo.

Gabarito: Errado

---

4. **(CESPE / MPO – 2024)** Em projetos de software, utilizam-se ferramentas de acompanhamento de equipes e tarefas entregues por meio de quadros do tipo Kanbam.

Comentários:

Kanban (a banca infelizmente escreveu "Kanbam") é uma ferramenta utilizada para acompanhar equipes e tarefas em projetos de software. Ele utiliza quadros visuais para gerenciar o fluxo de trabalho, ajudando a monitorar o progresso das tarefas e a identificar gargalos. Esta abordagem facilita a visualização das etapas de desenvolvimento e a coordenação das entregas.

Gabarito: Correto

---

5. **(CESPE / SEFIN de Fortaleza-CE – 2023)** *To do*, *doing* e *done* são três estágios básicos do método Kanbam usados para caracterizar o andamento das atividades de um projeto de desenvolvimento de software.

Comentários:

To Do, Doing e Done são três estágios básicos no método Kanban que são usados para visualizar e acompanhar o andamento das atividades em um projeto. No quadro Kanban, as tarefas são movidas entre essas colunas para indicar seu status: To Do (tarefas a serem iniciadas), Doing

(tarefas em andamento) e Done (tarefas concluídas). Essa abordagem ajuda a gerenciar o fluxo de trabalho e a identificar gargalos no processo.

Gabarito: Correto

---

6. **(CESPE / DATAPREV – 2023)** Uma das técnicas adotadas pelo Kanban para assegurar a agilidade nas entregas é limitar o trabalho em progresso.

Comentários:

Uma das técnicas fundamentais do Kanban é limitar o trabalho em progresso (Work In Progress - WIP). Essa abordagem ajuda a melhorar a eficiência e a agilidade nas entregas, evitando a sobrecarga da equipe e garantindo que o trabalho seja concluído de maneira mais rápida e eficiente. Ao limitar a quantidade de tarefas em andamento, o Kanban ajuda a identificar gargalos no fluxo de trabalho, promovendo uma melhor gestão dos recursos e foco nas prioridades.

Gabarito: Correto

---

7. **(CESPE / DATAPREV – 2023)** Kanban e Scrum são metodologias ágeis distintas e, portanto, não podem ser utilizadas simultaneamente em um mesmo projeto ou no desenvolvimento de um mesmo produto.

Comentários:

. Kanban e Scrum são metodologias ágeis distintas, mas podem ser usadas simultaneamente em um mesmo projeto. Essa abordagem é conhecida como Scrumban, que combina práticas de ambos os frameworks para aproveitar suas vantagens. Enquanto o Scrum organiza o trabalho em sprints e define papéis e cerimônias, o Kanban se concentra no fluxo contínuo de trabalho, visualização das tarefas e melhoria de processos. Juntas, essas metodologias podem aumentar a flexibilidade e eficiência do desenvolvimento.

Gabarito: Errado

---

8. **(CESPE / DATAPREV – 2023)** Na aplicação do Kanban, é necessário que se estabeleça limites de trabalhos em andamento.

Comentários:

Na aplicação do Kanban, é necessário estabelecer limites de trabalhos em andamento (Work in Progress, WIP) para melhorar o fluxo de trabalho e evitar sobrecarga da equipe. Esses limites ajudam a identificar gargalos no processo, promovendo eficiência e permitindo que a equipe se concentre em concluir tarefas antes de iniciar novas.

Gabarito: Correto

---

9. **(FGV / SEFAZ-MT - 2023)** Muitas organizações aplicam os princípios e práticas de gestão com Kanban para alcançar os objetivos de seus projetos de desenvolvimento de software. Sobre Kanban, assinale a afirmativa correta:

a) Incentiva ciclos anuais de feedbacks porque cadências de reuniões e revisões recorrentes criam muitas interrupções e impactam o tempo de entrega.

b) É um método de gestão baseado no Kaizen e que valoriza entrega de software funcional completo, sem entregas parciais, trabalhos inacabados e débitos técnico.

c) Fomenta uma cultura organizacional que privilegia o indivíduo em relação à coletividade, orientada ao cliente e sem compartilhar propósitos e metas.

d) Pressupõe que o escopo do projeto é sempre estável e o desenvolvimento de software precisa seguir de modo linear para garantir a produtividade.

e) Permite visualizar o processo e o fluxo de trabalho de projetos, programas e portfólios por meio de um conjunto de quadros interconectados.

Comentários:

(a) Errado. Kanban incentiva ciclos curtos de feedback e revisões contínuas, o que ajuda a identificar e resolver problemas rapidamente, sem esperar por ciclos anuais;

(b) Errado. Kanban é flexível e permite entregas parciais e incrementais, focando na melhoria contínua (Kaizen) e na gestão do fluxo de trabalho, mas não impõe a entrega de software funcional completo em cada iteração;

(c) Errado. Kanban promove a colaboração e o trabalho em equipe, e não privilegia o indivíduo em detrimento da coletividade. A cultura organizacional é orientada ao cliente, mas com ênfase na colaboração;

(d) Errado. Kanban é projetado para ser flexível e adaptável a mudanças no escopo do projeto, não pressupondo que o desenvolvimento de software siga um processo linear;

(e) Correto. Kanban permite visualizar o processo e o fluxo de trabalho por meio de quadros, que ajudam as equipes a monitorar o progresso e identificar gargalos, sendo aplicável a projetos, programas e portfólios.

Gabarito: Letra E

10. (CESPE / BANRISUL – 2022) O método Kanban pode ser utilizado em substituição à metodologia Scrum, mas também ambos podem ser combinados para o alcance de resultados mais eficazes.

Comentários:

Opa! O Kanban não pode ser utilizado como um substituto para o Scrum, mas os dois podem ser combinados para resultados mais eficazes.

Gabarito: Errado

---

11. (CESPE / BANRISUL – 2022) As equipes que utilizam o método Kanban não utilizam timeboxes, embora a maioria das equipes pratique uma cadência fixa de planejamento, revisão e entregas.

Comentários:

O método Kanban não é estruturado em torno de timeboxes, pois é mais flexível. No Kanban, o processo de desenvolvimento de software é dividido em etapas, que podem ser executadas de forma contínua e contínua. Isso significa que, em vez de criar um calendário de entrega e seguir um cronograma, as equipes Kanban se concentram em entregar trabalho de forma constante e incremental. Como resultado, as equipes podem responder mais rapidamente às mudanças no projeto, pois não estão limitadas por um cronograma pré-definido.

Gabarito: Correto

---

12. (CESPE / BANRISUL – 2022) O Kanban, devido à adoção dos princípios Lean, é um método ideal para utilização em projetos que adotam o Scrum; por outro lado, não se aplica a projetos tradicionais do tipo cascata.

Comentários:

Ele pode, sim, ser aplicado a projetos tradicionais do tipo cascata. O Kanban não é exclusivo para metodologias ágeis, sendo possível aplicá-lo em outros tipos de projetos. O objetivo do Kanban é ajudar os times a planejar, monitorar e gerenciar o trabalho, o que o torna uma ferramenta útil em qualquer tipo de projeto.

Gabarito: Errado

---

13. (CESPE / BANRISUL – 2022) O WIP descreve o total de trabalho que está em progresso no Kanban, podendo incluir todos os itens ou apenas aqueles selecionados para implementação.

Comentários:

O WIP (Work in Progress) descreve o total de trabalho que está em progresso no Kanban. Esta informação é usada para ajudar a equipe a tomar decisões informadas sobre quanto trabalho ela pode realizar ao mesmo tempo. O WIP pode incluir todos os itens que estão no Kanban, ou apenas aqueles itens selecionados para implementação. Isso depende da abordagem de gerenciamento de projeto adotada pela equipe.

Gabarito: Correto

---

14. (CESPE / BANRISUL – 2022) O Kanban é um método de gestão de mudanças que dá ênfase à visualização do trabalho em andamento.

Comentários:

Definição simples e precisa de Kanban! Nada a acrescentar...

Gabarito: Correto

---

15. (CESPE / BNB – 2022) Diferentemente do Scrum, o Kanban não prescreve interações com metas pré-definidas e de mesmo tamanho para a execução de atividades, como, por exemplo, as de planejamento, de desenvolvimento e de liberação.

Comentários:

Perfeito! O Kanban é uma metodologia de gerenciamento de projetos semelhante ao Scrum, mas com algumas diferenças significativas. Ele é focado na visualização do fluxo de trabalho, na limitação do trabalho em progresso e na otimização contínua. Em contraste com o Scrum, o Kanban não determina a duração de cada ciclo de trabalho. Em vez disso, os projetos são divididos em pequenas unidades de trabalho e as tarefas são desenvolvidas de forma iterativa sem metas pré-definidas. O processo de desenvolvimento é então ajustado de acordo com as necessidades da equipe.

Gabarito: Correto

---

16. (FGV / BANESTES – 2021) Observe o quadro comparativo a seguir, publicado em sites ligados ao estudo e à investigação de diferentes estratégias/metodologias para implementar um sistema ágil de desenvolvimento ou gestão de projetos.

Aspectos	X	Y
Ritmo	Sprints	Fluxo contínuo
Funções	Funções bem definidas	Sem funções necessárias
Entregas	Final de cada sprint	Entrega contínua
Mudanças	Evitar durante sprint	A qualquer momento

É correto identificar que X e Y representam, respectivamente:

a) Crystal e Scrum;
b) Extreme Programming e Crystal;
c) Kanban e Lean;
d) Lean e Extreme Programming;
e) Scrum e Kanban.

Comentários:

Usando apenas o primeiro critério, já é possível responder à questão: Sprint é típico do Scrum e Fluxo Contínuo é típico do Kanban.

Gabarito: Letra E

---

17. (CESPE / TCU – 2015) O método para a implantação de mudanças denominado Kanban não prevê papéis nem cerimônias específicas.

Comentários:

Realmente não se prevê papéis ou cerimônias no Kanban.

Gabarito: Correto

---

18. (CESPE / PROCEMPA – 2014) Kanban considera a utilização de uma sinalização ou registro visual para gerenciar o limite de atividades em andamento, indicando se um novo trabalho pode ou não ser iniciado e se o limite acordado para cada fase está sendo respeitado.

Comentários:

Esse registro visual é o famoso WIP.

Gabarito: Correto

---

19. (FGV / TJ-GO – 2014) Scrum e Kanban são metodologias de gerenciamento de projetos de software populares entre praticantes do desenvolvimento ágil. Um aspecto de divergência entre as duas metodologias é:

a) processo incremental;
b) processo iterativo;
c) uso de quadro de tarefas;
d) apresentação do estágio de desenvolvimento de uma tarefa;
e) valorização de feedback.

Comentários:

O Kanban não é necessariamente iterativo como é o Scrum! Na prática, esse assunto é rodeado de polêmicas e divergências. Já os outros itens tratam de convergências!

Gabarito: Letra B

---

20. (FGV / MPE-MS – 2013) Kanban é um dos métodos ágeis mais recentes e sofreu grande influência do movimento "Lean", surgido nos anos 1980. São práticas comuns a esse método:

a) limitar o WIP (Work In Progress) e uma visualização explícita do fluxo de trabalho.
b) integração Contínua e gerenciamento de configuração.
c) limitar o WIP (Work In Progress) e gerenciamento de configuração.
d) gerenciar o fluxo de trabalho e manter estimativas previamente definidas.
e) melhoria contínua e nunca limitar o WIP para evitar folgas no sistema de trabalho.

Comentários:

As práticas são: implemente mecanismos de feedback; gerencie e meça o fluxo de trabalho; visualize o processo; limite o WIP (Work In Progress); torne as políticas dos processos explícitas; melhore colaborativamente e com métodos científicos.

Gabarito: Letra A

---

21. (CESPE / SEDF – 2017) A técnica de Kanban é uma forma simples de visualizar o andamento das tarefas da equipe durante uma Sprint de Scrum. Nessa técnica, as tarefas são representadas por meio de pequenos papéis que indicam o que está pendente, em desenvolvimento e finalizado. Com isso, todos visualizam os gargalos e a equipe se organiza melhor, principalmente quando o projeto envolve ciclos longos de desenvolvimento.

Comentários:

Ela realmente pode ser considerada uma técnica que ajuda a visualizar o andamento do projeto – ele é muito utilizado em conjunto com o Scrum.

Gabarito: Correto

---

22. (FCC / TST – 2017) Um Analista de Sistemas do Tribunal Superior do Trabalho – TST, de modo hipotético, necessitou aplicar princípios ágeis e de controle usando elementos de três modelos, em processos de manutenção de software. Considere:

I. Dividir o cronograma em iterações time-box ou ciclos (sprints).

II. Orientar o trabalho a eventos ao invés de limite de tempo.
III. Aplicar a programação em pares, integração contínua, orientação a testes (TDD), revisão de código e todas as demais prescrições antes da implantação.

As características acima correspondem, respectivamente, a:

a) Kanban, XP e Scrum.
b) Kanban, Scrum e XP.
c) XP, Scrum e Kanban.
d) Scrum, XP e Kanban.
e) Scrum, Kanban e XP.

Comentários:

(I) Quem divide o cronograma em iterações time-box é o Scrum; (II) Quem orienta o trabalho a eventos ao invés de limite de tempo é o Kanban; (III) Quem aplica a programação em pares, TDD e revisão de código é o XP.

Gabarito: Letra E

---

23. (CESPE / STM – 2018) A implementação de um Kanban pressupõe a definição de um fluxo de trabalho pela equipe, o qual poderá ser revisto, mediante a inclusão ou a retirada de estágios, à medida que o trabalho evoluir.

Comentários:

Ele realmente pressupõe um fluxo de trabalho e não há nenhum problema em revisitá-lo com a inclusão ou exclusão de estágios com a evolução do trabalho.

Gabarito: Correto

---

24. (FCC / MPE-PE – 2018) Enquanto o processo de desenvolvimento Scrum usa sprints formais (ciclos de trabalho) com funções específicas atribuídas, o Kanban:

a) não define sprints formais nem papéis específicos para os integrantes da equipe do projeto.

b) não define ciclos formais, porém, prescreve papéis específicos para todos os integrantes da equipe do projeto.

c) define ciclos formais (sprints), porém, não define papéis específicos para os integrantes da equipe do projeto.

d) define ciclos formais de até 4 semanas e papéis específicos para os integrantes da equipe de desenvolvimento.

e) define apenas os papéis de Gerente de Projeto e Líder de Equipe, tendo o desenvolvimento pautado por ciclos de duas semanas chamados slices.

Comentários:

(a) Correto, ele não define iterações (muito menos sprints) – ou papeis específicos; (b) Errado, ele não prescreve papeis para os integrantes da equipe; (c) Errado, ele não define ciclos formais ou sprints; (d) Errado, ele não define ciclos formais; (e) Errado, ele não define nenhum papel.

Gabarito: Letra A

---

25. **(IF-RS / IF-RS – 2018)** Kanban foi criado pela Toyota com o objetivo de controlar melhor os níveis enormes de estoque em relação ao consumo real de materiais. Devido à sua eficiência, muitas empresas adotaram esse sistema para controlar tarefas das equipes do setor de Tecnologia da Informação. A respeito do Kanban, conforme visto em Dooley (2017), classifique cada uma das afirmativas abaixo como verdadeira (V) ou falsa (F) e assinale a alternativa que apresenta a sequência CORRETA, de cima para baixo:

( ) Através do quadro Kanban, compartilhado por todos, torna-se possível visualizar as tarefas com que cada membro da equipe está envolvido.

( ) Diferente do Scrum, Kanban baseia-se em iterações de tempo fixo. Os projetos são divididos em ciclos semanais denominados Sprints.

( ) Usa três ideias para influenciar um processo de desenvolvimento: trabalho em andamento (WIP), fluxo de trabalho e o custo médio financeiro.

( ) Geralmente utilizam-se post-its ou cartões de índice para representar uma tarefa no quadro Kanban.

a) V – V – F – V
b) V – F – F – V
c) F – F – V – F
d) V – V – V – V
e) F – F – V – V

Comentários:

(V) Correto, o WIP é compartilhado com todos para dar transparência; (F) Errado, ele não possui iterações nem ciclos formais ou sprints; (F) Errado, não existe a ideia de custo médio financeiro; (V) Correto, são realmente utilizados os post-its para representar tarefas.

Gabarito: Letra B

26.(CESPE / SERPRO – 2013) Kanban é um método de desenvolvimento de software que tem como uma de suas práticas o gerenciamento do fluxo de trabalho, que deve ser monitorado, medido e reportado a cada estado do fluxo.

Comentários:

Ele pode ser considerado um método de desenvolvimento de software e o fluxo de trabalho é constantemente monitorado a cada mudança, por meio de um quadro de fluxo.

Gabarito: Correto

LISTA DE QUESTÕES – DIVERSAS BANCAS

1. **(FGV / TJ-RJ – 2024)** Diversas empresas têm adotado o Kanban como ferramenta de aumento do fluxo e da produtividade no desenvolvimento de software. Considerando que esta tecnologia requer a adesão a algumas práticas fundamentais, avalie se as afirmativas a seguir são verdadeiras (V) ou falsas (F).

( ) Os quadros Kanban incorporam o princípio da visualização do trabalho que se baseia na exibição de cartões que correspondem a itens da lista de pendências do produto.

( ) Empregando o modelo de Pull a equipe puxa para seu fluxo de trabalho as pendências da lista conforme a sua capacidade se torna disponível.

( ) A imposição de limite para o número de tarefas que um time possui atualmente corresponde ao WIP (Work In Progress), e traz o benefício de aumentar o foco e, ao mesmo tempo, reduzir a mudança de contexto.

As afirmativas são, respectivamente,

a) F – V – F.
b) V – F – F.
c) F – F – V.
d) V – V – F.
e) V – V – V.

2. **(FGV / AL-SC – 2024)** A prática de Test Driven Development (TDD, ou Desenvolvimento Orientado por Testes) se relaciona com o conceito de verificação e validação e se baseia em um ciclo para garantir a qualidade do código. Entre as características do TDD, é correto o que se afirma em:

a) trata-se de desenvolvimento orientado para os comportamentos, sendo o desenvolvedor responsável por escrever os testes e validá-los de forma que eles funcionem.

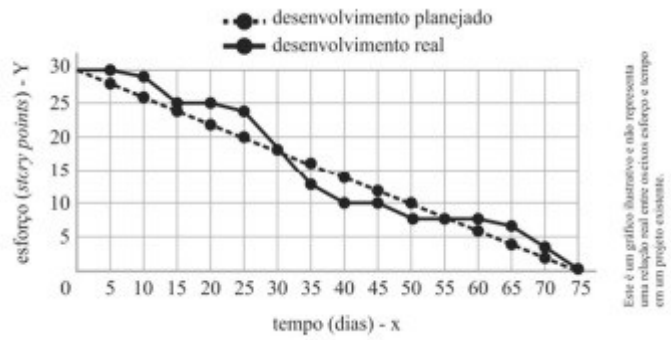
b) trata-se de desenvolvimento orientado para os testes, sendo atribuição do programador escrever como o problema deve se comportar.

c) na prática, se refere a escrever um teste automatizado durante o desenvolvimento do código de fato.

d) o ciclo do TDD se inicia com a criação de um teste, para auxiliar a codificação, segue com a realização da codificação para passar no teste, e finaliza com a eliminação das redundâncias, ao se refatorar o código.

e) no desenvolvimento de programas são intercalados testes, elicitação de requisitos e desenvolvimento de código, tendo os testes embutidos em um programa separado que os executa e invoca o sistema que está sendo testado.

3. **(CESPE / CAU-BR – 2024)** A figura a seguir descreve um Kanban de forma visual, mostrando a produtividade da equipe esforço (eixo-y) até o último dia da sprint (eixo-x), ou seja, esse Kanban liga os dias da sprint com o trabalho restante.



4. **(CESPE / MPO – 2024)** Em projetos de software, utilizam-se ferramentas de acompanhamento de equipes e tarefas entregues por meio de quadros do tipo Kanbam.

5. **(CESPE / SEFIN de Fortaleza-CE – 2023)** *To do*, *doing* e *done* são três estágios básicos do método Kanbam usados para caracterizar o andamento das atividades de um projeto de desenvolvimento de software.

6. **(CESPE / DATAPREV – 2023)** Uma das técnicas adotadas pelo Kanban para assegurar a agilidade nas entregas é limitar o trabalho em progresso.

7. **(CESPE / DATAPREV – 2023)** Kanban e Scrum são metodologias ágeis distintas e, portanto, não podem ser utilizadas simultaneamente em um mesmo projeto ou no desenvolvimento de um mesmo produto.

8. **(CESPE / DATAPREV – 2023)** Na aplicação do Kanban, é necessário que se estabeleça limites de trabalhos em andamento.

9. **(FGV / SEFAZ-MT - 2023)** Muitas organizações aplicam os princípios e práticas de gestão com Kanban para alcançar os objetivos de seus projetos de desenvolvimento de software. Sobre Kanban, assinale a afirmativa correta:

a) Incentiva ciclos anuais de feedbacks porque cadências de reuniões e revisões recorrentes criam muitas interrupções e impactam o tempo de entrega.

b) É um método de gestão baseado no Kaizen e que valoriza entrega de software funcional completo, sem entregas parciais, trabalhos inacabados e débitos técnico.

c) Fomenta uma cultura organizacional que privilegia o indivíduo em relação à coletividade, orientada ao cliente e sem compartilhar propósitos e metas.

d) Pressupõe que o escopo do projeto é sempre estável e o desenvolvimento de software precisa seguir de modo linear para garantir a produtividade.

e) Permite visualizar o processo e o fluxo de trabalho de projetos, programas e portfólios por meio de um conjunto de quadros interconectados.

10. **(CESPE / BANRISUL – 2022)** O método Kanban pode ser utilizado em substituição à metodologia Scrum, mas também ambos podem ser combinados para o alcance de resultados mais eficazes.

11. **(CESPE / BANRISUL – 2022)** As equipes que utilizam o método Kanban não utilizam timeboxes, embora a maioria das equipes pratique uma cadência fixa de planejamento, revisão e entregas.

12. **(CESPE / BANRISUL – 2022)** O Kanban, devido à adoção dos princípios Lean, é um método ideal para utilização em projetos que adotam o Scrum; por outro lado, não se aplica a projetos tradicionais do tipo cascata.

13. **(CESPE / BANRISUL – 2022)** O WIP descreve o total de trabalho que está em progresso no Kanban, podendo incluir todos os itens ou apenas aqueles selecionados para implementação.

14. **(CESPE / BANRISUL – 2022)** O Kanban é um método de gestão de mudanças que dá ênfase à visualização do trabalho em andamento.

15. **(CESPE / BNB – 2022)** Diferentemente do Scrum, o Kanban não prescreve interações com metas pré-definidas e de mesmo tamanho para a execução de atividades, como, por exemplo, as de planejamento, de desenvolvimento e de liberação.

16. **(FGV / BANESTES – 2021)** Observe o quadro comparativo a seguir, publicado em sites ligados ao estudo e à investigação de diferentes estratégias/metodologias para implementar um sistema ágil de desenvolvimento ou gestão de projetos.

Aspectos	X	Y
Ritmo	Sprints	Fluxo contínuo
Funções	Funções bem definidas	Sem funções necessárias
Entregas	Final de cada sprint	Entrega contínua
Mudanças	Evitar durante sprint	A qualquer momento

É correto identificar que X e Y representam, respectivamente:

a) Crystal e Scrum;
b) Extreme Programming e Crystal;

c) Kanban e Lean;
d) Lean e Extreme Programming;
e) Scrum e Kanban.

17. **(CESPE / TCU – 2015)** O método para a implantação de mudanças denominado Kanban não prevê papéis nem cerimônias específicas.

18. **(CESPE / PROCEMPA – 2014)** Kanban considera a utilização de uma sinalização ou registro visual para gerenciar o limite de atividades em andamento, indicando se um novo trabalho pode ou não ser iniciado e se o limite acordado para cada fase está sendo respeitado.

19. **(FGV / TJ-GO – 2014)** Scrum e Kanban são metodologias de gerenciamento de projetos de software populares entre praticantes do desenvolvimento ágil. Um aspecto de divergência entre as duas metodologias é:

a) processo incremental;
b) processo iterativo;
c) uso de quadro de tarefas;
d) apresentação do estágio de desenvolvimento de uma tarefa;
e) valorização de feedback.

20. **(FGV / MPE-MS – 2013)** Kanban é um dos métodos ágeis mais recentes e sofreu grande influência do movimento "Lean", surgido nos anos 1980. São práticas comuns a esse método:

a) limitar o WIP (Work In Progress) e uma visualização explícita do fluxo de trabalho.
b) integração Contínua e gerenciamento de configuração.
c) limitar o WIP (Work In Progress) e gerenciamento de configuração.
d) gerenciar o fluxo de trabalho e manter estimativas previamente definidas.
e) melhoria contínua e nunca limitar o WIP para evitar folgas no sistema de trabalho.

21. **(CESPE / SEDF – 2017)** A técnica de Kanban é uma forma simples de visualizar o andamento das tarefas da equipe durante uma Sprint de Scrum. Nessa técnica, as tarefas são representadas por meio de pequenos papéis que indicam o que está pendente, em desenvolvimento e finalizado. Com isso, todos visualizam os gargalos e a equipe se organiza melhor, principalmente quando o projeto envolve ciclos longos de desenvolvimento.

22. **(FCC / TST – 2017)** Um Analista de Sistemas do Tribunal Superior do Trabalho – TST, de modo hipotético, necessitou aplicar princípios ágeis e de controle usando elementos de três modelos, em processos de manutenção de software. Considere:

I. Dividir o cronograma em iterações time-box ou ciclos (sprints).
II. Orientar o trabalho a eventos ao invés de limite de tempo.
III. Aplicar a programação em pares, integração contínua, orientação a testes (TDD), revisão de código e todas as demais prescrições antes da implantação.

As características acima correspondem, respectivamente, a:

a) Kanban, XP e Scrum.
b) Kanban, Scrum e XP.
c) XP, Scrum e Kanban.
d) Scrum, XP e Kanban.
e) Scrum, Kanban e XP.

23. **(CESPE / STM – 2018)** A implementação de um Kanban pressupõe a definição de um fluxo de trabalho pela equipe, o qual poderá ser revisto, mediante a inclusão ou a retirada de estágios, à medida que o trabalho evoluir.

24. **(FCC / MPE-PE – 2018)** Enquanto o processo de desenvolvimento Scrum usa sprints formais (ciclos de trabalho) com funções específicas atribuídas, o Kanban:

a) não define sprints formais nem papéis específicos para os integrantes da equipe do projeto.

b) não define ciclos formais, porém, prescreve papéis específicos para todos os integrantes da equipe do projeto.

c) define ciclos formais (sprints), porém, não define papéis específicos para os integrantes da equipe do projeto.

d) define ciclos formais de até 4 semanas e papéis específicos para os integrantes da equipe de desenvolvimento.

e) define apenas os papéis de Gerente de Projeto e Líder de Equipe, tendo o desenvolvimento pautado por ciclos de duas semanas chamados slices.

25. **(IF-RS / IF-RS – 2018)** Kanban foi criado pela Toyota com o objetivo de controlar melhor os níveis enormes de estoque em relação ao consumo real de materiais. Devido à sua eficiência, muitas empresas adotaram esse sistema para controlar tarefas das equipes do setor de Tecnologia da Informação. A respeito do Kanban, conforme visto em Dooley (2017), classifique cada uma das afirmativas abaixo como verdadeira (V) ou falsa (F) e assinale a alternativa que apresenta a sequência CORRETA, de cima para baixo:

( ) Através do quadro Kanban, compartilhado por todos, torna-se possível visualizar as tarefas com que cada membro da equipe está envolvido.

( ) Diferente do Scrum, Kanban baseia-se em iterações de tempo fixo. Os projetos são divididos em ciclos semanais denominados Sprints.

( ) Usa três ideias para influenciar um processo de desenvolvimento: trabalho em andamento (WIP), fluxo de trabalho e o custo médio financeiro.

( ) Geralmente utilizam-se post-its ou cartões de índice para representar uma tarefa no quadro Kanban.

a) V – V – F – V
b) V – F – F – V
c) F – F – V – F
d) V – V – V – V
e) F – F – V – V

26. (CESPE / SERPRO – 2013) Kanban é um método de desenvolvimento de software que tem como uma de suas práticas o gerenciamento do fluxo de trabalho, que deve ser monitorado, medido e reportado a cada estado do fluxo.

GABARITO – DIVERSAS BANCAS

1. LETRA E
2. LETRA D
3. ERRADO
4. CORRETO
5. CORRETO
6. CORRETO
7. ERRADO
8. CORRETO
9. LETRA E
10. ERRADO
11. CORRETO
12. ERRADO
13. CORRETO
14. CORRETO
15. CORRETO
16. LETRA E
17. CORRETO
18. CORRETO
19. LETRA B
20. LETRA A
21. CORRETO
22. LETRA E
23. CORRETO
24. LETRA A
25. LETRA B
26. CORRETO

TEST DRIVEN DEVELOPMENT (TDD)

Conceitos Básicos

INCIDÊNCIA EM PROVA: MÉDIA

O Test-Driven Development (TDD) é uma abordagem de desenvolvimento de software em que se intercalam testes e desenvolvimento de código. Essencialmente, você desenvolve um código de forma incremental, em conjunto com um teste para esse incremento. Você não caminha para o próximo incremento até que o código desenvolvido passe no teste. O desenvolvimento dirigido a testes foi apresentado como parte dos métodos ágeis, como o XP.

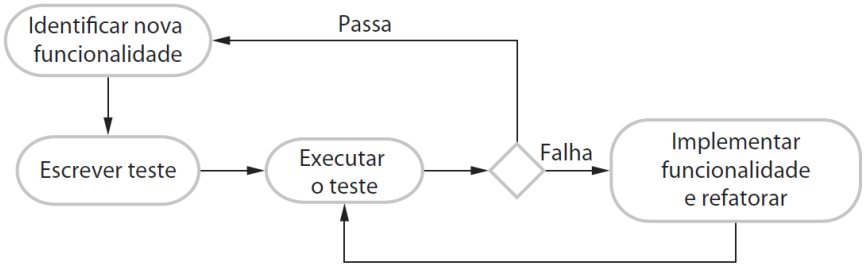
Por outro lado, ele também pode ser utilizado em processos de desenvolvimento dirigido a planos. Trata-se de uma abordagem que **se baseia na repetição de um ciclo de desenvolvimento curto focado em testes unitários**. A ideia fundamental dessa abordagem consiste em escrever o teste, encontrar uma falha nesse mesmo teste e depois refatorá-lo. Vamos agora ver as etapas do processo de desenvolvimento dirigido a testes:

ETAPA	DESCRIÇÃO
1	Você começa identificando o incremento de funcionalidade necessário. Este, normalmente, deve ser pequeno e implementável em poucas linhas de código.
2	Você escreve um teste para essa funcionalidade e o implementa como um teste automatizado. Isso significa que o teste pode ser executado e relatará se passou ou falhou.
3	Você, então, executa o teste, junto com todos os outros testes implementados. Inicialmente, você não terá implementado a funcionalidade, logo o novo teste falhará. Isso é proposital, pois mostra que o teste acrescenta algo ao conjunto de testes.
4	Você, então, implementa a funcionalidade e executa novamente o teste. Isso pode envolver a refatoração do código existente para melhorá-lo e adicionar um novo código sobre o que já está lá.
5	Depois que todos os testes forem executados com sucesso, você caminha para implementar a próxima parte da funcionalidade.

Como o código é desenvolvido em incrementos muito pequenos, você precisa ser capaz de executar todos os testes cada vez que adicionar funcionalidade ou refatorar o programa. Dessa forma, os testes são embutidos em um programa separado que os executa e invoca o sistema que está sendo testado. Usando essa abordagem, é possível rodar centenas e centenas de testes separados em poucos segundos.

Um argumento forte a favor do desenvolvimento dirigido a testes é que ele ajuda os programadores a clarear suas ideias sobre o que um segmento de código supostamente deve fazer**. Para escrever um teste, você precisa entender a que ele se destina, e como esse entendimento faz que seja**

mais fácil escrever o código necessário. Certamente, se você tem conhecimento ou compreensão incompleta, o desenvolvimento dirigido a testes não ajudará.



Se você não sabe o suficiente para escrever os testes, não vai desenvolver o código necessário. Por exemplo: se seu cálculo envolve divisão, você deve verificar se não está dividindo o número por zero. Se você se esquecer de escrever um teste para isso, então o código para essa verificação nunca será incluído no programa. Além de um melhor entendimento do problema, outros benefícios do desenvolvimento dirigido a testes são:

BENEFÍCIOS	DESCRIÇÃO
COBERTURA DE CÓDIGO	Em princípio, todo segmento de código que você escreve deve ter pelo menos um teste associado. Assim, você pode ter certeza de que todo o código no sistema foi realmente executado. Cada código é testado enquanto está sendo escrito; assim, os defeitos são descobertos no início do processo de desenvolvimento.
TESTE DE REGRESSÃO	Um conjunto de testes é desenvolvido de forma incremental enquanto um programa é desenvolvido. Você sempre pode executar testes de regressão para verificar se as mudanças no programa não introduziram novos bugs.
DEPURAÇÃO SIMPLIFICADA	Quando um teste falha, a localização do problema deve ser óbvia. O código recém-escrito precisa ser verificado e modificado. Você não precisa usar as ferramentas de depuração para localizar o problema. Alguns relatos de uso de desenvolvimento dirigido a testes sugerem que, em desenvolvimento dirigido a testes, quase nunca é necessário usar um sistema automatizado de depuração.
DOCUMENTAÇÃO DE SISTEMA	Os testes em si mesmos agem como uma forma de documentação que descreve o que o código deve estar fazendo. Ler os testes pode tornar mais fácil a compreensão do código.

Um dos benefícios mais importantes de desenvolvimento dirigido a testes é que ele reduz os custos dos testes de regressão. O teste de regressão envolve a execução de conjuntos de testes que tenham sido executados com sucesso, após as alterações serem feitas em um sistema. Ele também verifica se essas mudanças não introduziram novos bugs no sistema e se o novo código interage com o código existente conforme o esperado.

O teste de regressão é muito caro e geralmente impraticável quando um sistema é testado manualmente, pois os custos com tempo e esforço são muito altos. Em tais situações, você

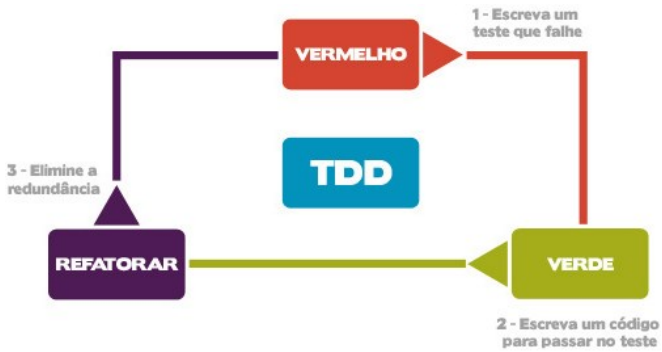
precisa tentar escolher os testes mais relevantes para executar novamente, e é fácil perder testes importantes. No entanto, testes automatizados – fundamentais para o desenvolvimento *test-first* – reduzem drasticamente os custos com testes de regressão.

Os testes existentes podem ser executados novamente de forma rápida e barata. **Após se fazer uma mudança para um sistema em desenvolvimento *test-first*, todos os testes existentes devem ser executados com êxito antes de qualquer funcionalidade ser adicionada.** Como um programador, você precisa ter certeza de que a nova funcionalidade não tenha causado ou revelado problemas com o código existente.

O desenvolvimento dirigido a testes é de maior utilidade no desenvolvimento de softwares novos, em que a funcionalidade seja implementada no novo código ou usando bibliotecas-padrão já testadas. **Se você estiver reusando componentes de código ou sistemas legados grandes, você precisa escrever testes para esses sistemas como um todo.** O desenvolvimento dirigido a testes também pode ser ineficaz em sistemas multi-threaded.

As *threads* diferentes podem ser intercalados em tempos diferentes, em execuções diferentes, e isso pode produzir resultados diferentes. Se você usa o desenvolvimento dirigido a testes, ainda precisa de um processo de teste de sistema para validar o sistema, isto é, verificar se atende aos requisitos dos stakeholders. O teste de sistema também testa o desempenho, a confiabilidade, e verifica se o sistema não faz coisas que não deveria, como produzir resultados indesejados etc.

O desenvolvimento dirigido a testes revelou-se uma abordagem de sucesso para projetos de pequenas e médias empresas. Geralmente, os programadores que adotaram essa abordagem estão satisfeitos com ela e acham que é uma maneira mais produtiva de desenvolver softwares. Em alguns experimentos, foi mostrado que essa abordagem gera melhorias na qualidade do código. Bem, agora vamos falar sobre o ciclo vermelho, verde e refatoração.



ETAPA	DESCRIÇÃO
VERMELHO (RED)	Escreva um teste que falha: casos de teste são escritos sem que exista o recurso a ser testado, levando o teste a falhar;
VERDE (GREEN)	Escreva código para passar no teste: o recurso é implementado com o mínimo de código necessário para que passe no teste.

REFATORAR (REFACTOR)	Elimine redundâncias: o código que implementa o recurso é refinado e melhorado, sem que seja adicionada novas funcionalidades.

Vamos lá! Nós sabemos que – para cada parte da aplicação – adiciona-se um teste escrito antes mesmo do desenvolvimento do código em si. *Por que?* **Porque eles podem ajudar a reduzir riscos de possíveis problemas no código.** Executamos o teste e ele... falha! Ele deve necessariamente falhar! *Por que?* Ora, porque ele é o primeiro teste e você nem criou a funcionalidade ainda, logo ele não irá funcionar!

Então nós adicionamos uma nova funcionalidade ao sistema apenas para que ele passe no teste e execute novamente (agora ele deve passar no teste). **Então, nós adicionamos um novo teste e rodamos o teste anterior e esse novo teste.** Se algum deles falhar, modifica-se o código da funcionalidade e rodam-se todos os testes novamente, e assim por diante – nós já vimos aquela imagem anterior que mostra como tudo isso funciona...

Galera, vocês percebem que o feedback sobre a nova funcionalidade ocorre de maneira bem rápido? **Além disso, cria-se um código mais limpo, visto que o código para passar nos testes deve ser bastante simples.** Há mais segurança na correção de eventuais bugs; aumenta-se a produtividade, visto que se perde menos tempo com depuradores; e o código se torna mais flexível, menos acoplado e mais coeso.

Nós podemos afirmar que, em geral, utilizam-se testes unitários, testes de integração ou testes de aceitação – sendo os dois primeiros os mais comuns. **Algumas ferramentas que podem ser utilizadas para implementar o processo de desenvolvimento orientado a testes:** JUnit, TesteNG, PHPUnit, SimpleTest, NUnit, Jasmine, CUnit, PyUnit, etc. Pessoal, agora um detalhe que nós passamos direto sobre a abordagem de desenvolvimento...

O TEST DRIVEN DEVELOPMENT (TDD) NÃO É UMA ABORDAGEM PARA REALIZAR TESTES - TRATA-SE DE UMA ABORDAGEM PARA DESENVOLVER SOFTWARES. ELA PODE EVENTUALMENTE SER CONSIDERADA TAMBÉM UMA TÉCNICA DE PROGRAMAÇÃO!

Professor, uma curiosidade: isso já caiu em alguma prova discursiva? **Sim, galera... o enunciado dessa prova requisitava ao aluno informar as vantagens do emprego do TDD em relação a outras metodologias ágeis.** Poderíamos responder essa pergunta afirmando que o software desenvolvido, em geral, apresenta maior qualidade, na medida em que é implementado direcionado às expectativas do cliente.

Poderíamos dizer também que há a possibilidade de se testar todo o código desenvolvido, o que oferece maior confiabilidade ao sistema. Por fim, em geral, o código é mais modularizado, flexível e extensível, visto que a metodologia requer que os desenvolvedores imaginem o software como pequenas unidades[1] que podem ser reescritas, desenvolvidas e testadas de forma independente e integradas em momento posterior.

Essa mesma prova perguntava também quais são os princípios da Metodologia Extreme Programming (XP) apoiados pelo TDD. Uma resposta adequada poderia afirmar que o XP apresenta diversas práticas que podem ser relacionadas com o TDD. *Qual é a mais óbvia?* **A mais óbvia é o *Test-First* (Teste Primeiro), ratificando a característica básica recomendada veementemente pelo desenvolvimento orientado a testes.**

O TDD pode apoiar esse princípio por fornecer detalhes para a realização dos testes de unidade e de funcionalidade, que são importantes e necessários. Ademais, o desenvolvimento orientado a testes apresenta relação intrínseca com a refatoração, tendo em vista que confere ao programador maior segurança para identificar e remover o código duplicado, e permite, assim, a melhoria contínua do programa. Vamos resumir as principais características do TDD:

CARACTERÍSTICAS DO TDD	DESCRIÇÃO
ORIENTADO A TESTES	O TDD coloca o teste no centro do processo de desenvolvimento, guiando a escrita do código. Cada nova funcionalidade é conduzida por um ou mais testes que definem o comportamento esperado.
PEQUENOS PASSOS	TDD promove a implementação em pequenos passos iterativos. Cada ciclo de desenvolvimento é curto, envolvendo a escrita de um pequeno teste, o código correspondente e a refatoração.
ALTA COBERTURA	Como os testes são escritos antes do código, TDD naturalmente resulta em uma alta cobertura de testes automatizados, o que ajuda a detectar regressões e garantir que o código funcione conforme esperado.
FEEDBACK IMEDIATO	O TDD oferece feedback imediato sobre a correção do código. Se um teste falha, o desenvolvedor pode corrigir o problema imediatamente, minimizando o tempo entre a introdução de um erro e sua correção.
DOCUMENTAÇÃO VIVA	Os testes servem como documentação executável do comportamento do sistema. Eles descrevem o que o código deve fazer de forma precisa e concisa, e essa "documentação" é sempre atualizada porque os testes são executados continuamente.
FACILITA REFATORAÇÃO SEGURA	A prática de TDD promove um design de código mais limpo e modular, já que o desenvolvedor refatora o código regularmente com a segurança de que os testes irão capturar qualquer erro introduzido.
PROJETO GUIADO POR TESTES	TDD influencia positivamente o design do software, encorajando a criação de código coeso e de baixo acoplamento. Escrever testes primeiro ajuda os desenvolvedores a pensar na interface e na responsabilidade das classes antes de implementá-las.

[1] Atenção: apesar de serem tipicamente realizados testes de unidade, não é obrigatória a utilização desse tipo de teste.

MAIOR QUALIDADE DE CÓDIGO	A prática constante de escrever testes e refatorar resulta em um código de maior qualidade, com menos bugs e mais fácil de manter e evoluir.
REDUÇÃO DE BUGS DE PRODUÇÃO	Com o TDD, muitos erros são capturados e corrigidos durante o desenvolvimento, o que resulta em uma menor probabilidade de bugs aparecerem em produção.
MELHORIA NA CONFIANÇA	Como os testes automatizados garantem que o código funciona conforme esperado, os desenvolvedores têm mais confiança em modificar, refatorar ou estender o código sem introduzir novos erros.
AUMENTO DA MANUTENI-BILIDADE	O código desenvolvido com TDD tende a ser mais modular e de fácil manutenção, uma vez que os testes forçam o desenvolvedor a pensar em como dividir o código em partes bem definidas e testáveis.
ASSERÇÕES DE VERIFICAÇÃO	Cada caso de teste deve incluir asserções que verifiquem se o comportamento do código está conforme o esperado. As asserções são fundamentais para garantir que o teste valide corretamente o comportamento do código.
TESTES DE INTEGRAÇÃO	Testes de integração são utilizados para validar recursos complexos, como acesso a dados, onde múltiplos componentes ou sistemas interagem. Esses testes garantem que as partes integradas funcionem corretamente juntas.
INDEPENDÊNCIA DOS TESTES	A independência dos testes é um princípio fundamental. Cada teste deve ser executado isoladamente, sem dependências entre eles. Comentários sobre quais testes foram criados não substituem a necessidade de garantir que os testes não se afetam mutuamente.

Galera, nós temos também uma variação chamada Acceptance Test-Driven Development (ATDD). **Ele é método ágil de desenvolvimento de software que se baseia na comunicação entre clientes do negócio, desenvolvedores e testadores.** Diferente do Test-Driven Development (TDD), que se concentra em testes unitários, o ATDD se concentra em testes de aceitação que validam se o sistema atende aos critérios de aceitação definidos pelo cliente ou pelo usuário final.

Ele promove uma estreita colaboração entre todas as partes interessadas (stakeholders). Isso inclui desenvolvedores, testadores, analistas de negócios e clientes que trabalham juntos para definir e escrever os testes de aceitação antes que o desenvolvimento comece. Essa colaboração garante que todos tenham um entendimento comum do que precisa ser construído. Os testes de aceitação servem como uma forma de documentação viva e executável.

Eles descrevem o comportamento esperado do sistema e podem ser usados para verificar automaticamente se o software atende aos requisitos acordados. Além disso, são diretamente derivados dos requisitos funcionais e não funcionais do sistema. Isso garante que o software entregue esteja em conformidade com as especificações e expectativas do cliente. Outra característica importante é o seu feedback imediato.

Como os testes são escritos antes do desenvolvimento, o ATDD oferece feedback imediato sobre a conformidade do software com os requisitos do cliente durante o processo de desenvolvimento. Isso permite que problemas e mal-entendidos sejam identificados e corrigidos mais cedo. Ao validar os critérios de aceitação antes de começar a codificar, o ATDD ajuda a reduzir a quantidade de retrabalho causado por mal-entendidos sobre os requisitos.

Embora similar ao TDD, o ciclo do ATDD adapta o ciclo Red-Green-Refactor para se concentrar em testes de aceitação. O ciclo pode ser descrito como:

ETAPA	DESCRIÇÃO
RED	Escrever um teste de aceitação que inicialmente falha, porque a funcionalidade ainda não foi implementada.
GREEN	Implementar o código necessário para fazer o teste passar.
REFACTOR	Melhorar o código enquanto se certifica de que todos os testes de aceitação continuam a passar.

O uso do ATDD resulta em um software de maior qualidade, pois os requisitos do cliente são continuamente validados ao longo do processo de desenvolvimento. A confiança na qualidade do software também é aumentada, pois os testes de aceitação garantem que o sistema se comporta conforme o esperado. A tabela exibida a seguir apresenta as principais características do ATDD e a tabela seguinte nos mostra as diferenças para o TDD:

CARACTERÍSTICAS DO ATDD	DESCRIÇÃO
FOCO EM TESTES DE ACEITAÇÃO	ATDD envolve a escrita de testes de aceitação antes do desenvolvimento, garantindo que os critérios do cliente sejam atendidos.
COLABORAÇÃO ENTRE EQUIPES	Promove a colaboração entre desenvolvedores, testadores e stakeholders para definir os critérios de aceitação.
TESTES BASEADOS EM REQUISITOS	Os testes de aceitação são diretamente derivados dos requisitos funcionais, garantindo que o software entregue esteja alinhado com as expectativas do cliente.
DOCUMENTAÇÃO EXECUTÁVEL	Os testes de aceitação servem como documentação viva e executável, descrevendo o comportamento esperado do sistema.
FEEDBACK IMEDIATO	Oferece feedback imediato sobre a conformidade do software com os requisitos do cliente durante o desenvolvimento.
CICLO ADAPTADO	Similar ao TDD, mas com foco em garantir que o software atenda aos critérios de aceitação antes de refatorar o código.
ENGAJAMENTO DE STAKEHOLDERS	Incentiva o envolvimento dos stakeholders no processo de desenvolvimento, garantindo que o produto final atenda às suas necessidades.
QUALIDADE DO PRODUTO	Ao focar nos testes de aceitação desde o início, ATDD ajuda a detectar problemas cedo, resultando em um produto de maior qualidade.
REDUÇÃO DE RETRABALHO	Ao validar os critérios de aceitação antes do desenvolvimento, minimiza a necessidade de retrabalho causado por mal-entendidos nos requisitos.
VALIDAÇÃO DE REQUISITOS	Ajuda a validar e clarificar os requisitos antes de iniciar o desenvolvimento, garantindo que todos tenham o mesmo entendimento sobre as expectativas.

TDD (TEST-DRIVEN DEVELOPMENT)	ATDD (ACCEPTANCE TEST-DRIVEN DEVELOPMENT)
O foco principal do TDD está nos testes unitários. O desenvolvedor escreve pequenos testes automatizados antes de escrever o código correspondente. O objetivo é garantir que cada unidade de código (geralmente uma função ou método) funcione corretamente em um nível granular.	O ATDD se concentra nos testes de aceitação, que validam o comportamento do sistema em um nível mais alto, assegurando que o software atende aos requisitos e critérios de aceitação definidos pelos stakeholders, incluindo clientes, analistas de negócios e desenvolvedores.
Os testes em TDD são geralmente limitados a uma unidade de código individual, como um método ou classe. Eles são escritos pelos desenvolvedores para verificar a correção funcional de pequenas partes do sistema.	Os testes de ATDD abrangem o sistema como um todo ou grandes partes dele, focando no comportamento e na interação entre os componentes do sistema. Esses testes validam se a aplicação atende às necessidades e expectativas do usuário final.
A prática de TDD é geralmente realizada exclusivamente pelos desenvolvedores. O ciclo de escrita de testes, codificação e refatoração é iterado pelos programadores durante o desenvolvimento.	ATDD envolve uma colaboração mais ampla entre desenvolvedores, testadores, analistas de negócios e clientes. Todos trabalham juntos para definir os critérios de aceitação e os testes que garantem que o sistema está alinhado com as expectativas do cliente.
O objetivo do TDD é criar um código que funcione corretamente desde o nível mais baixo, permitindo um design de software mais limpo, modular e testável. Ele também visa capturar erros no nível mais granular possível.	O objetivo do ATDD é garantir que o sistema como um todo funcione conforme esperado pelos usuários finais e que ele atenda aos requisitos de negócios. ATDD valida que o software satisfaz as necessidades e critérios de aceitação antes de ser considerado "pronto".
Os testes criados durante o TDD atuam como uma forma de documentação técnica do comportamento das unidades de código. Eles ajudam outros desenvolvedores a entender como o código foi projetado para funcionar.	Os testes de aceitação no ATDD servem como documentação executável do sistema. Eles descrevem o comportamento esperado do sistema de uma maneira que é compreensível tanto para técnicos quanto para stakeholders não técnicos.
O ciclo de TDD (Red-Green-Refactor) é rápido e focado, proporcionando feedback imediato sobre pequenas porções de código. Isso ajuda a manter o desenvolvimento alinhado com o design desejado desde o início.	O ATDD também oferece feedback, mas em um nível mais alto, garantindo que o software em desenvolvimento está de acordo com os requisitos do usuário. O feedback no ATDD é obtido antes mesmo do início do desenvolvimento real, durante a fase de especificação dos requisitos.
Os testes são escritos pelos desenvolvedores, que também são responsáveis por escrever o código para passar esses testes.	Os testes são definidos em colaboração entre desenvolvedores, testadores e stakeholders (como clientes ou analistas de negócios), e podem ser escritos por desenvolvedores ou testadores.

Em suma: TDD é focado em garantir que as pequenas partes do código (unidades) funcionem corretamente, através da escrita de testes unitários antes do desenvolvimento do código correspondente. Ele é mais técnico e orientado para a implementação. ATDD é focado em garantir que o software como um todo atende aos requisitos do cliente, através da escrita de testes de aceitação antes do desenvolvimento. Ele é mais colaborativo e orientado para o negócio.

QUESTÕES COMENTADAS – DIVERSAS BANCAS

1. **(CESGRANRIO / IPEA – 2024)** Uma gerente de testes de software propôs a seu time de desenvolvimento que começasse a aplicar a abordagem Test Driven Development (TDD). É uma das características principais dessa abordagem iniciar o desenvolvimento de testes:

   a) antes de implementar alguma funcionalidade em si.
   b) durante o período de homologação.
   c) após as funcionalidades serem construídas.
   d) quando a primeira leva de funcionalidades planejadas forem codificadas em algum sprint.
   e) pelos testes de interface automatizado, seguidos pelos testes unitários.

Comentários:

(a) Correto. No Test Driven Development (TDD), os testes são escritos antes da implementação das funcionalidades. Isso garante que o código seja desenvolvido para passar nos testes desde o início;

(b) Errado. O período de homologação ocorre após a implementação e testes iniciais, e não é o foco do TDD, que deve iniciar antes da implementação;

(c) Errado. Escrever testes após a construção das funcionalidades é a abordagem tradicional de desenvolvimento, não a abordagem TDD;

(d) Errado. No TDD, os testes são criados antes de qualquer funcionalidade ser codificada, não após uma leva de funcionalidades;

(e) Errado. A abordagem TDD prioriza a escrita de testes unitários antes da implementação das funcionalidades, não necessariamente começando pelos testes de interface automatizados.

Gabarito: Letra A

---

2. **(CESPE / INPI – 2024)** O desenvolvimento dirigido por testes (TDD) é modelado em três estados: vermelho, verde e refatorar. Um exemplo da ação de refatoração é a simulação do comportamento dos componentes que interagem com a unidade de teste que está falhando.

Comentários:

No desenvolvimento dirigido por testes (TDD), os três estados são: vermelho (escrever um teste que falha), verde (escrever o código mínimo necessário para fazer o teste passar) e refatorar (melhorar o código mantendo os testes passando). A simulação do comportamento dos componentes que interagem com a unidade de teste que está falhando é uma prática de criação de

mocks ou stubs, usada para isolar a unidade de teste, e não um exemplo de refatoração. Refatoração envolve melhorar a estrutura do código sem alterar seu comportamento externo.

Gabarito: Errado

---

3. **(CESPE / CNPq – 2024)** O TDD é uma tendência que enfatiza o projeto de casos de teste antes da criação do código fonte e se caracteriza como parte do modelo ágil de desenvolvimento de software.

Comentários:

Test-Driven Development (TDD) é uma prática que enfatiza a criação de casos de teste antes do desenvolvimento do código-fonte. É uma abordagem iterativa que faz parte das metodologias ágeis de desenvolvimento de software, promovendo um ciclo de feedback rápido e garantindo que o código seja continuamente testado e refinado.

Gabarito: Correto

---

4. **(CESPE / CNPq – 2024)** No TDD, o teste deve ser criado com o objetivo de fazer o segmento de código falhar, gerando-se um processo iterativo que permite a submissão de muitas subfunções simultaneamente, o que confere uma agilidade significativa ao processo.

Comentários:

Test-Driven Development (TDD) é uma prática que enfatiza a criação de casos de teste antes do desenvolvimento do código-fonte. É uma abordagem iterativa que faz parte das metodologias ágeis de desenvolvimento de software, promovendo um ciclo de feedback rápido e garantindo que o código seja continuamente testado e refinado.

Gabarito: Errado

---

5. **(CESPE / AGER-MT – 2023)** Assinale a opção que corresponde ao método de teste de software adotado, sob a perspectiva do desenvolvedor, a partir de casos de teste do código, escritos em linguagem técnica, para testar as funcionalidades antes da implementação da solução desenvolvida:

a) unit testing
b) TDD
c) BDD
d) ATDD
e) teste de caixa preta

Comentários:

(a) Errado. Unit testing (Teste de Unidade) é o teste de unidades individuais do código (como funções ou métodos), mas não necessariamente envolve escrever testes antes da implementação da funcionalidade.

(b) Correto. TDD (Test-Driven Development) é uma prática de desenvolvimento onde os desenvolvedores escrevem casos de teste antes de implementar a funcionalidade correspondente. Esses testes são escritos em linguagem técnica e guiam a implementação do código.

(c) Errado. BDD (Behavior-Driven Development) é uma abordagem de desenvolvimento que estende o TDD, focando na colaboração entre desenvolvedores, QA e não técnicos, utilizando uma linguagem natural para descrever o comportamento esperado.

(d) Errado. ATDD (Acceptance Test-Driven Development) é semelhante ao TDD, mas envolve stakeholders na criação dos testes de aceitação antes do desenvolvimento, focando na validação do comportamento do sistema do ponto de vista do usuário.

(e) Errado. Teste de caixa preta (Blackbox Testing) é uma técnica de teste que verifica a funcionalidade do software sem se preocupar com a estrutura interna do código, focando apenas nas entradas e saídas.

Gabarito: Letra B

---

6. **(CESPE / TC-DF – 2023)** Na etapa de refactor do processo TDD, parte-se do pressuposto de que os testes tenham passado nas fases anteriores, o que permite que o código seja aprimorado sem a preocupação de duplicações de código.

Comentários:

A questão é polêmica, mas a banda justificou o gabarito:

"É neste momento que retiramos duplicidade, renomeamos variáveis, extraímos métodos, extraímos classes, extraímos interfaces, usamos algum padrão conhecido, etc. É neste momento que podemos deixar o nosso código simples e claro e o melhor de tudo: Funcional. Temos um teste que indicará qualquer passo errado que podemos dar ao melhorar o código"o.

Gabarito: Errado

---

7. **(CESPE / EMPREL – 2023)** Ao adotar uma prática ágil para a criação de um software, seu desenvolvedor optou pela implementação com qualidade de uma funcionalidade do sistema; para isso, escreveu um caso de teste automatizado, com base nos requisitos especificados, e realizou testes de unidade em uma linguagem similar à usada no desenvolvimento da funcionalidade. Da situação hipotética precedente infere-se que a prática adotada pelo desenvolvedor está associada ao:

a) desenvolvimento orientado por comportamento (BDD).
b) gerenciamento de produtos com Scrum.
c) desenvolvimento guiado por testes (TDD).
d) desenvolvimento guiado por testes de aceitação (ATDD).
e) gerenciamento de produtos com Kanban.

Comentários:

(a) Errado. O Desenvolvimento Orientado por Comportamento (BDD) foca em descrever o comportamento esperado do software em uma linguagem que todos os stakeholders possam entender, geralmente com cenários de aceitação claros e exemplos. A questão não menciona o uso de cenários compreensíveis por não técnicos;

(b) Errado. O Gerenciamento de Produtos com Scrum é uma metodologia ágil para gerenciamento de projetos que utiliza sprints e eventos do Scrum, como reuniões diárias e retrospectivas. Não foca especificamente na criação de testes automatizados antes do desenvolvimento;

(c) Correto. O Desenvolvimento Guiado por Testes (TDD) é uma prática de desenvolvimento ágil onde os desenvolvedores escrevem testes de unidade antes da implementação do código funcional. O processo inclui a escrita de um teste que inicialmente falha, a implementação do código para passar o teste e a refatoração do código, se necessário. Isso está alinhado com a descrição fornecida;

(d) Errado. O Desenvolvimento Guiado por Testes de Aceitação (ATDD) envolve escrever testes de aceitação com base em requisitos, geralmente com a participação de clientes e stakeholders. Foca em garantir que a funcionalidade atenda aos requisitos de aceitação. A questão menciona testes de unidade, não de aceitação;

e) Errado: O Gerenciamento de Produtos com Kanban é uma abordagem ágil que visa melhorar o fluxo de trabalho visualizando e gerenciando tarefas através de um quadro Kanban. Não envolve diretamente a prática de escrita de testes automatizados antes da implementação.

Gabarito: Letra C

---

8. **(FGV / SEFAZ-MG – 2023)** Você entrou para um projeto novo, já em andamento, no qual a metodologia que a equipe do projeto segue é a de definir e escrever testes de software a partir das regras de negócio antes mesmo de implementar as funcionalidades propostas. Assinale a opção que indica o nome desse processo de desenvolvimento de software:

a) DDD
b) TDD
c) BDD
d) XP

e) Scrum

Comentários:

(a) Errado. DDD (Domain-Driven Design) é uma abordagem de desenvolvimento que foca em modelar o software de acordo com o domínio do problema, não necessariamente relacionada à escrita de testes antes do desenvolvimento;

(b) Correto. TDD (Test-Driven Development) é a prática de escrever testes de software antes da implementação das funcionalidades, garantindo que o código seja desenvolvido para passar nos testes;

(c) Errado. BDD (Behavior-Driven Development) é uma prática semelhante ao TDD, mas foca em descrever o comportamento esperado do software em linguagem natural, geralmente envolvendo colaboração entre desenvolvedores e não-desenvolvedores;

(d) Errado. XP (Extreme Programming) é uma metodologia ágil que inclui práticas como TDD, mas não é exclusivamente sobre escrever testes antes do desenvolvimento;

(e) Errado. Scrum é uma metodologia ágil para gerenciamento de projetos, não específica para a prática de escrever testes antes do desenvolvimento.

Gabarito: Letra B

---

9. **(FGV / BB – 2023)** O desenvolvimento orientado a testes (TDD) é um processo que se baseia na repetição em ciclos de desenvolvimento curtos. Ele é baseado no conceito test-first oriundo da programação extrema (XP) que incentiva o design simples com alto nível de confiança. O procedimento que conduz este ciclo é denominado:

a) refatoração vermelho-verde.
b) documentação executável.
c) testes da caixa-branca.
d) testes da caixa-preta.
e) mocking up.

Comentários:

(a) Correto. Trata-se de uma técnica de desenvolvimento ágil em que um teste falha inicialmente (red), é feito o mínimo de código para passar no teste (green), e, em seguida, o código é refatorado para melhorar a qualidade sem alterar o comportamento externo;

(b) Errado. Documentação Executável é a prática de criar documentação que também serve como testes automatizados, garantindo que a descrição funcional do software seja verificável e alinhada com o comportamento real do sistema;

(c) Errado. Testes de Caixa Branca são testes de software que verificam a lógica interna, a estrutura e o funcionamento do código, permitindo que os testadores utilizem seu conhecimento sobre a implementação para criar casos de teste;

(d) Errado. Testes de Caixa Preta são testes de software que avaliam a funcionalidade do aplicativo sem examinar seu código-fonte, focando nas entradas e saídas de acordo com as especificações de requisitos;

(e) Errado. Mocking Up é a prática de criar versões simuladas de componentes ou sistemas para testar funcionalidades isoladamente, permitindo que os desenvolvedores avaliem o comportamento do sistema sem depender de integrações externas.

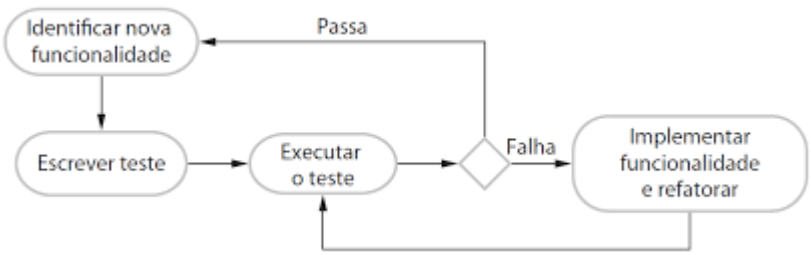
Gabarito: Letra A

---

10. (FGV / Senado Federal – 2022) Você foi contratado para liderar uma equipe de DevOps. Um dos objetivos da sua liderança é aumentar a velocidade das entregas e a qualidade de novos recursos das aplicações utilizando o desenvolvimento orientado a testes. Assinale a opção que apresenta a ordem que descreve o ciclo de desenvolvimento orientado a testes.

a) Refatorar - > Escrever um código funcional
b) Escrever um caso de teste -> Refatorar
c) Refatorar - > Escrever um código funcional - > Escrever um caso de teste
d) Escrever um caso de teste -> Escrever um código funcional -> Refatorar
e) Escrever um código funcional -> Escrever um caso de teste -> Refatorar

Comentários:



O ciclo de desenvolvimento orientado a testes segue o seguinte ciclo: (1) Identificar nova funcionalidade; (2) **Escrever um teste**; (3) Executar o teste; (4) **Implementar funcionalidade e refatorar**.

Gabarito: Letra D

---

11. (CESPE / BANRISUL – 2022) No processo de TDD, o código é desenvolvido em grandes blocos de requisitos do usuário. Cada iteração resulta em um novo teste, que faz parte um conjunto de testes de regressão executado no final do processo de integração.

Comentários:

Durante o TDD, o código é desenvolvido em incrementos **pequenos** e nenhum código é escrito enquanto não houver um teste para experimentá-lo. Cada iteração resulta em um ou mais novos testes, os quais são acrescentados a um conjunto de testes de regressão que são executados a cada mudança. Isso é feito para garantir que o novo código não tenha gerado efeitos colaterais que causem erros no código anterior.

Gabarito: Errado

---

12. **(CESPE / BANRISUL – 2022)** O TDD (Test-Driven Development) é uma metodologia que, ao longo do tempo, implica que o aplicativo em desenvolvimento tenha um conjunto abrangente de testes que ofereça confiança no que foi desenvolvido até então.

Comentários:

O TDD é um processo de desenvolvimento de software que incentiva os desenvolvedores a escrever testes antes de escrever código. A ideia por trás deste processo é que os desenvolvedores escrevam testes para cada pequena parte do código que eles escrevem, garantindo que o código funcione corretamente. Estes testes verificam se o código está funcionando como o esperado. O TDD também incentiva o uso de design incremental, onde pequenas partes do código são adicionadas gradualmente, permitindo que os problemas sejam detectados rapidamente e corrigidos. Ao longo do tempo, o TDD cria um conjunto abrangente de testes que fornecem confiança aos desenvolvedores de que o código é robusto e está funcionando como o esperado.

Gabarito: Correto

---

13. **(CESPE / BANRISUL – 2022)** O TDD (Test-Driven Development), como atividade da XP, é uma forma disciplinada de organizar o código, alterando-o de modo a aprimorar sua estrutura interna, sem que se altere o comportamento externo do software.

Comentários:

Essa não é a definição de TDD e, sim, Refactoring. O TDD é uma atividade do XP que envolve a escrita de testes antes mesmo de o código ser escrito, para que o desenvolvedor possa ter certeza de que o código que está escrevendo passará nos testes, e portanto que as funcionalidades desejadas estão sendo implementadas. Refatoração, por outro lado, é o processo de reescrever o código existente, para melhorar a sua qualidade, sem alterar a funcionalidade existente. Refatoração inclui a reestruturação do código para torná-lo mais limpo e legível, a remoção de código desnecessário, a simplificação da lógica usada e a correção de erros.

Gabarito: Errado

---

14. **(FGV / SEFAZ-MG – 2023)** Você entrou para um projeto novo, já em andamento, no qual a metodologia que a equipe do projeto segue é a de definir e escrever testes de software a partir das regras de negócio antes mesmo de implementar as funcionalidades propostas. Assinale a opção que indica o nome desse processo de desenvolvimento de software.

a) DDD
b) TDD
c) BDD
d) XP
e) Scrum

Comentários:

O TDD é um processo de desenvolvimento de software que incentiva a escrita de testes de software antes da implementação de qualquer código. O processo começa com a escrita de um teste para uma regra de negócio específica. Então, o código é escrito e, finalmente, os testes são executados para garantir que a regra de negócio esteja funcionando corretamente. O TDD é amplamente utilizado porque ajuda a garantir que o código seja testado de forma adequada, permitindo que qualquer problema seja detectado e corrigido antes que o código seja liberado para produção.

Gabarito: Letra B

---

15. **(FGV / Senado Federal – Análise de Sistemas – 2022)** Durante o processo de construção de software, a metodologia de Desenvolvimento Orientado a Testes é muito aplicada.

A ordem utilizada na prática do TDD é:

a) escrever a funcionalidade após escrever os testes unitários e, por fim, refatorar o código implementado.

b) escrever os testes unitários após escrever a funcionalidade e, por fim, refatorar o código implementado.

c) escrever a funcionalidade após refatorar o código implementado e por fim escrever os testes unitários.

d) escrever os testes unitários após escrever a funcionalidade e, por fim, escrever os testes de integração.

e) escrever os testes de integração após escrever a funcionalidade e, por fim, refatorar o código.

Comentários:

Essa é uma questão muito clássica! Basta saber a ordem correta: primeiro, escrevemos os testes unitários; depois, escrevemos a funcionalidade; por fim, refatoramos o código. Em outras palavras, a ordem correta é escrever a funcionalidade após escrever os testes unitários e, por fim, refatora o código implementado.

Gabarito: Letra A

---

16. (CESPE / SERPRO – 2021) Em TDD, os testes de um sistema devem ocorrer antes da implementação e ser oportunos, isolados e autoverificáveis.

Comentários:

Os testes devem seguir o modelo FIRST: F (Fast): devem ser rápidos, pois testam apenas uma unidade; I (Isolated): são isolados, testando individualmente as unidades e não sua integração; R (Repeateble): repetição nos testes, com resultados de comportamento constante; S (Self-verifying): autoverificação deve verificar se passou ou se deu como falha o teste; T (Timely): deve ser oportuno, sendo um teste por unidade.

Gabarito: Correto

---

17. (CESPE / INMETRO – 2009) A rotina diária dos desenvolvedores, ao empregar processos baseados no TDD (Test-Driven Development), é concentrada na elaboração de testes de homologação.

Comentários:

A rotina dos desenvolvedores é concentrada, na verdade, na elaboração de testes unitários e, não, de homologação. Lembrem-se: constrói o teste, constrói a funcionalidade e refatora a funcionalidade.

Gabarito: Errado

---

18. (CESPE / INPI – 2013) Usando-se o TDD, as funcionalidades devem estar completas e da forma como serão apresentadas aos seus usuários para que possam ser testadas e consideradas corretas.

Comentários:

ETAPA	DESCRIÇÃO
VERMELHO (RED)	Escreva um teste que falha: casos de teste são escritos sem que exista o recurso a ser testado, levando o teste a falhar;
VERDE (GREEN)	Escreva código para passar no teste: o recurso é implementado com o mínimo de código necessário para que passe no teste.

REFATORAR (REFACTOR)	Elimine redundâncias: o código que implementa o recurso é refinado e melhorado, sem que seja adicionada novas funcionalidades.

Pelo contrário, primeiro são feitos os testes e depois desenvolvem-se as funcionalidades.

Gabarito: Errado

---

19. (CESPE / ANCINE – 2013) No desenvolvimento de software conforme as diretivas do TDD (Test-Driven Development), deve-se elaborar primeiramente os testes e, em seguida, escrever o código necessário para passar pelos testes.

Comentários:

ETAPA	DESCRIÇÃO
VERMELHO (RED)	Escreva um teste que falha: casos de teste são escritos sem que exista o recurso a ser testado, levando o teste a falhar;
VERDE (GREEN)	Escreva código para passar no teste: o recurso é implementado com o mínimo de código necessário para que passe no teste.
REFATORAR (REFACTOR)	Elimine redundâncias: o código que implementa o recurso é refinado e melhorado, sem que seja adicionada novas funcionalidades.

Perfeito! Primeiro, criam-se os testes, depois cria-se o código.

▪

Gabarito: Correto

---

20. (CESPE / INMETRO – 2009) Considerando uma organização na qual a abordagem de Test Driven Development (TDD) esteja implementada, assinale a opção correta.

a) Nessa organização, ocorre a execução de iterações com ciclo longo, isto é, com duração de alguns meses.

b) No início de cada iteração, a primeira atividade realizada pela equipe de desenvolvimento é produzir o código que será validado através de testes.

c) O refactoring é uma das primeiras atividades realizada no início de cada iteração.

d) Entre as atividades finais de cada iteração, o desenvolvedor escreve casos de teste automatizados, cuja execução verifica se houve a melhoria desejada ou se uma nova funcionalidade foi implementada.

e) Há coerência e inter-relação com os princípios promovidos pela prática da extreme programming (XP).

Comentários:

(a) Errado, são ciclos curtos e, não, longos; (b) Errado, são produzidos primeiramente os testes e, depois, os códigos; (c) Errado, a refatoração é a última atividade realizada em uma iteração; (d) Errado, escrever casos de testes é uma das atividades iniciais; (e) Correto, trata-se inclusive de uma das práticas recomendadas pelo XP.

Gabarito: Letra E

---

21. (CESPE / MPOG – 2013) Ao realizar o TDD (test-driven development), o programador é conduzido a pensar em decisões de design antes de pensar em código de implementação, o que cria um maior acoplamento, uma vez que seu objetivo é pensar na lógica e nas responsabilidades de cada classe.

Comentários:

Em Engenharia de Software, há dois conceitos importantíssimos: coesão e acoplamento. Quando eu estudava, eu decorava uma frase pequena para entender: "*Coesão é a divisão de responsabilidades e Acoplamento é a dependência entre componentes*". Há outra que dizia assim: "*Uma boa arquitetura de software deve ter componentes de projeto com baixo acoplamento e alta coesão*".

O Acoplamento trata do nível de dependência entre módulos ou componentes de um software. *Por que é bom ter baixo acoplamento?* Porque se os módulos pouco dependem um do outro, modificações de um não afetam os outros, além de não prejudicar o reúso. Se esse princípio não for observado durante a construção da arquitetura de um sistema de software, pode haver problemas sérios de manutenção futura!

Voltando à questão: se o programador pensa em decisões de design antes de pensar em código de implementação, isso diminui o acoplamento - os componentes ficam menos dependentes, tornando a arquitetura bem mais flexível.

Gabarito: Errado

---

22. (CESPE / MPU – 2013) Na metodologia TDD, ou desenvolvimento orientado a testes, cada nova funcionalidade inicia com a criação de um teste, cujo planejamento permite a identificação dos itens e funcionalidades que deverão ser testados, quem são os responsáveis e quais os riscos envolvidos.

Comentários:

Perfeito! Essa metodologia permite um aprendizado maior sobre o problema a ser resolvido, permitindo (não obrigatoriamente) a identificação de itens, funcionalidades, responsáveis e riscos.

Gabarito: Correto

---

23. (CESPE / STF – 2013) No TDD, o primeiro passo do desenvolvedor é criar o teste, denominado teste falho, que retornará um erro, para, posteriormente, desenvolver o código e aprimorar a codificação do sistema.

Comentários:

ETAPA	DESCRIÇÃO
VERMELHO (RED)	Escreva um teste que falha: casos de teste são escritos sem que exista o recurso a ser testado, levando o teste a falhar;
VERDE (GREEN)	Escreva código para passar no teste: o recurso é implementado com o mínimo de código necessário para que passe no teste.
REFATORAR (REFACTOR)	Elimine redundâncias: o código que implementa o recurso é refinado e melhorado, sem que seja adicionada novas funcionalidades.

Perfeito! Cria-se o teste falho, desenvolve-se um código e só então ocorre a refatoração.

Gabarito: Correto

---

24. (CESPE / TRT17 – 2013) TDD consiste em uma técnica de desenvolvimento de software com abordagem embasada em perspectiva evolutiva de seu desenvolvimento. Essa abordagem envolve a produção de versões iniciais de um sistema a partir das quais é possível realizar verificações de suas qualidades antes que ele seja construído.

Comentários:

A questão diz que versões iniciais são produzidas e, a partir dessas versões, é possível realizar verificações de suas qualidades antes que ele seja construído. Na verdade, testes são criados inicialmente para verificar sua qualidade e, a partir daí, versões são produzidas. Logo, a questão inverteu os conceitos!

Gabarito: Correto

---

25. (CESPE / AL-RN – 2013) Um típico ciclo de vida de um projeto em TDD consiste em:

I. Executar os testes novamente e garantir que estes continuem tendo sucesso.
II. Executar os testes para ver se todos estes testes obtiveram êxito.
III. Escrever a aplicação a ser testada.
IV. Refatorar (refactoring).
V. Executar todos os possíveis testes e ver a aplicação falhar.
VI. Criar o teste.

A ordem correta e cronológica que deve ser seguida para o ciclo de vida do TDD está expressa em:

a) IV – III – II – V – I – VI.
b) V – VI – II – I – III – IV.
c) VI – V – III – II – IV – I.
d) III – IV – V – VI – I – II.
e) III – IV – VI – V – I – II.

Comentários:

A ordem correta é: (VI) Criar o teste; (V) Executar todos os possíveis testes e ver a aplicação falhar; (III) Escrever a aplicação a ser testada; (II) Executar os testes para ver se todos estes testes obtiveram êxito; (IV) Refatorar (refactoring); (I) Executar os testes novamente e garantir que estes continuem tendo sucesso.

Gabarito: Letra C

---

26. (FGV / ALMT – 2013) Com relação ao desenvolvimento orientado (dirigido) a testes (do Inglês Test Driven Development – TDD), analise as afirmativas a seguir.

I. TDD é uma técnica de desenvolvimento de software iterativa e incremental.

II. TDD implica escrever o código de teste antes do código de produção, um teste de cada vez, tendo certeza de que o teste falha antes de escrever o código que irá fazê-lo passar.

III. TDD é uma técnica específica do processo XP (Extreme Programming), portanto, só pode ser utilizada em modelos de processos ágeis de desenvolvimento de software.

Assinale:

a) Se somente as afirmativas I e II estiverem corretas.
b) Se somente as afirmativas I e III estiverem corretas.
c) Se somente as afirmativas II e III estiverem corretas.
d) Se somente a afirmativa III estiver correta.
e) Se somente a afirmativa I estiver correta.

Comentários:

(I) Correto, é iterativo e incremental (lembrando que, em sua imensa maioria, metodologias ágeis são iterativas/incrementais); (II) Correto, escrevem-se os testes antes, fá-lo falhar e só depois escreve-se o código da aplicação; (III) Errado, ele é uma abordagem independente que pode ser utilizado com metodologias ágeis ou tradicionais.

Gabarito: Letra A

---

27. (FCC / TRT-MG – 2015) Um analista de TI está participando do desenvolvimento de um software orientado a objetos utilizando a plataforma Java. Na abordagem de desenvolvimento adotada, o código é desenvolvido de forma incremental, em conjunto com o teste para esse incremento, de forma que só se passa para o próximo incremento quando o atual passar no teste. Como o código é desenvolvido em incrementos muito pequenos e são executados testes a cada vez que uma funcionalidade é adicionada ou que o programa é refatorado, foi necessário definir um ambiente de testes automatizados utilizando um framework popular que suporta o teste de programas Java.

A abordagem de desenvolvimento adotada e o framework de suporte à criação de testes automatizados são, respectivamente,

a) Behavior-Driven Development e JTest.
b) Extreme Programming e Selenium.
c) Test-Driven Development e Jenkins.
d) Data-Driven Development and Test e JUnit.
e) Test-Driven Development e JUnit.

Comentários:

A abordagem claramente é o TDD e o framework evidentemente é o JUnit (ferramenta de suporte à criação de testes unitários automatizados). Lembrando que: DDD é um modelo de programação em que os próprios dados controlam o fluxo do programa e não a lógica do programa; XP é uma metodologia ágil de gerenciamento de projetos que suporta lançamentos frequentes em curtos ciclos de desenvolvimento para melhorar a qualidade do software e permitir que os desenvolvedores respondam às mudanças nos requisitos dos clientes; BDD é um método de desenvolvimento ágil que encoraja a colaboração entre desenvolvedores; Selenium é uma ferramenta usada para testes funcionais em aplicações web; e JTest é um ramework de análise estática que usa a linguagem Java.

Gabarito: Letra E

---

28. (CESPE / TRE-PE – 2017) O desenvolvimento orientado a testes (TDD):

a) é um conjunto de técnicas que se associam ao XP (extreme programming) para o desenvolvimento incremental do código que se inicia com os testes.

b) agrega um conjunto de testes de integração para avaliar a interconexão dos componentes do software com as aplicações a ele relacionadas.

c) avalia o desempenho do desenvolvimento de sistemas verificando se o volume de acessos/transações está acima da média esperada.

d) averigua se o sistema atende aos requisitos de desempenho verificando se o volume de acessos/transações mantém-se dentro do esperado.

e) testa o sistema para verificar se ele foi desenvolvido conforme os padrões e a metodologia estabelecidos nos requisitos do projeto.

Comentários:

O único item que faz algum sentido em relação ao TDD é o primeiro, isto é, conjunto de técnicas intimamente ligadas ao XP para o desenvolvimento de software que se inicia com os testes.

Gabarito: Correto

---

29. (CESPE / STM – 2018) O TDD (test driven development) parte de um caso de teste que caracteriza uma melhoria desejada ou nova funcionalidade a ser desenvolvida, de modo a confirmar o comportamento correto e possibilitar a evolução ou refatoração do código.

Comentários:

TDD é um método para construir software que enfatiza a criação de testes antes da criação do código-fonte. Logo, faz-se o teste - se não passou, refatora!

Gabarito: Correto

---

30. (CESPE / TRE/PI – 2016) O TDD (test driven development):

a) apresenta como vantagem a leitura das regras de negócio a partir dos testes, e, como desvantagem, a necessidade de mais linhas de códigos que a abordagem tradicional, o que gera um código adicional.

b) impede que seja aplicada a prática de programação em pares, que é substituída pela interação entre analista de teste, testador e programador.

c) é um conjunto de técnicas associadas ao eXtremme Programing e a métodos ágeis, sendo, contudo, incompatível com o Refactoring, haja vista o teste ser escrito antes da codificação.

d) refere-se a uma técnica de programação cujo principal objetivo é escrever um código funcional limpo, a partir de um teste que tenha falhado.

e) refere-se a uma metodologia de testes em que se devem testar condições, loops e operações; no entanto, por questão de simplicidade, não devem ser testados polimorfismos.

Comentários:

(a) Errado, esse item não faz nenhum sentido em relação ao TDD; (b) Errado, não impede a programação em pares; (c) Errado, ela é totalmente compatível e dependente da refatoração; (d) Correto, ela pode ser vista como uma técnica de programação cujo objetivo é escrever um código funcional limpo a partir de um teste falho; (e) Errado, não se trata de uma metodologia de testes.

Gabarito: Letra D

---

31. (UFRRJ / UFRRJ – 2015) Os testes de unidade têm papel central na metodologia de implementação dirigida por testes, popularizada pelo processo XP e adotada em outros métodos. Esses testes são criados primeiro, exercitando o contrato de cada operação implementada pelos métodos. Em seguida, o código dos métodos é escrito para cumprir os contratos e, portanto, passar nos testes de unidade. Esse cenário corresponde à abordagem

a) TDD.
b) MDD.
c) DDC.
d) MDE.
e) FDD.

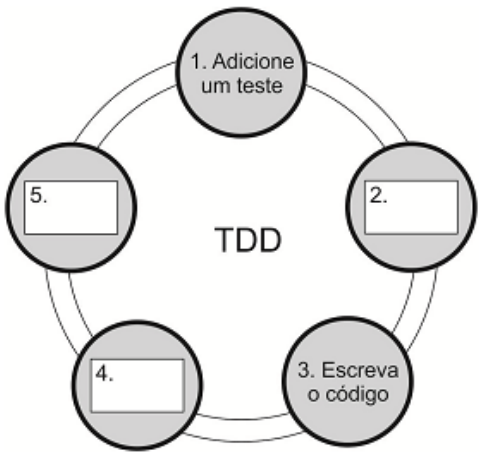
Comentários:

Testes de unidade têm papel central? Metodologia dirigida a testes? Popularizada pelo XP? Testes são criados primeiro? Tudo isso nos remente a... TDD!

Gabarito: Letra A

---

32. (FCC / TRE-PR – 2017) Considere o ciclo do Test-Driven Development – TDD.



A Caixa:

a) 2. corresponde a "Execute os testes automatizados".
b) 4. corresponde a "Refatore o código".
c) 5. corresponde a "Execute os testes novamente e observe os resultados".
d) 4. corresponde a "Execute os testes automatizados".
e) 5. corresponde a "Faça todos os testes passarem".

Comentários:

(a) Errado, corresponde a "Execute o teste e observe o resultado"; (b) Errado, corresponde a "Execute os testes automatizados"; (c) Errado, corresponde a "Refatore os códigos"; (d) Correto, corresponde realmente a "Execute os testes automatizados"; (e) Errado, corresponde a "Refatore os códigos".

Gabarito: Letra D

---

33. (IESES / TRE-MA – 2015) A respeito da técnica de testes TDD é correto afirmar que:

a) Testa o software com base no comportamento esperado.
b) É uma prática para desenvolvimento de testes unitário que pode utilizar o processo Red-Green-Refactor.
c) Utiliza-se da estrutura Dado, Quando e Então para montar os testes.
d) Prega que os testes devem ser realizados sempre após a implementação ser concluída.

Comentários:

(a) Errado, não vejo nada de errado nesse item – ele pode testar o software com base no comportamento esperado! No entanto, a banca considerou o item errado; (b) Correto, ele realmente utiliza o processo RED/GREEN/REFACTOR; (c) Errado, esse item não faz qualquer sentido; (d) Errado, testes são realizados antes de a implementação ser concluída.

Gabarito: Letra B

---

34. (CESPE / TER-RS – 2015) Projeto para o desenvolvimento de software que utilize TDD deve:

a) realizar sprints a cada quinzena.
b) desenvolver pequenos releases.
c) apresentar grande quantidade de testes unitários de código-fonte previamente desenvolvidos.
d) apresentar linguagem de programação estruturada.
e) recomendar a preparação dos testes para que, posteriormente, seja desenvolvido o código.

Comentários:

(a) Errado, não há nenhuma relação entre TDD e Sprints; (b) Errado, são desenvolvidas pequenas unidades e, não, releases; (c) Errado, os testes unitários são escritos iterativamente a medida que o desenvolvimento ocorre; (d) Errado, não há nenhuma relação ou dependência com linguagens de programação; (e) Correto, recomenda-se preparar testes antes do desenvolvimento do código.

Gabarito: Letra E

---

35. (FCC / TRE-AP – 2015) O TDD – Test Driven Development (Desenvolvimento orientado a teste):

a) é parte das metodologias ágeis UP – Unified Process e XP – Extreme Programming, tendo sido criado para ser usado em metodologias que respeitam os 4 princípios do Manifesto Ágil.

b) transforma o desenvolvimento, pois deve-se primeiro implementar o sistema antes de escrever os testes. Os testes são utilizados para facilitar no entendimento do projeto e para clarear o que se deseja em relação ao código.

c) baseia-se em um ciclo simples: escreve-se um código -> cria-se um teste para passar no código -> refatora-se.

d) propõe a criação de testes que validem o código como um todo para reduzir o tempo de desenvolvimento.

e) beneficia-se de testes que seguem o modelo FIRST: F (Fast) I (Isolated) R (Repeatable) S (Self-verifying) T (Timely).

Comentários:

ETAPA	DESCRIÇÃO

VERMELHO (RED)	Escreva um teste que falha: casos de teste são escritos sem que exista o recurso a ser testado, levando o teste a falhar;
VERDE (GREEN)	Escreva código para passar no teste: o recurso é implementado com o mínimo de código necessário para que passe no teste.
REFATORAR (REFACTOR)	Elimine redundâncias: o código que implementa o recurso é refinado e melhorado, sem que seja adicionada novas funcionalidades.

(a) Errado, Unified Process (UP) não é uma metodologia ágil; (b) Errado, devem ser implementados os testes antes do sistema; (c) Errado, cria-se um teste falho > escreve-se o código que passa > refatora-se; (d) Errado, criam-se testes que validam unidades e, não, o código como um todo; (e) Correto, vamos falar um pouquinho sobre isso agora:

Existe uma representação chamada Modelo FIRST: **F (Fast):** devem ser rápidos, pois testam apenas uma unidade; **I (Isolated):** testes unitários são isolados, testando individualmente as unidades e não sua integração; **R (Repeateble):** repetição nos testes, com resultados de comportamento constante; **S (Self-verifying):** a auto verificação deve verificar se passou ou se deu como falha o teste; **T (Timely):** o teste deve ser oportuno, sendo um teste por unidade.

Gabarito: Letra E

---

36. (CESPE / TRE-TO – 2017) O TDD (Test-Driven Development), que vem sendo adotado para testar os projetos de software,

a) utiliza os testes de caixa preta antes da entrega do software.
b) agiliza os testes por amostragem sem compatibilidade retroativa.
c) cobre amplamente os testes unitários.
d) escreve o teste antes da codificação do software.
e) realiza refactoring antes de escrever a aplicação a ser testada.

Comentários:

Eu já começo discordando do enunciado – ele vem sendo utilizado para desenvolver software utilizando testes, mas não para testar projetos de software. Ignorando a redação da questão, vamos aos comentários: (a) Errado, ele tipicamente utiliza testes de unidade antes da entrega do software; (b) Errado, esse item não faz qualquer sentido; (c) Errado, ele não tem o intuito principal de cobrir amplamente testes unitários; (d) Correto, o teste é escrito antes da codificação do software; (e) Errado, não faz sentido fazer *refactoring* antes de escrever a aplicação.

Gabarito: Letra D

---

37. (FGV / IBGE – 2016) O Desenvolvimento Orientado a Testes (TDD) é um método de desenvolvimento criado e disseminado por Kent Beck em seu livro "Test-driven development". O método define regras, boas práticas e um ciclo de tarefas com 3 etapas: a etapa vermelha, a etapa verde e a etapa de refatoração, ilustrado na imagem abaixo:

Com relação às regras e boas práticas de TDD e ao seu ciclo, é correto afirmar que:

a) pode-se escrever testes que não compilam na etapa vermelha;
b) na etapa verde deve-se escrever código que testa uma funcionalidade a fundo de forma criteriosa e detalhada;
c) código novo só é escrito se um teste automatizado passar;
d) a duplicação é tolerada na etapa de refatoração;
e) é uma boa prática de TDD iniciar o desenvolvimento do código de uma funcionalidade e, logo em seguida, testá-la.

Comentários:

ETAPA	DESCRIÇÃO
VERMELHO (RED)	Escreva um teste que falha: casos de teste são escritos sem que exista o recurso a ser testado, levando o teste a falhar;
VERDE (GREEN)	Escreva código para passar no teste: o recurso é implementado com o mínimo de código necessário para que passe no teste.
REFATORAR (REFACTOR)	Elimine redundâncias: o código que implementa o recurso é refinado e melhorado, sem que seja adicionada novas funcionalidades.

(a) Correto. Nessa etapa, o objetivo do desenvolvedor é escrever um pequeno teste que não funcione e que talvez nem mesmo compile inicialmente; (b) Errado. Nessa etapa, escreve-se o código que apenas passa no teste, sem necessidade de aprofundamento em detalhes de implementação; (c) Errado. O código novo é escrito para o teste automatizado passar; (d) Errado. Não é tolerada a duplicação, uma vez que o objetivo é eliminar redundâncias e melhorar o código; (e) Errado. Vimos centenas de vezes que o primeiro passo é criar o teste e depois escrever.

Gabarito: Letra A

---

38. (IADES / EBSERH – 2013) Assinale a alternativa que não corresponde a uma das fases do processo de desenvolvimento, dirigido a testes (TDD).

a) Executar o teste, com os outros testes implementados, que rodarão e fornecerão o resultado de que o software está sem problemas.

b) Escrever o teste para a funcionalidade e implementação.
c) Realizar a identificação do incremento de funcionalidade.
d) Implementar a funcionalidade e executar novamente o teste.
e) Implementar a próxima parte da funcionalidade, após todos os testes terem sido executados, com sucesso

Comentários:

(a) Errado. Na terceira fase, executa-se realmente o teste junto com todos os outros testes implementados. No entanto, inicialmente você não terá implementado a funcionalidade, logo o novo teste falhará; (b) Correto, essa é a segunda fase; (c) Correto, essa é a primeira fase; (d) Correto, essa é a quarta fase; (e) Correto, essa é a quinta fase.

Gabarito: Letra A

---

39. (PR-4 / UFRJ – 2018) O ciclo do TDD - Test Driven Development, ou, em português, Desenvolvimento Guiado por Testes consiste em:

a) implementar teste unitário falho, tornar o teste bem-sucedido e refatorar.
b) implementar a funcionalidade, executar teste unitário e refatorar.
c) implementar teste unitário falho, refatorar e tornar o teste bem-sucedido.
d) implementar a funcionalidade, refatorar e tornar o teste bem-sucedido.
e) refatorar, executar teste unitário e implementar a funcionalidade.

Comentários:

(a) Correto, as atividades são exatamente essas e nessa ordem; (b) Errado, o teste vem antes da implementação da funcionalidade; (c) Errado, primeiro você torna o teste bem sucedido e depois refatora; (d) Errado, não faz sentido refatorar antes do teste; (e) Errado, não faz sentido refatorar antes do teste.

Gabarito: Letra A

---

40. (CESPE / STJ – 2015) Um dos passos executados no ciclo de atividades do processo TDD é a criação de novos testes para as falhas encontradas no código original, sem alteração deste.

Comentários:

Nooooope... testes devem realmente encontrar falhas – o que se altera é o código para passar no teste e, não, o teste em si.

Gabarito: Errado

---

41. (CS-UFG / AL-GO – 2015) O desenvolvimento dirigido a testes (TDD, do Inglês Test-Driven Development) é uma abordagem de desenvolvimento de software na qual se intercalam testes e desenvolvimento de código. Uma das características da abordagem TDD é:

a) a sua utilidade no desenvolvimento de softwares novos.
b) o maior custo associado aos testes de regressão.
c) a redução da importância da automatização dos testes.
d) a sua adequação a processos de software sequenciais.

Comentários:

(a) Correto. Ela é mais ideal para o desenvolvimento de softwares novos; (b) Errado, ela reduz custos de testes de regressão; (c) Errado, ela aumenta a importância da automatização de testes; (d) Errado, ela é adequada a processos iterativos – lembrem-se do ciclo de testes, falhas e refatorações.

Gabarito: Letra A

---

42. (UECE-CEV / FUNCEME – 2018) Test-driven Development (TDD) é uma abordagem para o desenvolvimento de programas em que se intercalam testes e desenvolvimento de código
(Sommerville, I. Engenharia de Software, 9a edição, 2011).

A respeito do TDD, é correto afirmar que:

a) consiste em um processo iterativo que se inicia escrevendo um código de uma funcionalidade do sistema e, logo em seguida, testa-o para saber se a implementação foi correta.
b) apesar de útil, não diminui o custo de testes de regressão do sistema.
c) sua utilização elimina a necessidade de testes de validação do sistema, uma vez que ele já foi testado incrementalmente.
d) apesar de ter sido apresentado como parte dos métodos ágeis, também pode ser usado em outros processos de desenvolvimento de software.

Comentários:

(a) Errado, inicia-se escrevendo o teste; (b) Errado, diminui – sim – os custos de testes de regressão; (c) Errado, não elimina em nenhuma hipótese testes de validação/aceitação; (d) Correto, ele pode ser utilizado com métodos ágeis ou tradicionais.

Gabarito: Letra D

---

43. (FAUGRS / UFRGS – 2018) _____ é uma abordagem para o desenvolvimento de programas em que se intercalam testes e desenvolvimento de código. Essencialmente, desenvolve-se um código de forma incremental em conjunto com um teste para este incremento. Não se avança para o próximo incremento até que o código desenvolvido passe no teste. Essa abordagem foi introduzida como parte de métodos ágeis, mas pode ser também

usada em processos de desenvolvimento dirigido a planos. Assinale a alternativa que preenche corretamente a lacuna do texto acima.

a) Desenvolvimento Guiado por Testes (TDD)
b) Desenvolvimento em Espiral
c) Engenharia Dirigida a Modelos (MDD)
d) Rational Unified Process (RUP)
e) Teste de Sistema

Comentários:

TDD é uma **abordagem para o desenvolvimento** de programas em que se **intercalam testes e desenvolvimento de código**. Essencialmente, desenvolve-se um código de forma incremental em conjunto com um teste para este incremento. Não se avança para o próximo incremento até que o código desenvolvido passe no teste. Essa abordagem foi introduzida como parte de métodos ágeis, mas pode ser também usada em processos de desenvolvimento dirigido a planos.

Gabarito: Letra A

---

44. (VUNESP / TCE-SP – 2015) No Desenvolvimento Orientado a Testes (TDD), os casos de teste que definem o recurso a ser implementado devem ser elaborados:

a) assim que o código do teste estiver pronto.
b) antes de o código do recurso ser desenvolvido.
c) após o código do recurso ter sido completamente documentado.
d) simultaneamente com o desenvolvimento do código do recurso.
e) somente se o código do recurso apresentar erros.

Comentários:

(a) Errado, os casos de testes já devem estar prontos antes do código do teste ser feito; (b) Correto, devem ser elaborados antes de o código do recurso ser desenvolvido; (c) Errado, não existe o conceito de documentação formal, os próprios testes agem como uma forma de documentação que descreve o que o código deve fazer; (d) Errado, as etapas de desenvolvimento do código do recurso e a implementação dos casos de testes ocorrem em etapas distintas e não concomitantes; (e) Errado, casos de testes sempre são implementados porque eles representam a funcionalidade a ser desenvolvida.

Gabarito: Letra B

---

45. (IBFC / EMBASA – 2017) No Ciclo de Desenvolvimento do TDD (Test-Driven Development), utiliza-se a estratégia que aplica três palavras-chaves (em inglês), que é denominada:

a) Red, Green, Refactor
b) White, Gray, Black
c) White, Black, Refactor
d) Green, Yellow, Red

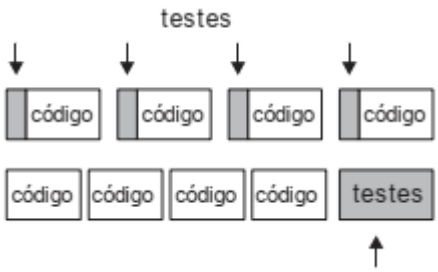
Comentários:

ETAPA	DESCRIÇÃO
VERMELHO (RED)	Escreva um teste que falha: casos de teste são escritos sem que exista o recurso a ser testado, levando o teste a falhar;
VERDE (GREEN)	Escreva código para passar no teste: o recurso é implementado com o mínimo de código necessário para que passe no teste.
REFATORAR (REFACTOR)	Elimine redundâncias: o código que implementa o recurso é refinado e melhorado, sem que seja adicionada novas funcionalidades.

Trata-se do RED > GREEN > REFACTOR.

Gabarito: Letra A

46.(FCC / CREMESP – 2016) Considere a figura abaixo que apresenta duas abordagens de teste.



A figura:

a) ilustra as duas fases do TDD, que correspondem a escrever pequenos testes e testá-los no final.
b) mostra o ciclo conhecido como Vermelho-Verde-Refatora.
c) apresenta a diferença entre testes automatizados e testes manuais no XP.
d) mostra que um desenvolvedor que pratica TDD tem mais feedbacks do que um que escreve testes ao final.
e) evidencia que TDD é impraticável, pois o desenvolvedor gasta muito tempo escrevendo código de testes.

Comentários:

Qual é a diferença entre a abordagem superior e inferior? Na parte superior, testes são aplicados individualmente em cada unidade de código. Na parte inferior, todos os códigos foram escritos e,

somente depois, foram testados. Logo, existem mais feedbacks na abordagem superior (TDD) do que na abordagem inferior.

Gabarito: Letra D

---

47. (CESPE / ANATEL – 2014) Em se tratando de desenvolvimento de softwares dirigidos a testes (TDD), a execução dos testes é realizada antes da implementação da funcionalidade.

Comentários:

Perfeito! A execução dos testes de realmente ocorrem antes da implementação da funcionalidade.

Gabarito: Correto

---

48. (CESPE / TC-DF – 2014) No TDD, o refatoramento do código deve ser realizado antes de se escrever a aplicação que deve ser testada.

Comentários:

Opaaaaa... como você vai refatorar um código que ainda não foi escrito? Não faz sentido!

Gabarito: Errado

---

49. (CESPE / STJ – 2015) No método de desenvolvimento TDD (Test Driven Development), o desenvolvedor escreve primeiro um caso de teste e, posteriormente, o código.

Comentários:

Perfeito! Esse é o conceito do *test-first*, isto é, o desenvolvedor escreve primeiro um caso de teste e, posteriormente, o código.

Gabarito: Correto

---

50. (CESPE / MPE-PI – 2018) O TDD possibilita o desenvolvimento de softwares fundamentado em testes. O ciclo de desenvolvimento do TDD segue os seguintes passos:

- escrever um teste que inicialmente não passa;
- adicionar uma nova funcionalidade do sistema;
- fazer o teste passar;
- realizar a integração contínua do código;
- escrever o próximo teste.

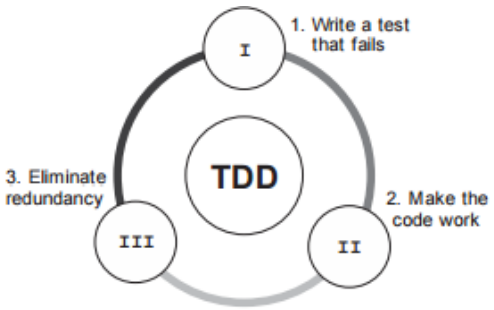
Comentários:

A ordem realmente está correta, no entanto há uma pegadinha: na quarta etapa, realiza-se a refatoração do código e, não, a integração contínua.

Gabarito: Errado

---

51. (FCC / Prefeitura de Teresina-PI – 2016) O Test Driven Development – TDD é uma das práticas sugeridas na eXtreme Programming – XP, onde o programador escreve o teste antes de escrever o código. O ciclo de desenvolvimento utilizando TDD é mostrado abaixo.



Considere:

I. Etapa inicial, onde se escreve um teste que falha, para alguma funcionalidade que ainda será Escrita.
II. Já com o teste criado, é o momento de executar o teste.
III. Eliminar códigos redundantes, remover acoplamentos, enfim, identificar pontos de melhoria no código.

As etapas I, II e III são, respectivamente,

a) Iniciação, Execução e Controle.
b) Red, Green e Refactor.
c) Iniciação, Atuação e Otimização.
d) Plan, Do e Check.
e) Planejamento, Execução e Melhoria.

Comentários:

ETAPA	DESCRIÇÃO
VERMELHO (RED)	Escreva um teste que falha: casos de teste são escritos sem que exista o recurso a ser testado, levando o teste a falhar;
VERDE (GREEN)	Escreva código para passar no teste: o recurso é implementado com o mínimo de código necessário para que passe no teste.
REFATORAR (REFACTOR)	Elimine redundâncias: o código que implementa o recurso é refinado e melhorado, sem que seja adicionada novas funcionalidades.

(RED) Etapa inicial, onde se escreve um teste que falha, para alguma funcionalidade que ainda será Escrita; (GREEN) Já com o teste criado, é o momento de executar o teste; (REFACTOR) Eliminar códigos redundantes, remover acoplamentos, enfim, identificar pontos de melhoria no código.

Gabarito: Letra B

---

52. (FAUGRS / BANRISUL – 2018) Considere as ações abaixo, executadas em desenvolvimento orientado a testes, Test-Driven Design (TDD).

I - Escrever código de teste.
II - Verificar se o teste falha.
III - Escrever código de produção.
IV - Executar teste até passar (reescrevendo o código de produção, se for necessário, até que o teste passe).
V - Refatorar código de produção e/ou de teste para melhorá-lo.

Considerando que se deseja incluir um novo caso de teste, assinale a alternativa que apresenta a sequência de ações que devem obrigatoriamente ocorrer para essa inclusão, segundo o TDD.

a) I, III e IV.
b) III, I e IV.
c) I, II, III e IV.
d) I, III, IV e V.
e) I, II, III, IV e V.

Comentários:

A ordem correta é: (I) Escrever código de teste; (II) Verificar se o teste falha; (III) Escrever código de produção; (IV) Executar teste até passar (reescrevendo o código de produção, se for necessário, até que o teste passe); (V) Errado, essa não é uma ação obrigatória – como apresenta o enunciado.

Gabarito: Letra C

---

53. (FGV / IBGE – 2017) Test Driven Development (TDD) é uma prática muito utilizada no processo de desenvolvimento de sistemas computacionais. Analise as afirmativas a seguir sobre o uso da prática de TDD:

I. Tornam os testes de regressão mais demorados porque o desenvolvedor precisará fazer testes manuais várias vezes por dia.

II. Garante que os requisitos do sistema sejam atendidos porque o desenvolvedor escreverá o código de testes sempre que acabar a implementação do código do sistema.

III. Ajuda o desenvolvedor a escrever código de qualidade porque ele gastará parte do seu tempo escrevendo código de testes.

Está correto o que se afirma em:

a) somente I;
b) somente II;
c) somente III;
d) somente II e III;
e) I, II e III.

Comentários:

(I) Errado. De acordo com Ian Sommerville, um dos benefícios mais importantes de desenvolvimento dirigido a testes é que ele reduz os custos dos testes de regressão, uma vez que testes automatizados – fundamentais para o desenvolvimento *test-first* – reduzem drasticamente os custos com testes de regressão; (II) Errado. O desenvolver escreverá o código de testes antes de acabar a implementação do código do sistema; (III) Correto. O TDD realmente ajuda o desenvolvedor a escrever código de qualidade porque ele gastará parte do seu tempo escrevendo código de testes.

Gabarito: Letra C

---

54. (CESPE / ANATEL – 2014) Na atividade de TDD (test-driven development), a escrita de teste primeiro define implicitamente tanto uma interface quanto uma especificação do comportamento para a funcionalidade que está sendo desenvolvida, estando, entretanto, a viabilidade do uso dessa abordagem limitada aos processos de desenvolvimento de software que seguem as práticas ágeis.

Comentários:

Opa... a interface deve ser definida explicitamente! Além disso, conforme afirma Ian Sommerville, ele também pode ser utilizado em processos de desenvolvimento dirigido a planos (tradicionais) e, não só, aos ágeis.

Gabarito: Errado

---

55. (CESPE / TRE-MT – 2015) Considere as seguintes etapas de um processo do tipo desenvolvimento orientado a testes (TDD).

I Implementar funcionalidade e refatorar.
II Identificar nova funcionalidade.
III Executar o teste.

IV Escrever o teste.
V Implementar a próxima parte da funcionalidade.

Assinale a opção que apresenta a sequência correta em que essas etapas devem ser realizadas.

a) I; IV; III; II; V
b) IV; III; II; I; V
c) I; IV; II; III; V
d) II; IV; III; I; V
e) IV; II; III; I; V

Comentários:

A ordem correta é: (II) Identificar nova funcionalidade; (IV) Escrever o teste; (III) Executar o teste; (I) Implementar funcionalidade e refatorar; (V) Implementar a próxima parte da funcionalidade.

Gabarito: Letra D

---

56. **(FCC / SEFAZ-SC – 2018)** O Test-Driven Development (TDD) é uma abordagem para o desenvolvimento de programas em que se intercalam testes e desenvolvimento de código. As etapas do processo fundamental de TDD são mostradas abaixo em ordem alfabética:

I. Escrever um teste para a funcionalidade identificada e implementá-lo como um teste automatizado.

II. Executar o teste, junto com os demais testes já implementados, sem implementar a nova funcionalidade no código.

III. Identificar e implementar uma outra funcionalidade, após todos os testes serem executados com sucesso.

IV. Identificar uma nova funcionalidade pequena para ser incrementada com poucas linhas em um código.

V. Implementar a nova funcionalidade no código e reexecutar o teste.

VI. Refatorar o código com melhorias incrementais até que o teste execute sem erros.

VII. Revisar a funcionalidade e o teste, caso o código execute sem falhar.

Considerando o item IV a primeira etapa e o item III a última etapa, a sequência intermediária correta das etapas do processo é:

a) I – II – VII – V e VI.
b) I – V – II – VII e VI.
c) I – VI – V – VII e II.
d) V – I – II – VII e VI.
e) V – I – VI – VII e II.

Comentários:

São somente cinco etapas: I. Escrever um teste para a funcionalidade identificada e implementá-lo como um teste automatizado; II. Executar o teste, junto com os demais testes já implementados, sem implementar a nova funcionalidade no código; VII. Revisar a funcionalidade e o teste, caso o código execute sem falhar; V. Implementar a nova funcionalidade no código e reexecutar o teste; VI. Refatorar o código com melhorias incrementais até que o teste execute sem erros.

Gabarito: Letra A

LISTA DE QUESTÕES – DIVERSAS BANCAS

1. **(CESGRANRIO / IPEA – 2024)** Uma gerente de testes de software propôs a seu time de desenvolvimento que começasse a aplicar a abordagem Test Driven Development (TDD). É uma das características principais dessa abordagem iniciar o desenvolvimento de testes:

   a) antes de implementar alguma funcionalidade em si.
   b) durante o período de homologação.
   c) após as funcionalidades serem construídas.
   d) quando a primeira leva de funcionalidades planejadas forem codificadas em algum sprint.
   e) pelos testes de interface automatizado, seguidos pelos testes unitários.

2. **(CESPE / INPI – 2024)** O desenvolvimento dirigido por testes (TDD) é modelado em três estados: vermelho, verde e refatorar. Um exemplo da ação de refatoração é a simulação do comportamento dos componentes que interagem com a unidade de teste que está falhando.

3. **(CESPE / CNPq – 2024)** O TDD é uma tendência que enfatiza o projeto de casos de teste antes da criação do código fonte e se caracteriza como parte do modelo ágil de desenvolvimento de software.

4. **(CESPE / CNPq – 2024)** No TDD, o teste deve ser criado com o objetivo de fazer o segmento de código falhar, gerando-se um processo iterativo que permite a submissão de muitas subfunções simultaneamente, o que confere uma agilidade significativa ao processo.

5. **(CESPE / AGER-MT – 2023)** Assinale a opção que corresponde ao método de teste de software adotado, sob a perspectiva do desenvolvedor, a partir de casos de teste do código, escritos em linguagem técnica, para testar as funcionalidades antes da implementação da solução desenvolvida:

   a) unit testing
   b) TDD
   c) BDD
   d) ATDD
   e) teste de caixa preta

6. **(CESPE / TC-DF – 2023)** Na etapa de refactor do processo TDD, parte-se do pressuposto de que os testes tenham passado nas fases anteriores, o que permite que o código seja aprimorado sem a preocupação de duplicações de código.

7. **(CESPE / EMPREL – 2023)** Ao adotar uma prática ágil para a criação de um software, seu desenvolvedor optou pela implementação com qualidade de uma funcionalidade do sistema; para isso, escreveu um caso de teste automatizado, com base nos requisitos especificados, e realizou testes de unidade em uma linguagem similar à usada no desenvolvimento da

funcionalidade. Da situação hipotética precedente infere-se que a prática adotada pelo desenvolvedor está associada ao:

a) desenvolvimento orientado por comportamento (BDD).
b) gerenciamento de produtos com Scrum.
c) desenvolvimento guiado por testes (TDD).
d) desenvolvimento guiado por testes de aceitação (ATDD).
e) gerenciamento de produtos com Kanban.

8. **(FGV / SEFAZ-MG – 2023)** Você entrou para um projeto novo, já em andamento, no qual a metodologia que a equipe do projeto segue é a de definir e escrever testes de software a partir das regras de negócio antes mesmo de implementar as funcionalidades propostas. Assinale a opção que indica o nome desse processo de desenvolvimento de software:

a) DDD
b) TDD
c) BDD
d) XP
e) Scrum

9. **(FGV / BB – 2023)** O desenvolvimento orientado a testes (TDD) é um processo que se baseia na repetição em ciclos de desenvolvimento curtos. Ele é baseado no conceito test-first oriundo da programação extrema (XP) que incentiva o design simples com alto nível de confiança. O procedimento que conduz este ciclo é denominado:

a) refatoração vermelho-verde.
b) documentação executável.
c) testes da caixa-branca.
d) testes da caixa-preta.
e) mocking up.

10. **(FGV / Senado Federal – 2022)** Você foi contratado para liderar uma equipe de DevOps. Um dos objetivos da sua liderança é aumentar a velocidade das entregas e a qualidade de novos recursos das aplicações utilizando o desenvolvimento orientado a testes. Assinale a opção que apresenta a ordem que descreve o ciclo de desenvolvimento orientado a testes.

a) Refatorar - > Escrever um código funcional
b) Escrever um caso de teste -> Refatorar
c) Refatorar - > Escrever um código funcional - > Escrever um caso de teste
d) Escrever um caso de teste -> Escrever um código funcional -> Refatorar
e) Escrever um código funcional -> Escrever um caso de teste -> Refatorar

11. **(CESPE / BANRISUL – 2022)** No processo de TDD, o código é desenvolvido em grandes blocos de requisitos do usuário. Cada iteração resulta em um novo teste, que faz parte um conjunto de testes de regressão executado no final do processo de integração.

12. **(CESPE / BANRISUL – 2022)** O TDD (Test-Driven Development) é uma metodologia que, ao longo do tempo, implica que o aplicativo em desenvolvimento tenha um conjunto abrangente de testes que ofereça confiança no que foi desenvolvido até então.

13. **(CESPE / BANRISUL – 2022)** O TDD (Test-Driven Development), como atividade da XP, é uma forma disciplinada de organizar o código, alterando-o de modo a aprimorar sua estrutura interna, sem que se altere o comportamento externo do software.

14. **(FGV / SEFAZ-MG – 2023)** Você entrou para um projeto novo, já em andamento, no qual a metodologia que a equipe do projeto segue é a de definir e escrever testes de software a partir das regras de negócio antes mesmo de implementar as funcionalidades propostas. Assinale a opção que indica o nome desse processo de desenvolvimento de software.

a) DDD
b) TDD
c) BDD
d) XP
e) Scrum

15. **(FGV / Senado Federal – Análise de Sistemas – 2022)** Durante o processo de construção de software, a metodologia de Desenvolvimento Orientado a Testes é muito aplicada.

A ordem utilizada na prática do TDD é:

a) escrever a funcionalidade após escrever os testes unitários e, por fim, refatorar o código implementado.

b) escrever os testes unitários após escrever a funcionalidade e, por fim, refatorar o código implementado.

c) escrever a funcionalidade após refatorar o código implementado e por fim escrever os testes unitários.

d) escrever os testes unitários após escrever a funcionalidade e, por fim, escrever os testes de integração.

e) escrever os testes de integração após escrever a funcionalidade e, por fim, refatorar o código.

16. **(CESPE / SERPRO – 2021)** Em TDD, os testes de um sistema devem ocorrer antes da implementação e ser oportunos, isolados e autoverificáveis.

17. **(CESPE / INMETRO / 2009)** A rotina diária dos desenvolvedores, ao empregar processos baseados no TDD (Test-Driven Development), é concentrada na elaboração de testes de homologação.

18. **(CESPE / INPI / 2013)** Usando-se o TDD, as funcionalidades devem estar completas e da forma como serão apresentadas aos seus usuários para que possam ser testadas e consideradas corretas.

19. **(CESPE / ANCINE / 2013)** No desenvolvimento de software conforme as diretivas do TDD (Test-Driven Development), deve-se elaborar primeiramente os testes e, em seguida, escrever o código necessário para passar pelos testes.

20. **(CESPE / INMETRO / 2009)** Considerando uma organização na qual a abordagem de Test Driven Development (TDD) esteja implementada, assinale a opção correta.

a) Nessa organização, ocorre a execução de iterações com ciclo longo, isto é, com duração de alguns meses.

b) No início de cada iteração, a primeira atividade realizada pela equipe de desenvolvimento é produzir o código que será validado através de testes.

c) O refactoring é uma das primeiras atividades realizada no início de cada iteração.

d) Entre as atividades finais de cada iteração, o desenvolvedor escreve casos de teste automatizados, cuja execução verifica se houve a melhoria desejada ou se uma nova funcionalidade foi implementada.

e) Há coerência e inter-relação com os princípios promovidos pela prática da extreme programming (XP).

21. **(CESPE / MPOG / 2013)** Ao realizar o TDD (test-driven development), o programador é conduzido a pensar em decisões de design antes de pensar em código de implementação, o que cria um maior acoplamento, uma vez que seu objetivo é pensar na lógica e nas responsabilidades de cada classe.

22. **(CESPE / MPU – 2013)** Na metodologia TDD, ou desenvolvimento orientado a testes, cada nova funcionalidade inicia com a criação de um teste, cujo planejamento permite a identificação dos itens e funcionalidades que deverão ser testados, quem são os responsáveis e quais os riscos envolvidos.

23. **(CESPE / STF – 2013)** No TDD, o primeiro passo do desenvolvedor é criar o teste, denominado teste falho, que retornará um erro, para, posteriormente, desenvolver o código e aprimorar a codificação do sistema.

24. **(CESPE / TRT-17 – 2013)** TDD consiste em uma técnica de desenvolvimento de software com abordagem embasada em perspectiva evolutiva de seu desenvolvimento. Essa abordagem envolve a produção de versões iniciais de um sistema a partir das quais é possível realizar verificações de suas qualidades antes que ele seja construído.

25. **(CESPE / ALRN – 2013)** Um típico ciclo de vida de um projeto em TDD consiste em:

I. Executar os testes novamente e garantir que estes continuem tendo sucesso.
II. Executar os testes para ver se todos estes testes obtiveram êxito.
III. Escrever a aplicação a ser testada.
IV. Refatorar (refactoring).
V. Executar todos os possíveis testes e ver a aplicação falhar.
VI. Criar o teste.

A ordem correta e cronológica que deve ser seguida para o ciclo de vida do TDD está expressa em:

a) IV – III – II – V – I – VI.
b) V – VI – II – I – III – IV.
c) VI – V – III – II – IV – I.
d) III – IV – V – VI – I – II.
e) III – IV – VI – V – I – II.

26. **(FGV / ALMT – 2013)** Com relação ao desenvolvimento orientado (dirigido) a testes (do Inglês Test Driven Development – TDD), analise as afirmativas a seguir.

I. TDD é uma técnica de desenvolvimento de software iterativa e incremental.

II. TDD implica escrever o código de teste antes do código de produção, um teste de cada vez, tendo certeza de que o teste falha antes de escrever o código que irá fazê-lo passar.

III. TDD é uma técnica específica do processo XP (Extreme Programming), portanto, só pode ser utilizada em modelos de processos ágeis de desenvolvimento de software.

Assinale:

a) Se somente as afirmativas I e II estiverem corretas.
b) Se somente as afirmativas I e III estiverem corretas.
c) Se somente as afirmativas II e III estiverem corretas.
d) Se somente a afirmativa III estiver correta.
e) Se somente a afirmativa I estiver correta.

27. **(FCC / TRT-MG – 2015)** Um analista de TI está participando do desenvolvimento de um software orientado a objetos utilizando a plataforma Java. Na abordagem de desenvolvimento adotada, o código é desenvolvido de forma incremental, em conjunto com o teste para esse incremento, de forma que só se passa para o próximo incremento quando o atual passar no teste. Como o código é desenvolvido em incrementos muito pequenos e são executados testes a cada vez que uma funcionalidade é adicionada ou que o programa é refatorado, foi necessário definir um ambiente de testes automatizados utilizando um framework popular que suporta o teste de programas Java.

A abordagem de desenvolvimento adotada e o framework de suporte à criação de testes automatizados são, respectivamente,

a) Behavior-Driven Development e JTest.
b) Extreme Programming e Selenium.
c) Test-Driven Development e Jenkins.
d) Data-Driven Development and Test e JUnit.
e) Test-Driven Development e JUnit.

28. **(CESPE / TRE-PE – 2017)** O desenvolvimento orientado a testes (TDD):

a) é um conjunto de técnicas que se associam ao XP (extreme programming) para o desenvolvimento incremental do código que se inicia com os testes.

b) agrega um conjunto de testes de integração para avaliar a interconexão dos componentes do software com as aplicações a ele relacionadas.

c) avalia o desempenho do desenvolvimento de sistemas verificando se o volume de acessos/transações está acima da média esperada.

d) averigua se o sistema atende aos requisitos de desempenho verificando se o volume de acessos/transações mantém-se dentro do esperado.

e) testa o sistema para verificar se ele foi desenvolvido conforme os padrões e a metodologia estabelecidos nos requisitos do projeto.

29. **(CESPE / STM – 2018)** O TDD (test driven development) parte de um caso de teste que caracteriza uma melhoria desejada ou nova funcionalidade a ser desenvolvida, de modo a confirmar o comportamento correto e possibilitar a evolução ou refatoração do código.

30. **(CESPE / TRE/PI – 2016)** O TDD (test driven development):

a) apresenta como vantagem a leitura das regras de negócio a partir dos testes, e, como desvantagem, a necessidade de mais linhas de códigos que a abordagem tradicional, o que gera um código adicional.

b) impede que seja aplicada a prática de programação em pares, que é substituída pela interação entre analista de teste, testador e programador.

c) é um conjunto de técnicas associadas ao eXtremme Programing e a métodos ágeis, sendo, contudo, incompatível com o Refactoring, haja vista o teste ser escrito antes da codificação.

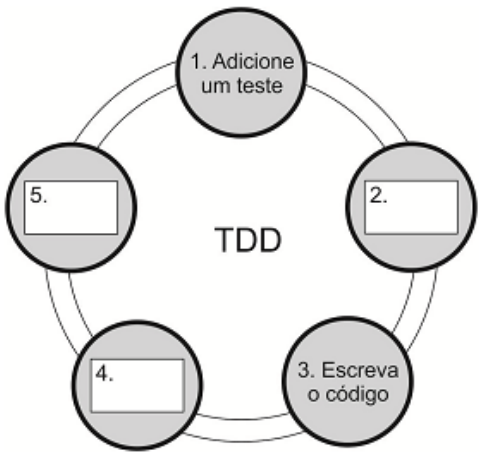
d) refere-se a uma técnica de programação cujo principal objetivo é escrever um código funcional limpo, a partir de um teste que tenha falhado.

e) refere-se a uma metodologia de testes em que se devem testar condições, loops e operações; no entanto, por questão de simplicidade, não devem ser testados polimorfismos.

31. **(UFRRJ / UFRRJ – 2015)** Os testes de unidade têm papel central na metodologia de implementação dirigida por testes, popularizada pelo processo XP e adotada em outros métodos. Esses testes são criados primeiro, exercitando o contrato de cada operação implementada pelos métodos. Em seguida, o código dos métodos é escrito para cumprir os contratos e, portanto, passar nos testes de unidade. Esse cenário corresponde à abordagem

a) TDD.
b) MDD.
c) DDC.
d) MDE.
e) FDD.

32. **(FCC / TRE-PR – 2017)** Considere o ciclo do Test-Driven Development – TDD.



A Caixa:

a) 2. corresponde a "Execute os testes automatizados".
b) 4. corresponde a "Refatore o código".
c) 5. corresponde a "Execute os testes novamente e observe os resultados".

d) 4. corresponde a "Execute os testes automatizados".
e) 5. corresponde a "Faça todos os testes passarem".

33. **(IESES / TRE/MA – 2015)** A respeito da técnica de testes TDD é correto afirmar que:

a) Testa o software com base no comportamento esperado.
b) É uma prática para desenvolvimento de testes unitário que pode utilizar o processo Red-Green-Refactor.
c) Utiliza-se da estrutura Dado, Quando e Então para montar os testes.
d) Prega que os testes devem ser realizados sempre após a implementação ser concluída.

34. **(CESPE / TRE/RS – 2015)** Projeto para o desenvolvimento de software que utilize TDD deve:

a) realizar sprints a cada quinzena.
b) desenvolver pequenos releases.
c) apresentar grande quantidade de testes unitários de código-fonte previamente desenvolvidos.
d) apresentar linguagem de programação estruturada.
e) recomendar a preparação dos testes para que, posteriormente, seja desenvolvido o código.

35. **(FCC / TRE/AP – 2015)** O TDD – Test Driven Development (Desenvolvimento orientado a teste):

a) é parte das metodologias ágeis UP – Unified Process e XP – Extreme Programming, tendo sido criado para ser usado em metodologias que respeitam os 4 princípios do Manifesto Ágil.

b) transforma o desenvolvimento, pois deve-se primeiro implementar o sistema antes de escrever os testes. Os testes são utilizados para facilitar no entendimento do projeto e para clarear o que se deseja em relação ao código.

c) baseia-se em um ciclo simples: escreve-se um código -> cria-se um teste para passar no código -> refatora-se.

d) propõe a criação de testes que validem o código como um todo para reduzir o tempo de desenvolvimento.

e) beneficia-se de testes que seguem o modelo FIRST: F (Fast) I (Isolated) R (Repeatable) S (Self-verifying) T (Timely).

36. **(CESPE / TRE/TO – 2017)** O TDD (Test-Driven Development), que vem sendo adotado para testar os projetos de software,

a) utiliza os testes de caixa preta antes da entrega do software.
b) agiliza os testes por amostragem sem compatibilidade retroativa.
c) cobre amplamente os testes unitários.

d) escreve o teste antes da codificação do software.
e) realiza refactoring antes de escrever a aplicação a ser testada.

37. **(FGV / IBGE – 2016)** O Desenvolvimento Orientado a Testes (TDD) é um método de desenvolvimento criado e disseminado por Kent Beck em seu livro "Test-driven development". O método define regras, boas práticas e um ciclo de tarefas com 3 etapas: a etapa vermelha, a etapa verde e a etapa de refatoração, ilustrado na imagem abaixo:

Com relação às regras e boas práticas de TDD e ao seu ciclo, é correto afirmar que:

a) pode-se escrever testes que não compilam na etapa vermelha;
b) na etapa verde deve-se escrever código que testa uma funcionalidade a fundo de forma criteriosa e detalhada;
c) código novo só é escrito se um teste automatizado passar;
d) a duplicação é tolerada na etapa de refatoração;
e) é uma boa prática de TDD iniciar o desenvolvimento do código de uma funcionalidade e, logo em seguida, testá-la.

38. **(IADES / EBSERH – 2013)** Assinale a alternativa que não corresponde a uma das fases do processo de desenvolvimento, dirigido a testes (TDD).

a) Executar o teste, com os outros testes implementados, que rodarão e fornecerão o resultado de que o software está sem problemas.
b) Escrever o teste para a funcionalidade e implementação.
c) Realizar a identificação do incremento de funcionalidade.
d) Implementar a funcionalidade e executar novamente o teste.
e) Implementar a próxima parte da funcionalidade, após todos os testes terem sido executados, com sucesso

39. **(PR-4 UFRJ / UFRJ – 2018)** O ciclo do TDD - Test Driven Development, ou, em português, Desenvolvimento Guiado por Testes consiste em:

a) implementar teste unitário falho, tornar o teste bem-sucedido e refatorar.
b) implementar a funcionalidade, executar teste unitário e refatorar.

c) implementar teste unitário falho, refatorar e tornar o teste bem-sucedido.
d) implementar a funcionalidade, refatorar e tornar o teste bem-sucedido.
e) refatorar, executar teste unitário e implementar a funcionalidade.

40. **(CESPE / STJ – 2015)** Um dos passos executados no ciclo de atividades do processo TDD é a criação de novos testes para as falhas encontradas no código original, sem alteração deste.

41. **(CS-UFG / AL-GO – 2015)** O desenvolvimento dirigido a testes (TDD, do Inglês Test-Driven Development) é uma abordagem de desenvolvimento de software na qual se intercalam testes e desenvolvimento de código. Uma das características da abordagem TDD é:

a) a sua utilidade no desenvolvimento de softwares novos.
b) o maior custo associado aos testes de regressão.
c) a redução da importância da automatização dos testes.
d) a sua adequação a processos de software sequenciais.

42. **(UECE-CEV / FUNCEME – 2018)** Test-driven Development (TDD) é uma abordagem para o desenvolvimento de programas em que se intercalam testes e desenvolvimento de código (Sommerville, I. Engenharia de Software, 9a edição, 2011).

A respeito do TDD, é correto afirmar que:

- a) consiste em um processo iterativo que se inicia escrevendo um código de uma funcionalidade do sistema e, logo em seguida, testa-o para saber se a implementação foi correta.
b) apesar de útil, não diminui o custo de testes de regressão do sistema.
c) sua utilização elimina a necessidade de testes de validação do sistema, uma vez que ele já foi testado incrementalmente.
d) apesar de ter sido apresentado como parte dos métodos ágeis, também pode ser usado em outros processos de desenvolvimento de software.

43. **(FAUGRS / UFRGS – 2018)** _____ é uma abordagem para o desenvolvimento de programas em que se intercalam testes e desenvolvimento de código. Essencialmente, desenvolve-se um código de forma incremental em conjunto com um teste para este incremento. Não se avança para o próximo incremento até que o código desenvolvido passe no teste. Essa abordagem foi introduzida como parte de métodos ágeis, mas pode ser também usada em processos de desenvolvimento dirigido a planos.

Assinale a alternativa que preenche corretamente a lacuna do texto acima.

a) Desenvolvimento Guiado por Testes (TDD)
b) Desenvolvimento em Espiral
c) Engenharia Dirigida a Modelos (MDD)
d) Rational Unified Process (RUP)
e) Teste de Sistema

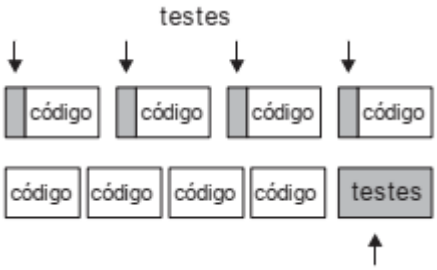
44. **(VUNESP / TCE-SP – 2015)** No Desenvolvimento Orientado a Testes (TDD), os casos de teste que definem o recurso a ser implementado devem ser elaborados:

a) assim que o código do teste estiver pronto.
b) antes de o código do recurso ser desenvolvido.
c) após o código do recurso ter sido completamente documentado.
d) simultaneamente com o desenvolvimento do código do recurso.
e) somente se o código do recurso apresentar erros.

45. **(IBFC / EMBASA – 2017)** No Ciclo de Desenvolvimento do TDD (Test-Driven Development), utiliza-se a estratégia que aplica três palavras-chaves (em inglês), que é denominada:

a) Red, Green, Refactor
b) White, Gray, Black
c) White, Black, Refactor
d) Green, Yellow, Red

46. **(FCC / CREMESP – 2016)** Considere a figura abaixo que apresenta duas abordagens de teste.



A figura:

a) ilustra as duas fases do TDD, que correspondem a escrever pequenos testes e testá-los no final.
b) mostra o ciclo conhecido como Vermelho-Verde-Refatora.
c) apresenta a diferença entre testes automatizados e testes manuais no XP.
d) mostra que um desenvolvedor que pratica TDD tem mais feedbacks do que um que escreve testes ao final.
e) evidencia que TDD é impraticável, pois o desenvolvedor gasta muito tempo escrevendo código de testes.

47. **(CESPE / ANATEL – 2014)** Em se tratando de desenvolvimento de softwares dirigidos a testes (TDD), a execução dos testes é realizada antes da implementação da funcionalidade.

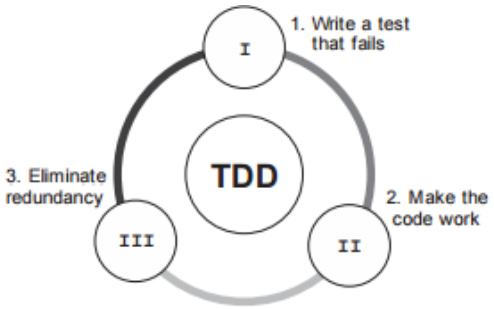
48. **(CESPE / TC/DF – 2014)** No TDD, o refatoramento do código deve ser realizado antes de se escrever a aplicação que deve ser testada.

49. **(CESPE / STJ – 2015)** No método de desenvolvimento TDD (Test Driven Development), o desenvolvedor escreve primeiro um caso de teste e, posteriormente, o código.

50. **(CESPE / MPE-PI – 2018)** O TDD possibilita o desenvolvimento de softwares fundamentado em testes. O ciclo de desenvolvimento do TDD segue os seguintes passos:

- escrever um teste que inicialmente não passa;
- adicionar uma nova funcionalidade do sistema;
- fazer o teste passar;
- realizar a integração contínua do código;
- escrever o próximo teste.

51. **(FCC / Pref. de Teresina/PI – 2016)** O Test Driven Development – TDD é uma das práticas sugeridas na eXtreme Programming – XP, onde o programador escreve o teste antes de escrever o código. O ciclo de desenvolvimento utilizando TDD é mostrado abaixo.



Considere:

I. Etapa inicial, onde se escreve um teste que falha, para alguma funcionalidade que ainda será Escrita.
II. Já com o teste criado, é o momento de executar o teste.
III. Eliminar códigos redundantes, remover acoplamentos, enfim, identificar pontos de melhoria no código.

As etapas I, II e III são, respectivamente,

a) Iniciação, Execução e Controle.
b) Red, Green e Refactor.
c) Iniciação, Atuação e Otimização.
d) Plan, Do e Check.
e) Planejamento, Execução e Melhoria.

52. **(FAUGRS / BANRISUL – 2018)** Considere as ações abaixo, executadas em desenvolvimento orientado a testes, Test-Driven Design (TDD).

I - Escrever código de teste.

II - Verificar se o teste falha.
III - Escrever código de produção.
IV - Executar teste até passar (reescrevendo o código de produção, se for necessário, até que o teste passe).
V - Refatorar código de produção e/ou de teste para melhorá-lo.

Considerando que se deseja incluir um novo caso de teste, assinale a alternativa que apresenta a sequência de ações que devem obrigatoriamente ocorrer para essa inclusão, segundo o TDD.

a) I, III e IV.
b) III, I e IV.
c) I, II, III e IV.
d) I, III, IV e V.
e) I, II, III, IV e V.

53. **(FGV / IBGE – 2017)** Test Driven Development (TDD) é uma prática muito utilizada no processo de desenvolvimento de sistemas computacionais. Analise as afirmativas a seguir sobre o uso da prática de TDD:

I. Tornam os testes de regressão mais demorados porque o desenvolvedor precisará fazer testes manuais várias vezes por dia.

II. Garante que os requisitos do sistema sejam atendidos porque o desenvolvedor escreverá o código de testes sempre que acabar a implementação do código do sistema.

III. Ajuda o desenvolvedor a escrever código de qualidade porque ele gastará parte do seu tempo escrevendo código de testes.

Está correto o que se afirma em:

a) somente I;
b) somente II;
c) somente III;
d) somente II e III;
e) I, II e III.

54. **(CESPE / ANATEL – 2014)** Na atividade de TDD (test-driven development), a escrita de teste primeiro define implicitamente tanto uma interface quanto uma especificação do comportamento para a funcionalidade que está sendo desenvolvida, estando, entretanto, a viabilidade do uso dessa abordagem limitada aos processos de desenvolvimento de software que seguem as práticas ágeis.

55. **(CESPE / TRE/MT – 2015)** Considere as seguintes etapas de um processo do tipo desenvolvimento orientado a testes (TDD).

I Implementar funcionalidade e refatorar.
II Identificar nova funcionalidade.
III Executar o teste.
IV Escrever o teste.
V Implementar a próxima parte da funcionalidade.

Assinale a opção que apresenta a sequência correta em que essas etapas devem ser realizadas.

a) I; IV; III; II; V
b) IV; III; II; I; V
c) I; IV; II; III; V
d) II; IV; III; I; V
e) IV; II; III; I; V

56. **(FCC / SEFAZ-SC – 2018)** O Test-Driven Development (TDD) é uma abordagem para o desenvolvimento de programas em que se intercalam testes e desenvolvimento de código. As etapas do processo fundamental de TDD são mostradas abaixo em ordem alfabética:

I. Escrever um teste para a funcionalidade identificada e implementá-lo como um teste automatizado.

II. Executar o teste, junto com os demais testes já implementados, sem implementar a nova funcionalidade no código.

III. Identificar e implementar uma outra funcionalidade, após todos os testes serem executados com sucesso.

IV. Identificar uma nova funcionalidade pequena para ser incrementada com poucas linhas em um código.

V. Implementar a nova funcionalidade no código e reexecutar o teste.

VI. Refatorar o código com melhorias incrementais até que o teste execute sem erros.

VII. Revisar a funcionalidade e o teste, caso o código execute sem falhar.

Considerando o item IV a primeira etapa e o item III a última etapa, a sequência intermediária correta das etapas do processo é:

a) I – II – VII – V e VI.
b) I – V – II – VII e VI.
c) I – VI – V – VII e II.
d) V – I – II – VII e VI.

e) V – I – VI – VII e II.

GABARITO – DIVERSAS BANCAS

1. LETRA A
2. ERRADO
3. CORRETO
4. ERRADO
5. LETRA B
6. ERRADO
7. LETRA C
8. LETRA B
9. LETRA A
10. LETRA D
11. ERRADO
12. CORRETO
13. ERRADO
14. LETRA B
15. LETRA A
16. CORRETO
17. ERRADO
18. ERRADO
19. CORRETO
20. LETRA E
21. ERRADO
22. CORRETO
23. CORRETO
24. CORRETO
25. LETRA C
26. LETRA A
27. LETRA E
28. CORRETO
29. CORRETO
30. LETRA D
31. LETRA A
32. LETRA D
33. LETRA B
34. LETRA E
35. LETRA E
36. LETRA D
37. LETRA A
38. LETRA A
39. LETRA A
40. ERRADO
41. LETRA A
42. LETRA D
43. LETRA A
44. LETRA B
45. LETRA A
46. LETRA D
47. CORRETO
48. ERRADO
49. CORRETO
50. ERRADO
51. LETRA B
52. LETRA C
53. LETRA C
54. ERRADO
55. LETRA D
56. LETRA A

BEHAVIOUR DRIVEN DEVELOPMENT (BDD)

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

No ano de 2003, Dan North desenvolveu o *Behaviour Driven Development* (BDD) como uma resposta aos famosos Test Driven Development (TDD) e Acceptance Test Driven Development (ATDD). Na verdade, eu diria que foi mais uma evolução do que uma resposta! **Essa metodologia de desenvolvimento ágil busca tornar as práticas destas outras metodologias mais acessíveis e intuitivas para os novatos e especialistas.**

O BDD colabora para que o desenvolvimento foque na entrega de valor, através da formação de um vocabulário comum, reduzindo a distância entre o Negócio e a TI. Em geral, o cliente deve prover à equipe informações sobre o problema que ele quer resolver, **e juntos podem pensar nos exemplos concretos que vão nortear o processo de desenvolvimento.** Galera, exemplos são essenciais nessa metodologia! *Por que, professor?*

Cara, vocês são estudantes, vocês sabem muito bem. *Qual a melhor maneira de entender algo?* Por meio de exemplos! **Dessa forma, é muito mais fácil entender um domínio de negócios complexo.** *Vocês já tiveram que modelar um domínio em que vocês não sacam nada? Que é tudo vago e obscuro?* Pois é, a luz no fim do túnel é o exemplo. O ser humano precisa de exemplos para compreender um tópico.

Exemplos reais são uma excelente forma de se comunicar, e no nosso dia-a-dia nós os usamos sem nem mesmo perceber. Ao trabalhar com exemplos reais, nós nos comunicamos melhor, pois as pessoas serão capazes de se relacionar com eles mais facilmente. **Isso tudo é muito mais perceptível quando o domínio do negócio é complexo.** O BDD busca melhorar a comunicação e a interação entre os stakeholders (clientes, programadores, analistas de qualidade, etc).

Fica claro como o software deve se comportar por meio de cenários escritos em linguagem natural. Na verdade, é a combinação da linguagem natural com uma linguagem ubíqua – que é uma linguagem comum entre programadores e especialistas no domínio (jargões técnicos, terminologias do dia-a-dia, entre outros). **Isso minimiza ruídos de comunicação, previne falhas e reduz riscos – este é o grande foco dessa metodologia.** Vamos ver as principais práticas:

PRÁTICAS DO BDD
Envolver as partes interessadas no processo através de Outside-in Development (Desenvolvimento de fora para dentro);
Usar exemplos para descrever o comportamento de uma aplicação ou unidades de código;
Automatizar os exemplos para prover um feedback rápido e possivelmente testes de regressão;

Usar "*deve*" na hora de descrever o comportamento ajuda a esclarecer responsabilidades e permitir que funcionalidades sejam questionadas;
Usar simuladores de teste (*Mocks*, *Stubs*, *Fakes*, *Dummies*, *Spies*) para auxiliar na colaboração entre módulos e códigos que ainda não foram escritos.

Professor, como eu posso escrever um cenário? Cara, é muito fácil! Primeiro, eu crio uma história – todos elas seguem mais ou menos o mesmo padrão:

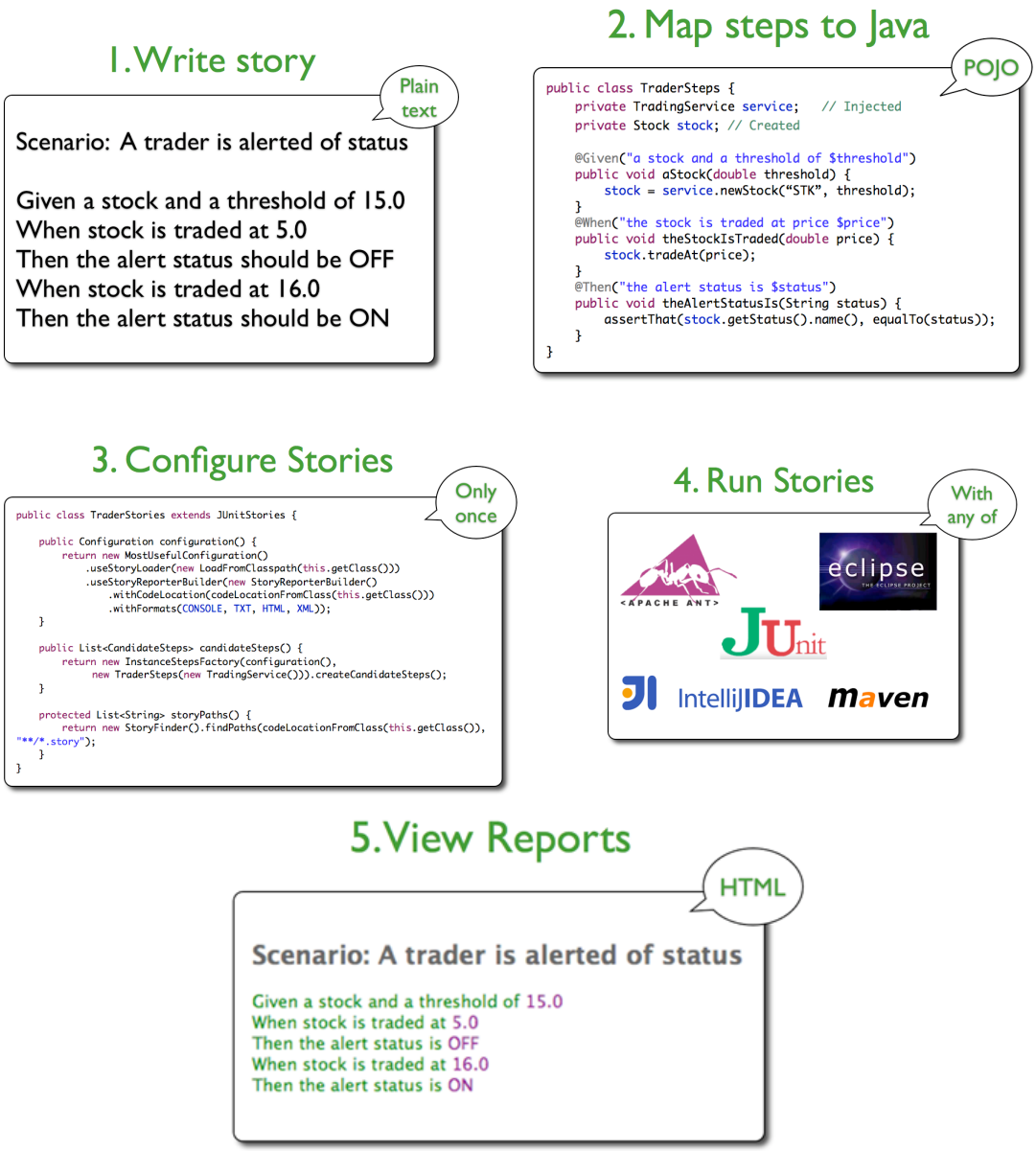
EU, COMO <PAPEL>, DESEJO <NECESSIDADE> PARA <MOTIVO>

Em outras palavras, pode ser entendido como: "*Eu, como Analista de Relacionamento, desejo poder manipular todas as informações de um chamado para que eu possa resolvê-lo*". **Pronto, essa é uma história que possui algum valor para o negócio!** No entanto, nosso papo não acabou! Baseado nessa história, eu devo criar um cenário – todos eles também seguem mais ou menos um mesmo padrão:

DADO QUE <CONTEXTO INICIAL>, QUANDO <EVENTO>, ENTÃO <RESULTADO>

Em outras palavras, pode ser entendido como: "*Dado que não existam chamados abertos, quando abrirem um chamado, então eu devo poder manipulá-lo*". Percebam que a história é apenas uma narrativa de base para a especificação de um ou mais cenários. **Vejam também que a linguagem é voltada para o domínio do negócio, sem detalhes técnicos.** *Entendido?* Agora vem a parte mais interessante: a comparação entre BDD x TDD!

Galera, vocês se lembram que eu falei para vocês que essa metodologia era uma evolução de TDD/ATDD? **Pois é, é possível automatizar testes escritos dessa maneira por meio de ferramentas específicas para realizar a validação do software, tais como: SpecFlow, JBehave, JSpec, RSpec, etc.** O JBehave, por exemplo, segue cinco passos bem definidos apresentados na imagem a seguir:



O BDD associa os benefícios de uma documentação formal, escrita e mantida pelo negócio, com testes de unidade que "demonstram" que essa documentação é efetivamente válida. Na prática, isso garante que a documentação deixa de ser um registro estático, que se converte em algo gradualmente ultrapassado, em um artefato dinâmico que reflete constantemente o estado atual de um projeto.

Legal, mas chegou o momento de resumir esse nosso papo! Se você estiver desesperado porque não conseguiu fechar o edital e precisa saber rapidamente apenas os fundamentos básicos dessa metodologia, eu vou facilitar sua vida e te mostrar o que você precisa saber em poucas palavras. **Vou fazer isso em um conjunto de pontos para que fique mais claro para vocês – espero que vocês compreendam. Vamos lá...**

- Behaviour-Driven Development (BDD) é uma técnica ágil de desenvolvimento de software focada em atender **funcionalidades ou comportamentos** desejados pelos usuários;

- Tentativa de **abarcar diversos conceitos**, tais como: Especificação por Exemplo; Definição de "Pronto"; Estórias de Usuário; Desenvolvimento Outside-In; TDD; ATDD; DDD; DSL; etc;

- Assim como o Test-Driven Development (TDD), essa técnica busca **escrever testes antes de codificar qualquer regra de negócio do sistema**;

- Em contraste com o Test-Driven Development (TDD), **testes são escritos sob o ponto de vista do usuário** e, não, sob o ponto de vista do desenvolvedor;

- Assim como o Acceptance Test-Driven Development (ATDD), essa técnica busca desenvolver o software **baseado em critérios automáticos de aceitação**;

- Assim como o Domain-Driven Development (DDD), **utiliza uma linguagem ubíqua**, i.e., um vocabulário comum entre usuários e desenvolvedores baseado no domínio do negócio;

- Isso permite uma **melhor comunicação/colaboração entre equipes**; compartilhamento de conhecimento; documentação dinâmica; visão holística do sistema; foco na entrega de valor;

- Apresenta um conjunto de cenários, protótipos e **especificações orientadas a exemplos**, que servem de entrada para o processo de desenvolvimento;

- TDD é similar a um Teste Caixa-Branca (especificado pelo desenvolvedor) e o **BDD é similar a um Teste Caixa-Preta (especificado pelo usuário)**;

(TRE/MA – 2015) Qual a alternativa correta a respeito do BDD?

a) Utiliza o conceito de Ubiquitous Language de modo a facilitar o entendimento de todos os fazendo com que os integrantes falem a mesma língua.

b) É uma técnica de desenvolvimento de testes que funciona exclusivamente se utilizada alguma metodologia ágil de desenvolvimento.

c) Técnica de testes onde somente os programadores participam, deixando analistas e testadores cuidando das demais atividades.

d) Maneira de testar os sistemas de acordo com o comportamento esperado, portando só pode ser feito quando o software estiver concluído.

Comentários: (a) Correto, nós vimos que ele realmente utiliza uma linguagem ubíqua de modo a facilitar a colaboração entre a equipe; (b) Errado, não é uma técnica de desenvolvimento de testes, é uma técnica de desenvolvimento de software orientada a testes comportamentais; (c) Errado, não é uma técnica de teste e não participam apenas programadores – participam todos

os membros da equipe, inclusive usuários; (d) Errado, é iterativo e incremental, logo pode ser feito enquanto o software está sendo construído (Letra A).

(TST – 2012) Leia o texto:

"O TDD – Test-Driven Development é focado em testes unitários, em que você isola um modelo (por exemplo) e monta-o de acordo com os testes que você escrever. Quando você tiver um determinado número de modelos, aí você testa a integração entre eles, e assim por diante. Fazendo uma analogia, isso é mais ou menos como construir o software "de dentro para fora", em que partimos escrevendo testes específicos (unitários) e depois vamos abrangendo outras regiões do sistema (integração, funcional, aceitação etc). Já em (.....) podemos dizer que o software é desenvolvido "de fora para dentro", já que os testes são construídos baseados nos requisitos do cliente ou em um roteiro de aceitação (também conhecidos por estórias). Esta prática é semelhante ao TDD: testes são escritos primeiro, funções depois. O diferencial é que (.....) aborda o comportamento e o resultado como um todo, não se preocupando, necessariamente, com as classes, métodos e atributos de forma isolada. Neste texto, foi omitida a referência à técnica conhecida como:

a) TFD – Test First Development
b) FTR – Formal Test Review
c) BDD – Behaviour-Driven Development
d) RTT – Real Time Test
e) FDT – Feature-Driven Test

_____

Comentários: essa é uma excepcional definição de Behaviour-Driven Development (Letra C).

(TRE/BA – 2017) Entre os métodos e técnicas ágeis, a técnica que utiliza a linguagem ubíqua, visando, entre outras coisas, a integração de regras de negócios com linguagem de programação, com foco no comportamento do software, e na qual os testes orientam o desenvolvimento, ou seja, primeiro se escreve o teste e depois o código, é a:

a) BDD (Behavior Driven Development).
b) Kanban.
c) automação de builds.
d) automação de testes.
e) TDD (Test Driven Development).

_____

Comentários: *Linguagem Ubíqua?* BDD! *Integração de regras de negócio com linguagem de programação?* BDD! *Testes orientam o desenvolvimento?* BDD ou TDD. Logo, só pode tratar do BDD (Letra A).

(BANRISUL – 2022) Desenvolvedores que se beneficiam das vantagens do BDD escrevem os testes em sua língua nativa, em combinação com a linguagem ubíqua.

_____

Comentários: BDD (Behavior Driven Development) é uma técnica de desenvolvimento usada para ajudar a criar software de forma colaborativa. Ele usa um formato de comunicação comum para descrever e validar a funcionalidade do software, permitindo que desenvolvedores, testadores, gerentes de projeto e outros participantes falem a mesma linguagem. Os testes são escritos em sua língua nativa, que é geralmente uma linguagem de programação, como Java ou C#. Esses testes então são traduzidos para uma linguagem ubíqua, como Gherkin, que é a linguagem usada para escrever o BDD. Essa linguagem é universalmente compreendida pelos participantes, independentemente de sua experiência com programação, permitindo que eles tomem parte ativa na definição de requisitos e na verificação (Correto).

(BANRISUL – 2022) Entre os métodos de testes ágeis, o TDD (Test-Driven Development) é uma extensão do BDD (Behavior Driven Development).

_____

Comentários: a questão inverteu os conceitos: BDD é uma extensão do TDD (Errado).

(BANRISUL – 2022) Entre os métodos de testes ágeis, o BDD (Behavior Driven Development) é aquele que, por natureza, é o mais orientado para o cliente.

_____

Comentários: BDD é uma abordagem de desenvolvimento orientada a comportamento. É uma estratégia de teste na qual os desenvolvedores de software, os clientes e os testadores colaboram para criar testes de software que descrevem o comportamento do sistema. O objetivo é criar testes baseados no comportamento esperado do software, que é entendido por todos os envolvidos (Correto).

(BANRISUL – 2022) No desenvolvimento orientado a comportamento (BDD), os ciclos iniciam-se com a criação de testes de unidade e integração.

_____

Comentários: BDD está relacionado a comportamentos de uma regra - quem se inicia com um teste de unidade é o TDD (Errado).

QUESTÕES COMENTADAS

1. **(FGV / CGM-BH – 2024)** Os benefícios da introdução do Behavior-Driven Development (BDD) em uma organização são significativos, ainda que sua implementação nem sempre ocorra sem dificuldades. Com relação aos desafios da introdução do BDD, analise as afirmativas a seguir e assinale (V) para a verdadeira e (F) para a falsa.

   I. O BDD requer um alto envolvimento e colaboração empresarial. As práticas de BDD são baseadas em conversas e feedback dos usuários, que impulsionam e constroem a compreensão da equipe sobre os requisitos e sobre como eles podem agregar valor ao negócio com base nesses requisitos.

   II. O BDD funciona melhor com a adoção de metodologias ágeis ou iterativa. As práticas de análise de requisitos do BDD mostram que é difícil, se não impossível, definir completamente os requisitos de modo antecipado, e que estes evoluirão à medida que a equipe aprenda mais sobre o projeto.

   III. Os testes, mesmo que mal escritos, não ocasionam custos de manutenção elevados. A criação de testes automatizados, especialmente para aplicações web complexas, requer baixa habilidade, e as equipes que estão começando a adotar o BDD não consideram isso um desafio significativo.

   As afirmativas são, respectivamente,

   a) F – V – F.
   b) V – F – V.
   c) V – V – F.
   d) F – F – V.

Comentários:

(V) Correto. BDD requer envolvimento e colaboração empresarial; (V) Correto. BDD funciona melhor com metodologias ágeis ou iterativas, e os requisitos evoluem à medida que a equipe aprende mais sobre o projeto; (F) Errado. Testes mal escritos podem ocasionar altos custos de manutenção, e a criação de testes automatizados para aplicações web complexas requer habilidades significativas.

Gabarito: Letra C

---

2. **(CESPE / CNPq – 2024)** Ao se utilizar a metodologia BDD, espera-se que as entregas sejam feitas com menor quantidade de retrabalho e com cobertura de testes automatizados, a

documentação passe a ser integrada ao sistema, e os projetos passem a ter estimativas dentro do prazo, com custos controlados.

Comentários:

O BDD promove a colaboração entre desenvolvedores, testadores e não técnicos para definir os requisitos usando exemplos concretos, resultando em menor retrabalho e maior cobertura de testes automatizados. A documentação se torna parte integrante do sistema, pois os cenários de teste são escritos em linguagem natural. Com requisitos claros e bem definidos, os projetos tendem a ter estimativas mais precisas, prazos cumpridos e custos controlados.

Gabarito: Correto

---

3. **(CESPE / SEFIN de Fortaleza-CE – 2023)** No desenvolvimento orientado por comportamento (BDD), as palavras-chave utilizadas nos blocos que formam os cenários são given, when e then.

Comentários:

No Desenvolvimento Orientado por Comportamento (BDD), os cenários são descritos usando as palavras-chave Given (Dado), When (Quando), e Then (Então).

- Given estabelece o contexto inicial ou pré-condições.
- When descreve a ação ou evento que ocorre.
- Then define o resultado esperado ou as consequências.

Esse formato ajuda a criar uma linguagem comum entre desenvolvedores, testers e stakeholders, facilitando o entendimento dos requisitos e expectativas do sistema.

Gabarito: Correto

QUESTÕES COMENTADAS

1. **(FGV / CGM-BH – 2024)** Os benefícios da introdução do Behavior-Driven Development (BDD) em uma organização são significativos, ainda que sua implementação nem sempre ocorra sem dificuldades. Com relação aos desafios da introdução do BDD, analise as afirmativas a seguir e assinale (V) para a verdadeira e (F) para a falsa.

   I. O BDD requer um alto envolvimento e colaboração empresarial. As práticas de BDD são baseadas em conversas e feedback dos usuários, que impulsionam e constroem a compreensão da equipe sobre os requisitos e sobre como eles podem agregar valor ao negócio com base nesses requisitos.

   II. O BDD funciona melhor com a adoção de metodologias ágeis ou iterativa. As práticas de análise de requisitos do BDD mostram que é difícil, se não impossível, definir completamente os requisitos de modo antecipado, e que estes evoluirão à medida que a equipe aprenda mais sobre o projeto.

   III. Os testes, mesmo que mal escritos, não ocasionam custos de manutenção elevados. A criação de testes automatizados, especialmente para aplicações web complexas, requer baixa habilidade, e as equipes que estão começando a adotar o BDD não consideram isso um desafio significativo.

   As afirmativas são, respectivamente,

   a) F – V – F.
   b) V – F – V.
   c) V – V – F.
   d) F – F – V.

2. **(CESPE / CNPq – 2024)** Ao se utilizar a metodologia BDD, espera-se que as entregas sejam feitas com menor quantidade de retrabalho e com cobertura de testes automatizados, a documentação passe a ser integrada ao sistema, e os projetos passem a ter estimativas dentro do prazo, com custos controlados.

3. **(CESPE / SEFIN de Fortaleza-CE – 2023)** No desenvolvimento orientado por comportamento (BDD), as palavras-chave utilizadas nos blocos que formam os cenários são given, when e then.

GABARITO

1. LETRA C
2. CORRETO
3. CORRETO

DOMAIN DRIVEN DESIGN (DDD)

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

O Domain Driven Design (DDD[1]) é uma abordagem de desenvolvimento de software para lidar com comportamentos complexos na construção de um software. Ele tem três premissas: focar o projeto no domínio principal e no domínio lógico; basear projetos complexos em um modelo do domínio; e iniciar uma colaboração com os especialistas técnicos e de domínio para refinar o modelo conceitual.

Deve-se saber ordenar suas prioridades com o objetivo de acelerar os projetos de software que tenham que lidar com domínios complexos. Para tal, a equipe necessita de um conjunto de práticas, técnicas e princípios. *Professor, mas você está falando tanto de Domínio e eu nem sei o que é isso!* Domínio é uma área de conhecimento, isto é, é o assunto do problema. *E um modelo? O que seria, professor?*

Modelo é um sistema de abstrações que descrevem aspectos específicos de um domínio e pode ser utilizado para resolver problemas relacionados a ele. Nessa abordagem, não se pode ter um domínio não trivial. Além disso, a equipe deve ser experiente na utilização do paradigma orientado a objetos e eles devem ter contato com os especialistas do domínio. Por fim, trata-se de um processo iterativo!

(STJ – 2015) Domain-Driven Design pode ser aplicada ao processo de concepção arquitetural de um sistema de software, sendo que domain, em um software, designa o campo de ação, conhecimento e influência.

Comentários: "domain" refere-se ao campo de conhecimento e atividades relacionados ao negócio que o software deve atender. DDD é frequentemente aplicado na concepção arquitetural de sistemas, garantindo que a arquitetura do software esteja alinhada com as regras e necessidades específicas do domínio do negócio (Correto).

Para ter um software que atenda perfeitamente a um determinado domínio, é necessário que se estabeleça, em primeiro lugar, uma Linguagem Ubíqua. Nessa linguagem estão termos que fazem parte das conversas diárias entre especialistas de negócio e times de desenvolvimento. Todos devem usar os mesmos termos tanto na linguagem falada quanto no código. Essa linguagem ubíqua deve ser compreendida por todos e não pode haver ambiguidades.

[1] Também chamado Domain-Driven Programming (DDP).

(STJ – 2015) Um dos princípios-chave do Domain-Driven Design é o uso de uma linguagem ubíqua com termos bem definidos, que integram o domínio do negócio e que são utilizados entre desenvolvedores especialistas de negócio.

_____

Comentários: Para ter um software que atenda perfeitamente a um determinado domínio, é necessário que se estabeleça, em primeiro lugar, uma Linguagem Ubíqua. De fato, essa é a linguagem comum do domínio do negócio e dos desenvolvedores especialistas do negócio (Correto).

(SEFAZ-MG – 2023) Em domain-driven design (DDD), a linguagem ubíqua ou linguagem onipresente é um conceito central. Assinale a opção que indica seu principal objetivo:

a) Documentar o domínio.
b) Providenciar um meio de comunicação com os especialistas do domínio.
c) Facilitar a comunicação entre especialistas e desenvolvedores.
d) Criar uma linguagem compartilhada que é usada durante o processo de desenvolvimento.
e) Criar convenções de nomenclatura.

_____

Comentários: (a) Errado. Documentar o domínio não é o principal objetivo da linguagem ubíqua, embora a documentação possa ser um subproduto; (b) Errado. Providenciar um meio de comunicação com os especialistas do domínio é importante, mas a linguagem ubíqua vai além disso, integrando a comunicação entre todos os envolvidos; (c) Errado. Facilitar a comunicação entre especialistas e desenvolvedores é um aspecto da linguagem ubíqua, mas não define todo o seu propósito; (d) Correto. A linguagem ubíqua visa criar uma linguagem compartilhada que é usada por toda a equipe de desenvolvimento, desde especialistas do domínio até desenvolvedores, garantindo consistência e clareza; (e) Errado. Embora a criação de convenções de nomenclatura possa ser parte da linguagem ubíqua, o principal objetivo é a criação de uma linguagem comum utilizada em todo o processo de desenvolvimento. (Letra D).

(BACEN – 2013) A linguagem ubíqua utiliza termos que fazem parte das discussões entre os especialistas de negócio e as equipes de desenvolvimento, os quais devem utilizar a mesma terminologia na linguagem falada e no código:

_____

Comentários: Perfeito! É muito importante que todos os envolvidos no desenvolvimento do sistema usem a mesma linguagem, que ele chama de Linguagem Ubíqua (Ubiquitous Language). Desta forma, conseguimos diminuir o abismo que existe entre programadores e especialistas no negócio, facilitando comunicação e diminuindo os clássicos problemas do "telefone sem fio" (Correto).

Toda vez que alguém perceber que um determinado conceito do domínio possui várias palavras que o represente, essa pessoa deve tentar readequar tanto a linguagem falada e escrita, quanto o código. Outra forma de colocar o modelo para funcionar é utilizando Model Driven Design (MDD). **O DDD utiliza princípios como alinhamento do código ao negócio, favorecimento da reutilização de código, mínimo acoplamento e independência de tecnologia.**

(SLU/DF – 2019) No desenvolvimento embasado em domain-driven design, a definição da tecnologia a ser utilizada tem importância secundária no projeto.

_____

Comentários: o foco principal é no domínio do problema e na modelagem dos conceitos que representam esse domínio. A tecnologia é escolhida posteriormente para implementar a solução, mas não é o fator determinante no design do sistema. A prioridade é entender e refletir as regras e necessidades do negócio no software. (Correto).

As pessoas que trabalham na construção de aviões, por exemplo, têm uma visão limitada do que, de fato, constitui o avião. Elas enxergam o avião como um apanhado de peças, as quais precisam ser colocadas juntas. No entanto, um avião é muito mais do que isso. **Um avião começa com um projeto, com um bom projeto que é formulado e reformulado centenas de vezes durante anos antes de atingir a perfeição.**

Somente após a criação de modelos em escala, testes aerodinâmicos, testes de situações críticas e mais, as peças que formam o avião serão construídas e postas em suas posições. **É possível observar algumas semelhanças entre esse exemplo e o desenvolvimento de software.** O desenvolvedor não pode, e mais do que isso, não consegue sentar e escrever código. É preciso entender o propósito do software para criar sistemas com qualidade.

Não é possível criar um sistema para uma seguradora sem entender como funcionam seus processos de negócio, o domínio no qual a ela está contida. **O domínio é onde devemos começar e os especialistas de negócio que trabalham na seguradora são as pessoas que mais entendem daquele domínio.** Eles conhecem todas as características do negócio, todas as peculiaridades, problemas e segredos – eles têm grande experiência naquilo que fazem todos os dias há anos.

Estas pessoas sabem o que deve ser feito, como o software deve se comportar, mas não sabem como construir o software de fato. Isso porque o negócio deles não é construir software; é a oferta e a contratação de seguros. **Os desenvolvedores, arquitetos e analistas de requisitos sabem como construir software, porque este é domínio do negócio deles, mas não o ramo de seguros.** O projeto de um software deve estar fundamentado no domínio em que irá atuar.

O objetivo central de um software é melhorar a utilização ou aplicação dos processos de negócio de um domínio específico. **Para que isso seja possível o software precisa estar harmoniosamente ligado com seu domínio. Essa é a sacada do Desenho Orientado a Domínio**. A melhor forma para conseguir um bom relacionamento entre o software e o domínio é projetar o software de maneira que ele reflita exatamente o domínio.

É preciso incorporar os conceitos chave, os elementos do domínio e o relacionamento entre eles. O software precisa ser a representação do domínio, caso contrário não reagirá bem às mudanças que certamente ocorrerão no negócio ao qual se aplica. Muito do conhecimento adquirido é fruto de conversas aparentemente intermináveis com os especialistas de negócio. Todo este conhecimento, certamente não ficará armazenado na cabeça dos analistas.

É preciso então, transformar tudo aquilo que foi conversado em um modelo passível de implementação. *Como representar isso?* Por meio de Regras de Negócio! As Regras de Negócio realizam os comportamentos necessários para que os objetivos da atividade de negócio sejam

atingidos. Vejamos na tabela apresentada a seguir algumas das principais características do Domain-Driven Design:

CARACTERÍSTICAS	DESCRIÇÃO
FOCO NO DOMÍNIO	O DDD coloca o domínio do negócio no centro do desenvolvimento, garantindo que o software atenda às necessidades específicas do negócio. **Exemplo:** uma empresa desenvolvedora de software está criando um sistema de gestão de hospital. O foco no domínio significa que a equipe de desenvolvimento, em colaboração com especialistas da área da saúde (médicos, enfermeiros, administradores hospitalares), concentra-se em entender profundamente como o hospital opera, quais são os processos críticos, como o atendimento ao paciente é gerido, e quais são as necessidades específicas do hospital. Esse entendimento profundo do domínio hospitalar guia todas as decisões de design e desenvolvimento do sistema.
MODELO DE DOMÍNIO	Um modelo conceitual que captura os principais conceitos e regras do domínio, usado para guiar o desenvolvimento do software. **Exemplo:** no sistema de gestão de hospital, o Modelo de Domínio pode incluir abstrações como Paciente, Médico, Enfermeiro, Consulta, Prescrição e Leito. Esses são conceitos que representam os elementos centrais do domínio hospitalar. O modelo de domínio é um diagrama ou uma estrutura que descreve como esses elementos se relacionam e interagem entre si. Por exemplo, um Paciente pode ter várias Consultas, e cada Consulta é realizada por um Médico.
LINGUAGEM UBÍQUA	Uma linguagem compartilhada entre todos os envolvidos no projeto (desenvolvedores, especialistas de domínio, stakeholders) para evitar mal-entendidos e garantir clareza. **Exemplo:** A equipe de desenvolvimento e os especialistas em saúde decidem que sempre usarão os termos "Paciente", "Médico", "Consulta", "Prescrição" e "Leito" exatamente da mesma maneira durante as reuniões, no código e na documentação. Isso elimina ambiguidades e garante que todos, desde desenvolvedores até médicos, estejam falando a mesma "língua". Quando alguém menciona uma "Consulta", todos sabem exatamente o que isso significa, tanto no contexto do hospital quanto no código do software.
BOUNDED CONTEXT	Delimitação clara de diferentes partes do domínio, onde termos e conceitos podem ter significados específicos e não ambíguos. **Exemplo:** no sistema de gestão de hospital, a equipe reconhece que o conceito de Paciente pode variar em diferentes partes do sistema. No contexto da administração hospitalar, um Paciente pode ser apenas uma entidade com informações de contato e seguro. No contexto do sistema de prontuário eletrônico, o Paciente pode ter um histórico médico completo, com diagnósticos, tratamentos e exames. Esses dois conceitos de Paciente são implementados em Bounded Contexts diferentes, garantindo que cada contexto trate apenas das preocupações que lhe dizem respeito, sem interferir ou conflitar com outros contextos.
ENTIDADES	Objetos do domínio que têm identidade única e um ciclo de vida distinto, refletindo elementos chave do negócio. **Exemplo:** no sistema hospitalar, o Paciente é uma entidade porque tem uma identidade única e pode mudar ao longo do tempo. Um Paciente pode mudar de endereço, ser internado, ou receber novos diagnósticos, mas ainda é identificado pelo mesmo número de registro ou CPF. A identidade do Paciente é o que define a entidade, mesmo que seus atributos ou estado mudem.

OBJETOS DE VALOR	Objetos definidos por seus atributos, e não por uma identidade única. Eles são imutáveis e dois objetos de valor com os mesmos atributos são considerados iguais. **Exemplo:** no sistema, o endereço do Paciente pode ser representado como um Objeto de Valor. Um Endereço pode incluir atributos como Rua, Cidade, Estado e CEP. Diferente de uma entidade, um Objeto de Valor não tem identidade própria – dois endereços com os mesmos atributos são considerados iguais. Se o Paciente se muda para um novo endereço, o antigo endereço não precisa ser rastreado individualmente; ele é simplesmente substituído.
SERVIÇOS	São usados para modelar operações do domínio que não se encaixam naturalmente como responsabilidade de uma entidade ou de um objeto de valor. **Exemplo:** o sistema de gestão hospitalar pode incluir um serviço para calcular o total de custos de uma internação. Este serviço, chamado CalculoDeCustoDeInternacao, não pertence a nenhuma entidade ou objeto de valor específico. Em vez disso, ele realiza uma operação importante no domínio, utilizando informações de várias entidades como Paciente, Internação, Tratamento e Médico. O serviço encapsula a lógica de negócio que não se encaixa naturalmente dentro de uma entidade ou objeto de valor, mas ainda é essencial para o domínio.

Se uma operação pertence claramente ao comportamento de uma entidade ou de um objeto de valor, então essa operação deve ser um método ou comportamento dessa entidade ou objeto de valor. A esse comportamento, nós chamamos de Resposabilidade Natural. Por exemplo: "alterar o endereço de um cliente" é uma responsabilidade natural de uma entidade "Cliente". É bastante intuitivo. *Concordam?*

Quando uma operação é importante para o domínio, mas não pertence a uma entidade específica ou não está relacionada à mudança de estado de um objeto de valor, essa operação é modelada como um serviço de domínio. O serviço é uma interface autônoma que encapsula essa lógica de negócio. Imaginem, por exemplo, o contexto de um sistema de contabilidade de uma organização qualquer.

Calcular os impostos de uma venda pode envolver várias entidades (Cliente, Produto, Venda) e não se enquadra diretamente como responsabilidade de nenhuma delas. Nesse caso, "Calcular Imposto" seria modelado como um serviço. Logo, serviços são essenciais para manter a modelagem do domínio limpa e coerente, garantindo que cada elemento (entidade, objeto de valor, serviço) tenha responsabilidades claras e bem definidas.

Isso permite que a lógica de negócio seja bem organizada e alinhada com o domínio real, facilitando a manutenção e evolução do sistema.

Saiba mais:

Imagine que você está construindo uma casa. O Domain-Driven Design (DDD) seria como o processo de construção onde cada parte da casa é projetada e construída com base nas necessidades e no estilo de vida dos futuros moradores, em colaboração com um arquiteto especializado.

A Casa como o Software: a casa representa o software que você está desenvolvendo. Assim como uma casa deve atender às necessidades diárias dos moradores, o software deve atender às necessidades do negócio ou do domínio específico para o qual está sendo criado.

O Arquiteto como o Especialista em Domínio: o arquiteto desempenha o papel de um especialista em domínio. Ele entende profundamente o que os moradores (os usuários ou clientes) precisam, como eles vivem, suas preferências e hábitos. O especialista em domínio é alguém que conhece intimamente o negócio para a qual o software está sendo desenvolvido.

Os Projetos como o Modelo de Domínio: antes de qualquer construção começar, o arquiteto cria projetos detalhados. Esses projetos são como o modelo de domínio no DDD. Eles capturam a essência do que a casa (o software) precisa ser, definindo a estrutura, os materiais e o layout. O modelo de domínio captura os conceitos fundamentais do negócio e os transforma em componentes de software.

Os Cômodos como Bounded Contexts: cada cômodo da casa tem uma função específica - a cozinha para cozinhar, o quarto para dormir, a sala para socializar. No DDD, cada função ou área do negócio é tratada como um Bounded Context. Assim como os cômodos da casa são projetados para atender a diferentes necessidades, os Bounded Contexts são delimitados para encapsular partes do domínio que funcionam de forma independente, mas que se conectam de maneira coerente.

A Linguagem Compartilhada: durante o processo de construção, é crucial que o arquiteto e os futuros moradores usem uma linguagem comum. Quando discutem sobre a casa, todos devem entender o que significa "suíte", "cozinha gourmet" ou "espaço aberto". No DDD, isso é chamado de Linguagem Ubíqua. Todos - desenvolvedores, especialistas em domínio, e clientes - usam a mesma terminologia para evitar mal-entendidos e garantir que o software atenda às expectativas.

Os Construtores como os Desenvolvedores: os construtores são os desenvolvedores de software. Eles pegam o projeto detalhado criado pelo arquiteto e o transformam em uma casa real. No DDD, os desenvolvedores usam o modelo de domínio e a linguagem ubíqua para criar um software que reflete com precisão o negócio.

Flexibilidade: durante a construção, os moradores podem perceber que precisam de uma mudança - talvez uma parede deva ser movida ou uma janela adicionada. Como o arquiteto está envolvido, essas mudanças são gerenciadas para garantir que a casa ainda funcione como um todo. No DDD, o modelo de domínio é flexível o suficiente para adaptar-se às mudanças no negócio sem comprometer a integridade do software.

No final, você não está apenas construindo uma casa qualquer; você está criando um lar personalizado que atende perfeitamente às necessidades dos moradores. Da mesma forma, com DDD, você não está apenas desenvolvendo software; você está criando uma solução que se encaixa perfeitamente no domínio específico do negócio, com cada componente projetado para refletir e atender às necessidades reais e dinâmicas do negócio.

Tipos de Relacionamento

INCIDÊNCIA EM PROVA: BAIXA

Os Bounded Contexts representam limites claros dentro de um domínio onde uma linguagem ubíqua e regras específicas são aplicadas. Esses contextos podem interagir entre si de várias maneiras, formando diferentes tipos de relacionamentos. Vejamos os principais tipos:

TIPOS DE RELACIONAMENTOS	DESCRIÇÃO
SHARED KERNEL	Dois ou mais Bounded Contexts compartilham uma parte do modelo de domínio, geralmente uma pequena porção essencial que ambos precisam utilizar de forma consistente. Nesse relacionamento, as equipes precisam trabalhar de maneira colaborativa e coordenada para evitar mudanças que afetem os contextos compartilhados negativamente. Exemplo: Dois contextos que dependem do mesmo modelo de "Cliente", onde ambos precisam garantir que as informações do cliente estejam consistentes.
COSTUMER-SUPPLIER	Um contexto (Supplier/Fornecedor) fornece dados ou funcionalidades para outro contexto (Customer/Cliente). No relacionamento Customer-Supplier, o contexto fornecedor precisa garantir que suas funcionalidades atendam às necessidades do contexto cliente, mas sem dependência direta entre eles. O fornecedor tem mais controle sobre seu próprio modelo, enquanto o cliente depende das informações fornecidas por ele. Exemplo: Um contexto de "Gestão de Produtos" fornece informações sobre produtos para um contexto de "Gestão de Pedidos", que depende desses dados para realizar operações.
CONFORMIST	Um contexto é dependente de outro e, ao contrário do Customer-Supplier, ele simplesmente aceita o modelo e as regras do outro sem impor requisitos. O contexto conformista se ajusta ao modelo fornecido pelo contexto fornecedor, usando a mesma linguagem e regras de negócio. Isso geralmente ocorre quando o contexto dependente não tem poder para exigir mudanças. Exemplo: Um contexto de "Relatórios" que precisa de dados de um contexto de "Transações" e adota o modelo de dados e as regras conforme fornecido.
ANTI-CORRUPTION LAYER	Uma camada é criada para proteger o modelo de um contexto dos efeitos de outro contexto. Essa camada adapta e converte dados entre os contextos para evitar que um contexto "contamine" o outro. É útil quando um contexto precisa interagir com um sistema legado ou um contexto com um modelo muito diferente. A camada anti-corrupção traduz e mapeia os dados, protegendo o modelo do contexto principal. Exemplo: Um contexto moderno de "Pedidos Online" interage com um contexto legado de "Gestão de Pedidos", e uma camada anti-corrupção converte dados para evitar que o modelo do sistema legado afete o modelo do contexto moderno.
SEPARATE WAYS	Dois contextos que têm pouca ou nenhuma dependência e podem evoluir de forma completamente independente. Os contextos operam de maneira totalmente isolada, cada um com seu próprio modelo e sem necessidade de integração direta. Essa abordagem é aplicada quando as operações dos contextos são independentes e não há ganho significativo em integrá-los.

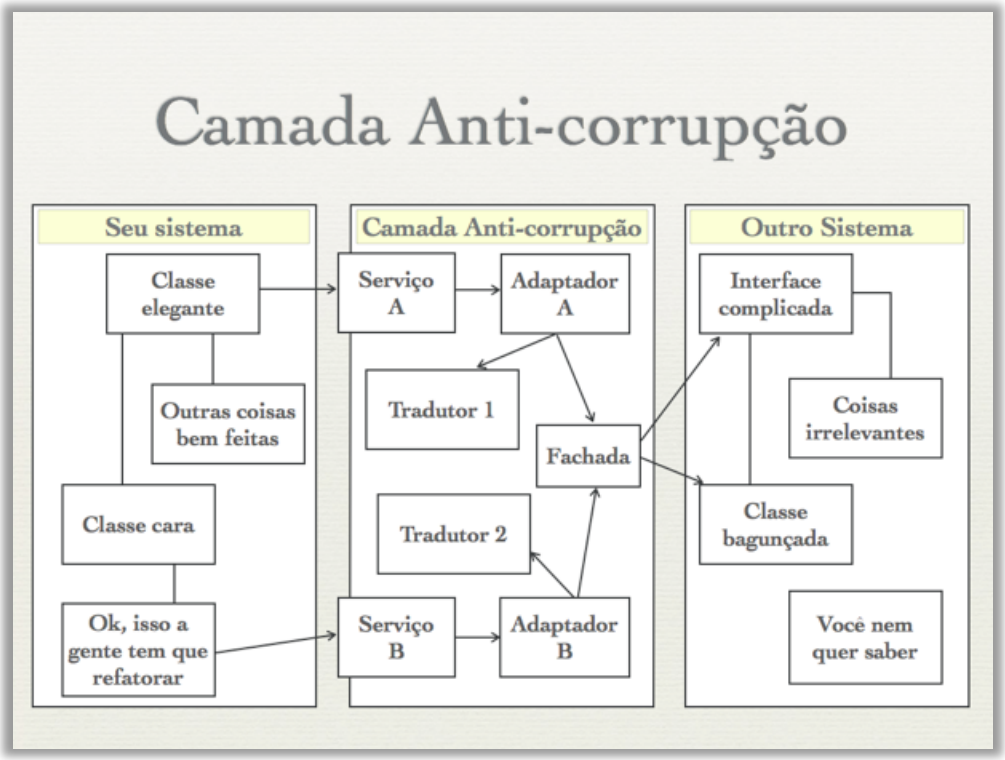
	Exemplo: Um contexto de "RH" e um contexto de "Gestão de Estoque" que não compartilham dados e não têm dependências entre si.
PARTNERSHIP	Dois contextos trabalham juntos com uma relação próxima e simétrica para atender a uma necessidade de negócio compartilhada. Os contextos parceiros colaboram ativamente, cada um com responsabilidades distintas, mas trabalhando de maneira coordenada para manter os modelos alinhados. Isso exige comunicação contínua entre as equipes para garantir que mudanças em um contexto sejam compatíveis com o outro. Exemplo: Um contexto de "Vendas" e um contexto de "Marketing" que colaboram para oferecer experiências integradas ao cliente.

Camada Anticorrupção

INCIDÊNCIA EM PROVA: BAIXA

No DDD, existe um padrão arquitetural que serve como uma barreira protetora entre diferentes sistemas ou bounded contexts para evitar que o modelo de um domínio seja "corrompido" por outro. A ideia é proteger o modelo de domínio principal de influências externas que possam impor conceitos, terminologias ou estruturas que não sejam consistentes com o modelo interno desejado. O nome desse padrão é: Camada Anticorrupção. Vejamos suas principais funções:

FUNÇÕES DA CAMADA ANTICORRUPÇÃO	DESCRIÇÃO
ISOLAMENTO DO DOMÍNIO	A camada anticorrupção isola o modelo de domínio interno do modelo de outro sistema ou bounded context. Isso garante que o domínio central permaneça puro e alinhado com as necessidades do negócio, sem ser influenciado por modelos externos que possam ter conceitos conflitantes ou inadequados.
TRADUÇÃO DE MODELOS	A camada anticorrupção atua como um tradutor entre os dois sistemas ou bounded contexts. Ela converte os dados e comandos de um modelo externo para a terminologia e estruturas apropriadas ao modelo de domínio interno. Por exemplo, se um sistema externo usa o termo "cliente" para algo que, no modelo interno, é melhor representado como "usuário", a ACL fará essa conversão.
ADAPTADORES E FAÇADES	Implementações típicas de uma camada anticorrupção incluem adaptadores, facades ou outros padrões que encapsulam a lógica de conversão. Isso permite que o sistema interno interaja com sistemas externos sem precisar adaptar seu próprio modelo de domínio, mantendo assim a integridade do seu design.
MINIMIZAÇÃO DE DEPENDÊNCIAS	A ACL reduz o acoplamento entre sistemas, permitindo que o modelo de domínio interno evolua independentemente dos sistemas externos. Isso é crucial para manter a flexibilidade e a capacidade de adaptação às mudanças do negócio.



É recomendável utilizar essa camada quando: o sistema precisa interagir com sistemas legados que não seguem os mesmos padrões ou que têm um modelo de dados que não se alinha bem ao modelo de domínio atual; há necessidade de se integrar com serviços de terceiros que impõem suas próprias terminologias e estruturas de dados; o modelo de domínio interno precisa ser protegido contra influências externas que possam introduzir inconsistências ou complexidade desnecessária.

Imagine que sua empresa está desenvolvendo um novo sistema de CRM (Customer Relationship Management), mas precisa integrar esse sistema com um ERP (Enterprise Resource Planning) legado. O ERP utiliza terminologias e estruturas de dados que não se encaixam bem no modelo de domínio do CRM. **Para evitar que o modelo de domínio do CRM seja "poluído" pelas terminologias e estruturas inadequadas do ERP, você implementa uma camada anticorrupção.**

Esta camada traduz as solicitações e dados entre o CRM e o ERP, garantindo que o CRM mantenha um modelo de domínio limpo e alinhado com os objetivos do negócio, enquanto ainda consegue se comunicar efetivamente com o ERP legado. A Camada Anticorrupção é uma estratégia essencial no DDD para proteger a integridade e a pureza do modelo de domínio ao integrar sistemas externos ou trabalhar com Bounded Contexts que têm diferentes modelos. Vamos continuar...

(MPOG – 2013) De acordo com os padrões de DDD (Domain-Driven Design), ao se escrever um novo sistema para também interagir com um sistema legado (considerado um código de difícil manutenção), cria-se uma camada entre os dois sistemas denominada camada anticorrupção.

_____

Comentários: quando temos um sistema legado, com código muito bagunçado e uma interface complexa, e estamos escrevendo um sistema novo com o código razoavelmente bem feito, criamos uma camada entre esses dois sistemas. O nosso sistema novo e bem feito falará com essa camada, que possui uma interface bem feita. E a camada anti-corrupção é responsável por traduzir e adaptar as chamadas para o sistema legado, usando uma fachada interna, ou seja, ela funciona como um adaptador (Correto).

No contexto do Domain-Driven Design (DDD), temos um conceito importante: Domain Specific Language (DSL). Trata-se de uma linguagem projetada especificamente para um domínio particular de aplicação, com o objetivo de facilitar a comunicação entre especialistas do domínio (como analistas de negócios ou clientes) e desenvolvedores de software. A DSL ajuda a capturar e representar o conhecimento do domínio de forma clara e precisa dentro do código. Vejamos:

CARACTERÍSTICAS DA DSL	DESCRIÇÃO
FOCADA NO DOMÍNIO	A DSL é projetada para expressar conceitos e operações que são específicos do domínio em que a aplicação opera. Ela usa a terminologia e a semântica familiar para os especialistas no domínio, o que torna o código mais intuitivo e fácil de entender para todos os envolvidos.
COMUNICAÇÃO EFICAZ	Uma DSL facilita a comunicação entre desenvolvedores e especialistas do domínio, garantindo que o software reflita com precisão as regras e os processos do negócio. Isso minimiza o risco de mal-entendidos e erros na implementação.
EXPRESSIVIDADE	A DSL permite expressar complexidades do domínio de maneira concisa e direta, capturando a lógica de negócios sem a necessidade de construções de linguagem de programação geral que poderiam obscurecer o significado.
ABSTRAÇÃO	A DSL abstrai detalhes técnicos que não são relevantes para o domínio, concentrando-se em como o domínio funciona. Isso ajuda a manter o código focado nas regras de negócio e nas funcionalidades principais, sem distrações técnicas.

Existem dois tipos de DSL: Interna e Externa. Vejamos:

TIPOS	DESCRIÇÃO
INTERNA	É implementada dentro de uma linguagem de programação existente, aproveitando a sintaxe e as capacidades dessa linguagem para criar uma DSL específica. Por exemplo, muitas bibliotecas em Ruby ou Python são consideradas DSLs internas porque usam a sintaxe da linguagem para expressar operações de domínio de forma clara. Exemplo: Ruby, uma DSL interna pode ser criada para definir regras de validação de um modelo de dados usando uma sintaxe natural como *validates_presence_of :name*.
EXTERNA	É uma linguagem separada, com sua própria sintaxe e gramática, criada especificamente para o domínio em questão. Essa DSL precisa ser interpretada ou compilada para ser utilizada dentro de uma aplicação. SQL pode ser considerado uma DSL externa para o domínio de bancos de dados, permitindo que usuários definam e manipulem dados usando uma linguagem específica para esse propósito.

VANTAGENS	DESCRIÇÃO

ALINHAMENTO COM O CONHECIMENTO DO DOMÍNIO	A DSL alinha o código com o conhecimento e a terminologia dos especialistas do domínio, facilitando a validação e verificação de que o software implementa as regras de negócio corretamente.
REDUÇÃO DE AMBIGUIDADE	Como a DSL é criada usando a linguagem do domínio, ela reduz a ambiguidade que pode surgir quando desenvolvedores tentam traduzir requisitos de negócio para código utilizando uma linguagem de programação genérica.
DOCUMENTAÇÃO IMPLÍCITA	A DSL serve como uma forma de documentação que é sempre atualizada, já que está embutida no próprio código. Isso facilita a manutenção do software, pois a lógica de negócios é claramente expressa na linguagem do domínio.
FOCO NAS REGRAS DE NEGÓCIO	A DSL permite que o código se concentre nas regras de negócios em vez de detalhes técnicos, resultando em uma implementação que é mais fácil de compreender, modificar e expandir.

Em suma, uma Domain Specific Language (DSL) é uma ferramenta poderosa que ajuda a garantir que o software reflete com precisão o conhecimento do domínio, melhora a comunicação entre desenvolvedores e especialistas do domínio, e facilita a implementação e a manutenção das regras de negócios. Ao capturar a essência do domínio em uma linguagem que todos possam entender, a DSL se torna um componente central na criação de sistemas orientados ao domínio.

(TRE-PE – 2017) O DDD (domain-driven design):

a) consiste em uma técnica que trata os elementos de domínio e que garante segurança à aplicação em uma programação orientada a objetos na medida em que esconde as propriedades desses objetos.

b) não tem como foco principal a tecnologia, mas o entendimento das regras de negócio e de como elas devem estar refletidas no código e no modelo de domínio.

c) prioriza a simplicidade do código, sendo descartados quaisquer usos de linguagem ubíqua que fujam ao domínio da solução.

d) constitui-se de vários tratadores e (ou) programas que processam os eventos para produzir respostas e de um disparador que invoca os pequenos tratadores.

e) define-se como uma interface de domínio normalmente especificada e um conjunto de operações que permite acesso a uma funcionalidade da aplicação.

Comentários: (a) Errado. O DDD não se resume a esconder propriedades de objetos para garantir segurança, mas sim em modelar o domínio de forma que reflita as regras de negócio – esse conceito parece estar mais próximo à encapsulamento; (b) Correto. O foco do DDD é compreender e refletir as regras de negócio no código e no modelo de domínio, colocando a tecnologia em segundo plano; (c) Errado. A simplicidade é importante, mas o uso da linguagem ubíqua deve sempre ser relevante e significativo, não necessariamente simples ou restrito; (d) Errado. O DDD não se define por tratadores de eventos e disparadores, mas pela modelagem centrada no domínio – isso parece mais relacionado ao paradigma orientado a eventos; (e) Errado. Essa descrição se aproxima mais de uma definição de serviço ou API, não do conceito central de DDD – isso parece mais relacionado ao paradigma orientado a objetos (Letra E).

Model Driven

Model Driven Architecture (MDA)

INCIDÊNCIA EM PROVA: BAIXA

O MDA (do português, Arquitetura orientada a Modelo) é um método de desenvolvimento de software baseada em modelos. Em outras palavras, é um conjunto de práticas que guiam a estruturação de especificações expressas como modelos. Galera, saca só... *a questão falou em MDA?* **De cara, tem que lembrar que a modelagem é o foco do processo de desenvolvimento**. Refina-se um modelo abstrato até chegar a um modelo concreto. *Como assim?*

O modelo mais concreto da representação de um sistema é o código fonte. **Portanto, inicia-se com um modelo abstrato até chegar ao código fonte da maneira mais automatizada possível, entenderam?** Galera, o modelo pode ser definido em três etapas. A primeira etapa é a construção de um modelo com alto nível de abstração independente de tecnologia, chamado PIM – Platform-Independent Model.

A segunda etapa, considerada mais complexa, é o refinamento desse modelo em algo mais específico, chamado PSM – Platform Specific Model. Por fim, transforma-se esse PSM em código fonte. *Sacaram?* Então vamos resumir: MDA é focado em modelos! **Começamos com um modelo mais abstrato e terminamos em um modelo mais concreto.** *Como?* Partimos de um modelo de plataforma independente.

Em seguida, refinamos para um modelo de plataforma específica e terminamos com a produção do código fonte – de preferência, de modo automático. *Galera, se uma questão perguntasse onde está o maior esforço de desenvolvimento?* **Está na fase de modelagem – é lá que se encontra o foco!** Como a transformação para código fonte ocorre de forma automática, o tempo gasto para programar de fato é bastante reduzido. *Bacana?* É isso que vocês devem saber sobre MDA!

Model Driven Design (MDD)

INCIDÊNCIA EM PROVA: BAIXA

Desde as primeiras linguagens de programação, pesquisadores têm trabalhado para aumentar o grau de abstração em que se escreve programas. O primeiro compilador Fortran foi um grande marco na ciência da computação porque, pela primeira vez, permitiu aos programadores especificar o que a máquina deveria fazer e, não, como a máquina deveria fazer. Desde então, engenheiros têm tentado aumentar o nível de abstração da programação de software.

O MDD é a tentativa de continuar com esse intuito! Em vez de fazer os desenvolvedores especificarem todos os detalhes da implementação do sistema, ele permite que os desenvolvedores apenas modelem a funcionalidade e a arquitetura geral que o sistema deve ter.

Ele é uma abordagem de engenharia de software que utiliza um modelo para criar um produto de software. Ele faz uma separação clara entre modelo e código!

O usuário trabalha em um modelo independente de plataformas, seleciona uma plataforma específica e a ferramenta gera o código. Galera, isso aumenta bastante a produtividade, visto que o código é gerado automaticamente. Claro, ainda é necessário realizar testes, mas já é um grande avanço. **O intuito é desenvolver sistemas de modo bastante abstrato, fácil de escrever e rápido de produzir** – sem preocupação com detalhes de persistência, interoperabilidade ou distribuição.

(MPOG – 2013) De acordo com o OMG (Object Management Group), na MDA (Model-Driven Architecture), as especificações e funcionalidades do software devem ser modeladas por meio de um modelo independente de plataforma.

_____

Comentários: as especificações e funcionalidades devem ser modeladas por meio de um modelo independente de plataforma e só depois serão refinadas (Correto).

(TRT15 – 2009) A abordagem de arquitetura dirigida a modelos (MDA Model Driven Architecture) propõe que os sistemas devem ser projetados explicitamente com a visão focada em dois modelos:

a) visão de caso de uso e projeto de caso de teste.
b) independente de plataforma e plataforma específica.
c) plataforma de negócio e plataforma de classe.
d) visão de objetos empresa e visão de caso de uso.
e) orientado a eventos e orientado a objetos.

_____

Comentários: trata-se de um modelo independente de plataforma inicialmente e, depois, é gerado um modelo específico para uma plataforma (Letra B).

FEATURE DRIVEN DEVELOPMENT (FDD)

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Vamos falar agora sobre o Feature Driven Development (FDD). Também conhecido como Desenvolvimento orientado a Funcionalidades/Características, é uma das seis implementações de metodologias ágeis originais preconizadas pelo Manifesto Ágil. Criada como um projeto de um banco de Singapura, ela oferece um conjunto coeso de princípios e práticas tanto para gestão de projetos quanto para engenharia de software, e harmoniza bem com outras abordagens.

Essa metodologia se fundamenta em técnicas de gerenciamento de projetos e de modelagem orientada a objetos, equilibrando vantagens de metodologias tradicionais (Ex: planejamento e modelagem) e de metodologias ágeis (Ex: ciclos curtos, orientação ao cliente e ênfase em programação). Pessoal, eu diria que essa metodologia é um meio-termo entre XP e RUP. *Mas professor, o que seria uma Feature?*

Uma *feature* é basicamente uma funcionalidade ou característica valorizada pelo cliente, que pode ser implementada em duas semanas ou menos. Alguns dizem ser similar a um requisito funcional que gera valor ao cliente. À medida que há participação ativa do cliente no projeto, os resultados têm bastante visibilidade e de forma muito rápida. O método oferece algumas características importantes:

CARACTERÍSTICAS DO FDD
- Fornece uma estrutura adequada para equipes maiores;
- Enfatiza a produção de software de qualidade;
- Fornece informação de estado e progresso de forma simples;
- Agradam clientes, gerentes e desenvolvedores;
- Entrega resultados frequentes, tangíveis e funcionais;
- Planejamento detalhado e guiado para medição;
- Rastreabilidade e relatórios com precisão.

Essa metodologia recomenda também a adoção de um conjunto de melhores práticas para que o método atinja seus objetivos principais, são eles: modelagem de objetos de domínio; desenvolvimento por feature; posse individual do código; equipes de features; inspeções; builds regulares; gerenciamento de configuração; relatório e visibilidade de resultados. Possui duas grandes fases e cinco processos. Vejamos:

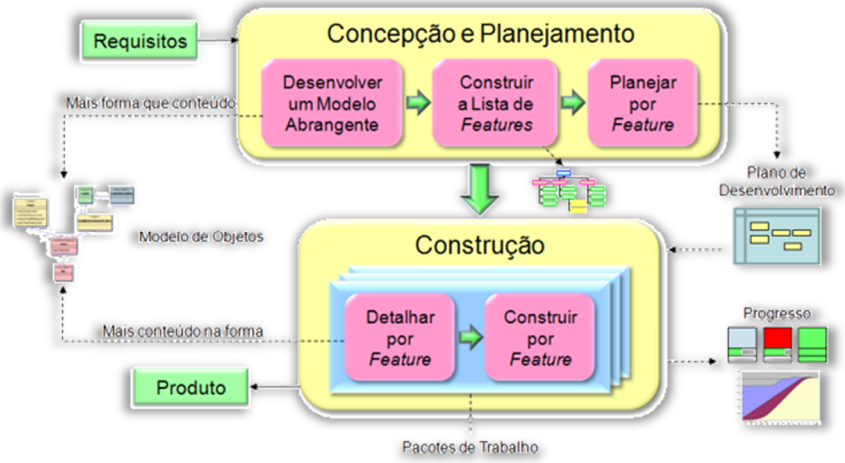
Fase 1 – Concepção & Planejamento: é uma parte crítica do processo, pois é nessa fase que são listadas as características (Features) que serão desenvolvidas, e em um primeiro momento é nessa fase que são definidas todas as características e fases do sistema e projetos, respectivamente. Seus processos são:

PROCESSO	DESCRIÇÃO
DESENVOLVER UM MODELO ABRANGENTE	Abrangendo todo o projeto, realiza-se um estudo dirigido sobre o escopo do sistema e seu contexto. Então, são realizados estudos mais detalhados para cada área a ser modelada. Assim, pequenos grupos são formados por membros do domínio do negócio e por desenvolvedores, que comporão seus próprios modelos. Os pequenos grupos apresentam seus modelos para serem revisados por parceiros e para discussão. Um dos modelos propostos é selecionado por consenso, tornando-se, assim, o modelo para aquela área do domínio do negócio. Realiza-se, então, uma combinação do modelo da área do domínio dentro de um modelo abrangente.
CONSTRUIR UMA LISTA DE FUNCIONALIDADES	Abrangendo todo o projeto, identificam-se todas as funcionalidades que satisfaçam os requisitos. Uma equipe é formada para decompor funcionalmente o domínio em áreas de negócio, atividades de negócio dentro delas e passos dentro de cada atividade de negócio, formando uma lista categorizada de funcionalidades.
PLANEJAR POR FUNCIONALIDADE	Abrangendo todo o projeto, busca-se produzir o plano de desenvolvimento. O gerente de projeto, o gerente de desenvolvimento e os programadores-líderes planejam a ordem de implementação, baseado nas dependências entre elas, na carga de trabalho da equipe e na complexidade das funcionalidades a serem implementadas. As principais atividades neste processo não são uma sequência estrita. Como muitas atividades de planejamento, elas são consideradas em conjunto, com refinamentos feitos a partir de uma ou mais atividades e então considerando os outros novamente. Após isso, a posse das classes estará completada (além das classes principais que já foram consideradas para posse).

Fase 2 – Construção: a implementação inicia agrupando *features* relacionadas e agrupando dentro de um pacote de trabalho, que deve ser completada dentro de uma iteração. Um pacote de trabalho completo representa uma parte do sistema que já pode ser utilizada pelo cliente.

PROCESSO	DESCRIÇÃO
DETALHAR POR FUNCIONALIDADE	Abrangendo cada funcionalidade, busca-se produzir o pacote de projeto para ela. Certo número de funcionalidades são agendadas para desenvolvimento ao atribuí-las a um programador-líder. Ele seleciona as funcionalidades para desenvolvimento a partir de sua caixa de entrada de funcionalidades atribuídas. Ele pode escolher diversas funcionalidades que utilizem as mesmas classes (e portanto, desenvolvedores). Operacionalmente, com frequência acontece o caso de conjuntos de funcionalidades serem agendados para desenvolvimento de uma vez pelo programador-líder. Tal conjunto é chamado de Pacote de Trabalho do Programador-Líder (PTPL). O programador-líder forma uma equipe de funcionalidades, identificando os proprietários das classes que provavelmente serão envolvidos no desenvolvimento das funcionalidades que ele selecionou. Esta equipe produz diagrama de sequência para as funcionalidades atribuídas. O programador-líder, então, refina o modelo de objetos e realiza-se uma inspeção no projeto.
CONSTRUIR FUNCIONALIDADE	Abrangendo cada funcionalidade, busca-se produzir uma função com valor para o cliente. Começando com o pacote de projeto, os proprietários de classes implementam os itens

necessários para que suas classes suportem o projeto para esta funcionalidade. O código passa por testes e inspeções. Após isso, é promovido à versão atual (build).



MARCOS
Walkthroughs do projeto
Projeto
Inspeção do projeto
Código
Inspeção de código
Progressão para construção

O Feature Driven Development pode facilitar imensuravelmente o fardo de reportar o status do projeto. Ele permite que o rastreio do progresso do desenvolvimento possa ser feito através de marcos definidos por funcionalidade, o que facilita a visualização do projeto como um todo. **Os marcos começam a ser monitorados pelo gerente do projeto a partir da fase de construção**. Vamos ver agora alguns exercícios sobre o tema...

QUESTÕES COMENTADAS – DIVERSAS BANCAS

1. **(FGV / AL-TO – 2024)** Relacione as metodologias listadas a seguir às suas características específicas.

1. Scrum
2. Programação Extrema (XP)
3. Modelo V
4. Lean Software Development
5. Feature-Driven Development (FDD)

( ) Enfatiza a redução de desperdícios no processo de desenvolvimento de software.
( ) Caracteriza-se por sua abordagem iterativa e incremental focada no desenvolvimento de características específicas do software.
( ) Diferencia-se por sua estrutura em que o teste é planejado e executado de forma a refletir inversamente as etapas de desenvolvimento.
( ) Implementa práticas como programação em pares, desenvolvimento orientado a testes e integração contínua.
( ) Emprega sprints como unidades de tempo fixas para o desenvolvimento de incrementos do produto

Assinale a opção que indica a relação correta, na ordem apresentada:

a) 4 – 5 – 3 – 2 – 1.
b) 5 – 3 – 4 – 1 – 2.
c) 2 – 1 – 5 – 3 – 4.
d) 3 – 2 – 1 – 4 – 5.
e) 1 – 4 – 2 – 5 – 3.

Comentários:

- **(4)** Lean Software Development: enfatiza a redução de desperdícios no processo de desenvolvimento de software.
- **(5)** Feature-Driven Development (FDD): caracteriza-se por sua abordagem iterativa e incremental focada no desenvolvimento de características específicas do software.
- **(3)** Modelo V: diferencia-se por sua estrutura em que o teste é planejado e executado de forma a refletir inversamente as etapas de desenvolvimento
- **(2)** Programação Extrema (XP): Implementa práticas como programação em pares, desenvolvimento orientado a testes e integração contínua.
- **(1)** Scrum: emprega sprints como unidades de tempo fixas para o desenvolvimento de incrementos do produto.

Gabarito: Letra A

2. **(FCC / TRF4– 2010)** A Feature Driven Development (FDD) é uma metodologia ágil de desenvolvimento de software, sobre a qual é correto afirmar:

a) Não pode ser combinada a outras técnicas para a produção de sistemas.

b) Possui cinco processos: Desenvolver um Modelo Abrangente, Construir a Lista de Funcionalidades, Planejar por Funcionalidade, Detalhar por Funcionalidade e Implementar por Funcionalidade.

c) Divide os papéis em dois grupos: papéis chave e papéis de apoio. Dentro de cada categoria, os papéis são atribuídos a um único participante que assume a responsabilidade pelo papel.

d) Mantém seu foco apenas na fase de modelagem.

e) Mantém seu foco apenas na fase de implementação.

Comentários:

(a) Errado. *Como não?* O próprio modelo é a combinação de ferramentas de diferentes metodologias – elas interagem bem com outros modelos; (b) Correto. Esses são de fato os cinco processos; (c) Errado. Na verdade, há diversos cargos e responsabilidades entre as categorias principais (ou chave), de apoio e adicionais; (d) Errado. O foco é tanto na Modelagem quanto na Implementação; (e) Errado. O foco é tanto na Modelagem quanto na Implementação.

Gabarito: Letra B

3. **(FCC / TRF3 – 2014)** Os modelos ágeis de desenvolvimento de software têm menos ênfase nas definições de atividades e mais ênfase na pragmática e nos fatores humanos do desenvolvimento. Um destes modelos enfatiza o uso de orientação a objetos e possui apenas duas grandes fases: 1 – Concepção e Planejamento e 2 – Construção. A fase de Concepção e Planejamento possui três disciplinas (chamadas de processos): Desenvolver Modelo Abrangente, Construir Lista de Funcionalidades e Planejar por funcionalidade. Já a fase de Construção incorpora duas disciplinas (processos): Detalhar por Funcionalidade e Construir por Funcionalidade. O texto acima apresenta a metodologia ágil conhecida como:

a) XP.
b) SCRUM.
c) Crystal Clear.
d) ASD.
e) FDD.

Comentários:

Todo enunciado apresenta características do Feature Driven Development – FDD.

Gabarito: Letra B

---

4. **(FCC / TRF3 – 2014)** Os modelos de processos tradicionais surgiram em um cenário muito diferente do atual, baseado em mainframes e terminais remotos. Já os modelos de processos ágeis são adequados para situações atuais nas quais a mudança de requisitos é frequente. Dentre os modelos de processos ágeis mais comuns temos: Extreme Programming (XP), Scrum e Feature Driven Development (FDD). Algumas das práticas e características desses modelos de processo são descritas a seguir:

I. Programação em pares, ou seja, a implementação do código é feita em dupla.
II. Desenvolvimento dividido em ciclos iterativos de até 30 dias chamados de sprints.
III. Faz uso do teste de unidades como sua tática de testes primária.
IV. A atividade de levantamento de requisitos conduz à criação de um conjunto de histórias de usuários.
V. O ciclo de vida é baseado em três fases: pre-game phase, game-phase, post-game phase.
VI. Tem como único artefato de projeto os cartões CRC.
VII. Realiza reuniões diárias de acompanhamento de aproximadamente 15 minutos.
VIII. Define seis marcos durante o projeto e a implementação de uma funcionalidade: walkthroughs do projeto, projeto, inspeção do projeto, codificação, inspeção de código e progressão para construção.
IX. Os requisitos são descritos em um documento chamado backlog e são ordenados por prioridade.

A relação correta entre o modelo de processo ágil e a prática/característica é:

	XP	SCRUM	FDD
a)	II, V e VII	II, IV e VIII	VII e IX
b)	I, III, IV e VI	II, V, VII e IX	VIII
c)	II, VII e VIII	III, IV, VI e IX	I e V
d)	II, VII e VIII	I, III, IV e V	VI e IX
e)	I, III, IV e VI	II, VIII, VII e IX	V

Comentários:

(I) XP; (II) Scrum; (III) XP; (IV) XP; (V) Scrum; (VI) XP; (VII) Scrum; (VIII) FDD; (IX) Scrum. Vamos comentar apenas a que nos interessa: FDD define seis marcos durante o projeto e implementação de uma funcionalidade: walkthroughs (travessia) do projeto, projeto, inspeção do projeto, codificação, inspeção de código e progressão para construção.

Gabarito: Letra B

---

5. **(FCC / TRT23 – 2011)** FDD (Feature Driven Development) é uma metodologia muito objetiva, possuindo apenas duas fases:

a) Concepção & Planejamento e Construção.
b) Decomposição Funcional e Construção.
c) Análise dos Requisitos e Desenvolvimento.
d) Planejamento Incremental e Desenvolvimento por Funcionalidade.
e) Planejamento por Funcionalidade e Construção por Funcionalidade.

Comentários:

Trata-se da Concepção & Planejamento e Construção.

Gabarito: Letra A

---

6. **(FCC / TRE-AM – 2010)** São algumas das metodologias de desenvolvimento de software consideradas ágeis (Agile Software Process Models):

a) RUP, XP e DSDM.
b) Waterfall, RUP e FDD.
c) XP, FDD e RUP.
d) Scrum, XP e FDD.
e) Scrum, Waterfall e DSDM.

Comentários:

(a) Errado. RUP não é ágil; (b) Errado. Waterfall e RUP não são ágeis; (c) Errado. RUP não é ágil; (d) Correto. Todos eles são ágeis; (e) Errado. Waterfall não é ágil.

Gabarito: Letra D

---

7. **(FCC / TRT-PR – 2015)** O Feature Driven Development – FDD é uma metodologia ágil de desenvolvimento de software que:

a) divide o processo de desenvolvimento nas etapas de Planejamento, Construção, Implantação e Manutenção.

b) normalmente possui papéis definidos relacionados ao desenvolvimento, como Gerente de Projeto, Testador etc.

c) está associado a práticas de modelagem estruturada, não orientada a objetos do domínio.

d) usa a propriedade coletiva do código, ou seja, o código não tem um dono definido.

e) na fase de construção possui os processos de Detalhamento das Funcionalidades, Programação, Teste e Melhoria Continuada.

Comentários:

(a) Errado, divide em Concepção & Planejamento e Construção; (b) Correto, ele possui alguns papeis definidos; (c) Errado, é uma metodologia associada à orientação a objetos; (d) Errado, o código possui um dono bem definido; (e) Errado, os processos são: detalhar por funcionalidade e construir funcionalidade.

Gabarito: Letra D

---

8. **(QUADRIX / COBRA – 2014)** "É um método ágil que enfatiza o uso de orientação a objetos. Possui apenas duas fases: 1 - Concepção e Planejamento; e 2 - Construção. A fase de Concepção e Planejamento possui três disciplinas: 1 - Desenvolver Modelo Abrangente; 2 - Construir Lista de Funcionalidades; e 3 - Planejar por funcionalidade. A fase de Construção possui duas disciplinas: 1 - Detalhar por Funcionalidade; e Construir por Funcionalidade". O texto descreve o método:

a) Feature-Driven Development (FDD).
b) Daily Scrum.
c) Crystal Clear.
d) Extreme Programming (XP).
e) Dynamic Systems Development Method (DSDM).

Comentários:

São todas características do Método FDD.

Gabarito: Letra A

---

9. **(FCC / TRT-SE – 2016)** Um Técnico está trabalhando em um projeto de desenvolvimento de software usando um método ágil que divide o processo de desenvolvimento em duas grandes fases: 1a Concepção e Planejamento; 2a Construção. A fase de Construção utiliza apenas duas disciplinas (processos). O nome do método e das disciplinas são, respectivamente,

a) Método - Feature-Driven Development
Disciplinas - Detalhar por Funcionalidade e Construir por Funcionalidade

b) Método - Daily Scrum
Disciplinas - Criar Backlog do Produto e Desenvolver Sprint

c) Método - Dynamic Systems Development
Disciplinas - Desenvolver Modelo Abrangente e Construir Iterativamente

d) Método - Rapid Application Development
Disciplinas - Projetar por Funcionalidade e Desenvolver por Funcionalidade

e) Método - Test-Driven Development
Disciplinas - Implementar Caso de Teste e Desenvolver Caso de Teste.

Comentários:

O método é o Feature-Driven Development e os processos da fase de construção são: projetar por funcionalidade e desenvolver por funcionalidade.

Gabarito: Letra D

---

10. (FCC / TJ-BA – 2015 – Item III) FDD (Feature Driven Development) pressupõe a entrega de pequenas versões funcionais, isto é, blocos bem pequenos de funcionalidade valorizada pelo cliente, a cada duas semanas ou menos.

Comentários:

Ele realmente pressupõe entregas pequenas de blocos de funcionalidade.

Gabarito: Correto

---

11. (FGV / COBRA – 2015) As características listadas a seguir referem-se, preferencialmente, a qual modelo de desenvolvimento?

Resultados úteis a cada duas semanas ou menos;
Blocos pequenos de funcionalidade valorizada pelo cliente, chamados "Features";
Planejamento detalhado e guia para medição; # Rastreabilidade e relatórios com maior precisão;
Monitoramento detalhado, com resumos para clientes e gerentes, em termos de negócio;
Fornece uma forma de saber, dentro dos primeiros 10% de um projeto, se o plano e a estimativa são sólidos.

a) SCRUM.
b) XP.
c) FDD.
d) DAS.
e) DSDM.

Comentários:

Todas essas são características apresentadas em nossa tabela.

Gabarito: Letra C

---

12. (FCC / TCE-CE – 2015) É um método ágil que enfatiza o uso da orientação a objetos. Possui duas fases que são "Concepção e Planejamento" e "Construção". A fase de "Concepção e Planejamento" possui três processos: "Desenvolver Modelo Abrangente", "Construir Lista de Funcionalidade" e "Planejar por Funcionalidade". Já a fase de "Construção" incorpora os processos "Detalhar por Funcionalidade" e "Construir por Funcionalidade". Trata-se do método:

a) Dynamic Systems Development Method – DSDM.
b) eXtreme Programming – XP.
c) Feature-Driven Development – FDD.
d) Crystal Clear – CC.
e) Adaptive Software Development – ASD.

Comentários:

Todas as características apresentadas no enunciado nos remetem ao Feature-Driven Development – FDD.

Gabarito: Letra C

---

13. (FCC / TJ-AP – 2014) Um analista judiciário está participando de um debate sobre metodologias ágeis a serem utilizadas no Tribunal de Justiça do Amapá. Ele afirma corretamente que:

a) o XP é uma metodologia adequada para equipes grandes que desenvolvem software baseado em requisitos precisos e que se modificam raramente. Entre suas características estão reuniões diárias e feedback constante.

b) o FDD busca o desenvolvimento por funcionalidade. Pode atuar bem em conjunto com o Scrum, pois quando o Scrum atuar com foco no gerenciamento do projeto, o FDD pode atuar no processo de desenvolvimento.

c) a MDA é uma abordagem em que modelos abrangentes são criados depois que o código-fonte está escrito, de forma a validar os modelos ágeis que guiam os esforços de desenvolvimento

d) um ponto em comum entre o XP e o FDD é que ambos defendem o desenvolvedor como único responsável pelo módulo que este desenvolve. Além disso, recomendam entregas e contatos mensais com o cliente.

e) no MDD a implementação do código é feita em dupla, de forma a procurarem identificar erros sintáticos e semânticos, pensando em como melhorar o código que está sendo implementado.

Comentários:

(a) Errado. É adequado para equipes pequenas que desenvolvem software baseado em requisitos imprecisos e que se modificam constantemente; (b) Correto. Ele realmente busca o desenvolvimento por funcionalidade, podendo atuar com Scrum focado no processo de desenvolvimento; (c) Errado. Modelos são criados antes da escrita do código-fonte; (d) Errado. O XP preconiza a responsabilidade compartilhada do código; (e) Errado. Essa é uma característica do TDD.

Gabarito: Letra B

---

14. **(CESGRANRIO / PETROBRÁS – 2006)** Há um considerável debate sobre os benefícios e a aplicabilidade do desenvolvimento ágil de software em contraposição aos processos mais convencionais de engenharia de software. Relacione o modelo ágil de software com a sua respectiva característica.

Modelo
I - DAS II - DSDM III - FDD IV - XP

Característica

(P) Define um ciclo de vida que incorpora três fases: especulação, colaboração e aprendizado. Durante a fase de aprendizado, à medida que os membros de uma equipe começam a desenvolver os componentes que fazem parte de um ciclo adaptativo, a ênfase está tanto no aprendizado quanto no progresso em direção a um ciclo completo.

(Q) O conceito característica é uma função valorizada pelo cliente, que pode ser implementada em duas semanas ou menos. Este modelo define seis marcos de referência durante o projeto e implementação de uma característica: travessia do projeto, projeto, inspeção de projeto, código, inspeção de código, promoção para construção.

(R) Fornece um arcabouço para construir e manter sistemas que satisfazem às restrições de prazo apertadas por meio do uso de prototipagem incremental em ambiente controlado de projeto. Essa abordagem sugere uma filosofia que é emprestada de uma versão modificada do princípio de Pareto.

A relação correta é:

a) I - P, II - Q, III - R.

b) I - P, II - R, III - Q.
c) I - Q, III - R, IV - P.
d) II - P, III - R, IV - Q.
e) II - Q, III - P, IV - R.

Comentários:

(I) DAS - define um ciclo de vida que incorpora três fases: especulação, colaboração e aprendizado. Durante a fase de aprendizado, à medida que os membros de uma equipe começam a desenvolver os componentes que fazem parte de um ciclo adaptativo, a ênfase está tanto no aprendizado quanto no progresso em direção a um ciclo completo; (II) DSDM - fornece um arcabouço para construir e manter sistemas que satisfazem às restrições de prazo apertadas por meio do uso de prototipagem incremental em ambiente controlado de projeto. Essa abordagem sugere uma filosofia que é emprestada de uma versão modificada do princípio de Pareto; (III) FDD - o conceito característica é uma função valorizada pelo cliente, que pode ser implementada em duas semanas ou menos. Este modelo define seis marcos de referência durante o projeto e implementação de uma característica: travessia do projeto, projeto, inspeção de projeto, código, inspeção de código, promoção para construção.

Gabarito: Letra B

LISTA DE QUESTÕES – DIVERSAS BANCAS

1. **(FGV / AL-TO – 2024)** Relacione as metodologias listadas a seguir às suas características específicas.

1. Scrum
2. Programação Extrema (XP)
3. Modelo V
4. Lean Software Development
5. Feature-Driven Development (FDD)

( ) Enfatiza a redução de desperdícios no processo de desenvolvimento de software.
( ) Caracteriza-se por sua abordagem iterativa e incremental focada no desenvolvimento de características específicas do software.
( ) Diferencia-se por sua estrutura em que o teste é planejado e executado de forma a refletir inversamente as etapas de desenvolvimento.
( ) Implementa práticas como programação em pares, desenvolvimento orientado a testes e integração contínua.
( ) Emprega sprints como unidades de tempo fixas para o desenvolvimento de incrementos do produto

Assinale a opção que indica a relação correta, na ordem apresentada:

a) 4 – 5 – 3 – 2 – 1.
b) 5 – 3 – 4 – 1 – 2.
c) 2 – 1 – 5 – 3 – 4.
d) 3 – 2 – 1 – 4 – 5.
e) 1 – 4 – 2 – 5 – 3.

2. **(FCC / TRF4– 2010)** A Feature Driven Development (FDD) é uma metodologia ágil de desenvolvimento de software, sobre a qual é correto afirmar:

a) Não pode ser combinada a outras técnicas para a produção de sistemas.

b) Possui cinco processos: Desenvolver um Modelo Abrangente, Construir a Lista de Funcionalidades, Planejar por Funcionalidade, Detalhar por Funcionalidade e Implementar por Funcionalidade.

c) Divide os papéis em dois grupos: papéis chave e papéis de apoio. Dentro de cada categoria, os papéis são atribuídos a um único participante que assume a responsabilidade pelo papel.

d) Mantém seu foco apenas na fase de modelagem.

e) Mantém seu foco apenas na fase de implementação.

3. **(FCC / TRF3 – 2014)** Os modelos ágeis de desenvolvimento de software têm menos ênfase nas definições de atividades e mais ênfase na pragmática e nos fatores humanos do desenvolvimento. Um destes modelos enfatiza o uso de orientação a objetos e possui apenas duas grandes fases: 1 – Concepção e Planejamento e 2 – Construção. A fase de Concepção e Planejamento possui três disciplinas (chamadas de processos): Desenvolver Modelo Abrangente, Construir Lista de Funcionalidades e Planejar por funcionalidade. Já a fase de Construção incorpora duas disciplinas (processos): Detalhar por Funcionalidade e Construir por Funcionalidade. O texto acima apresenta a metodologia ágil conhecida como:

a) XP.
b) SCRUM.
c) Crystal Clear.
d) ASD.
e) FDD.

4. **(FCC / TRF3 – 2014)** Os modelos de processos tradicionais surgiram em um cenário muito diferente do atual, baseado em mainframes e terminais remotos. Já os modelos de processos ágeis são adequados para situações atuais nas quais a mudança de requisitos é frequente. Dentre os modelos de processos ágeis mais comuns temos: Extreme Programming (XP), Scrum e Feature Driven Development (FDD). Algumas das práticas e características desses modelos de processo são descritas a seguir:

I. Programação em pares, ou seja, a implementação do código é feita em dupla.
II. Desenvolvimento dividido em ciclos iterativos de até 30 dias chamados de sprints.
III. Faz uso do teste de unidades como sua tática de testes primária.
IV. A atividade de levantamento de requisitos conduz à criação de um conjunto de histórias de usuários.
V. O ciclo de vida é baseado em três fases: pre-game phase, game-phase, post-game phase.
VI. Tem como único artefato de projeto os cartões CRC.
VII. Realiza reuniões diárias de acompanhamento de aproximadamente 15 minutos.
VIII. Define seis marcos durante o projeto e a implementação de uma funcionalidade: walkthroughs do projeto, projeto, inspeção do projeto, codificação, inspeção de código e progressão para construção.
IX. Os requisitos são descritos em um documento chamado backlog e são ordenados por prioridade.

A relação correta entre o modelo de processo ágil e a prática/característica é:

	XP	SCRUM	FDD
a)	II, V e VII	II, IV e VIII	VII e IX
b)	I, III, IV e VI	II, V, VII e IX	VIII
c)	II, VII e VIII	III, IV, VI e IX	I e V
d)	II, VII e VIII	I, III, IV e V	VI e IX
e)	I, III, IV e VI	II, VIII, VII e IX	V

5. **(FCC / TRT23 – 2011)** FDD (Feature Driven Development) é uma metodologia muito objetiva, possuindo apenas duas fases:

a) Concepção & Planejamento e Construção.
b) Decomposição Funcional e Construção.
c) Análise dos Requisitos e Desenvolvimento.
d) Planejamento Incremental e Desenvolvimento por Funcionalidade.
e) Planejamento por Funcionalidade e Construção por Funcionalidade.

6. **(FCC / TRE-AM – 2010)** São algumas das metodologias de desenvolvimento de software consideradas ágeis (Agile Software Process Models):

a) RUP, XP e DSDM.
b) Waterfall, RUP e FDD.
c) XP, FDD e RUP.
d) Scrum, XP e FDD.
e) Scrum, Waterfall e DSDM.

7. **(FCC / TRT-PR – 2015)** O Feature Driven Development – FDD é uma metodologia ágil de desenvolvimento de software que:

a) divide o processo de desenvolvimento nas etapas de Planejamento, Construção, Implantação e Manutenção.

b) normalmente possui papéis definidos relacionados ao desenvolvimento, como Gerente de Projeto, Testador etc.

c) está associado a práticas de modelagem estruturada, não orientada a objetos do domínio.

d) usa a propriedade coletiva do código, ou seja, o código não tem um dono definido.

e) na fase de construção possui os processos de Detalhamento das Funcionalidades, Programação, Teste e Melhoria Continuada.

8. **(QUADRIX / COBRA – 2014)** "É um método ágil que enfatiza o uso de orientação a objetos. Possui apenas duas fases: 1 - Concepção e Planejamento; e 2 - Construção. A fase de Concepção e Planejamento possui três disciplinas: 1 - Desenvolver Modelo Abrangente; 2 - Construir Lista

de Funcionalidades; e 3 - Planejar por funcionalidade. A fase de Construção possui duas disciplinas: 1 - Detalhar por Funcionalidade; e Construir por Funcionalidade". O texto descreve o método:

a) Feature-Driven Development (FDD).
b) Daily Scrum.
c) Crystal Clear.
d) Extreme Programming (XP).
e) Dynamic Systems Development Method (DSDM).

9. **(FCC / TRT-SE – 2016)** Um Técnico está trabalhando em um projeto de desenvolvimento de software usando um método ágil que divide o processo de desenvolvimento em duas grandes fases: 1a Concepção e Planejamento; 2a Construção. A fase de Construção utiliza apenas duas disciplinas (processos). O nome do método e das disciplinas são, respectivamente,

a) Método - Feature-Driven Development
Disciplinas - Detalhar por Funcionalidade e Construir por Funcionalidade

b) Método - Daily Scrum
Disciplinas - Criar Backlog do Produto e Desenvolver Sprint

c) Método - Dynamic Systems Development
Disciplinas - Desenvolver Modelo Abrangente e Construir Iterativamente

d) Método - Rapid Application Development
Disciplinas - Projetar por Funcionalidade e Desenvolver por Funcionalidade

e) Método - Test-Driven Development
Disciplinas - Implementar Caso de Teste e Desenvolver Caso de Teste.

10. **(FCC / TJ-BA – 2015 – Item III)** FDD (Feature Driven Development) pressupõe a entrega de pequenas versões funcionais, isto é, blocos bem pequenos de funcionalidade valorizada pelo cliente, a cada duas semanas ou menos.

11. **(FGV / COBRA – 2015)** As características listadas a seguir referem-se, preferencialmente, a qual modelo de desenvolvimento?

Resultados úteis a cada duas semanas ou menos;
Blocos pequenos de funcionalidade valorizada pelo cliente, chamados "Features";
Planejamento detalhado e guia para medição; # Rastreabilidade e relatórios com maior precisão;
Monitoramento detalhado, com resumos para clientes e gerentes, em termos de negócio;
Fornece uma forma de saber, dentro dos primeiros 10% de um projeto, se o plano e a estimativa são sólidos.

a) SCRUM.
b) XP.
c) FDD.
d) DAS.
e) DSDM.

12. **(FCC / TCE-CE – 2015)** É um método ágil que enfatiza o uso da orientação a objetos. Possui duas fases que são "Concepção e Planejamento" e "Construção". A fase de "Concepção e Planejamento" possui três processos: "Desenvolver Modelo Abrangente", "Construir Lista de Funcionalidade" e "Planejar por Funcionalidade". Já a fase de "Construção" incorpora os processos "Detalhar por Funcionalidade" e "Construir por Funcionalidade". Trata-se do método:

a) Dynamic Systems Development Method – DSDM.
b) eXtreme Programming – XP.
c) Feature-Driven Development – FDD.
d) Crystal Clear – CC.
e) Adaptive Software Development – ASD.

13. **(FCC / TJ-AP – 2014)** Um analista judiciário está participando de um debate sobre metodologias ágeis a serem utilizadas no Tribunal de Justiça do Amapá. Ele afirma corretamente que:

a) o XP é uma metodologia adequada para equipes grandes que desenvolvem software baseado em requisitos precisos e que se modificam raramente. Entre suas características estão reuniões diárias e feedback constante.

b) o FDD busca o desenvolvimento por funcionalidade. Pode atuar bem em conjunto com o Scrum, pois quando o Scrum atuar com foco no gerenciamento do projeto, o FDD pode atuar no processo de desenvolvimento.

c) a MDA é uma abordagem em que modelos abrangentes são criados depois que o código-fonte está escrito, de forma a validar os modelos ágeis que guiam os esforços de desenvolvimento

d) um ponto em comum entre o XP e o FDD é que ambos defendem o desenvolvedor como único responsável pelo módulo que este desenvolve. Além disso, recomendam entregas e contatos mensais com o cliente.

e) no MDD a implementação do código é feita em dupla, de forma a procurarem identificar erros sintáticos e semânticos, pensando em como melhorar o código que está sendo implementado.

14. **(CESGRANRIO / PETROBRÁS – 2006)** Há um considerável debate sobre os benefícios e a aplicabilidade do desenvolvimento ágil de software em contraposição aos processos mais convencionais de engenharia de software. Relacione o modelo ágil de software com a sua respectiva característica.

Modelo
I - DAS II - DSDM III - FDD IV - XP

Característica

(P) Define um ciclo de vida que incorpora três fases: especulação, colaboração e aprendizado. Durante a fase de aprendizado, à medida que os membros de uma equipe começam a desenvolver os componentes que fazem parte de um ciclo adaptativo, a ênfase está tanto no aprendizado quanto no progresso em direção a um ciclo completo.

(Q) O conceito característica é uma função valorizada pelo cliente, que pode ser implementada em duas semanas ou menos. Este modelo define seis marcos de referência durante o projeto e implementação de uma característica: travessia do projeto, projeto, inspeção de projeto, código, inspeção de código, promoção para construção.

(R) Fornece um arcabouço para construir e manter sistemas que satisfazem às restrições de prazo apertadas por meio do uso de prototipagem incremental em ambiente controlado de projeto. Essa abordagem sugere uma filosofia que é emprestada de uma versão modificada do princípio de Pareto.

A relação correta é:

a) I - P, II - Q, III - R.
b) I - P, II - R, III - Q.
c) I - Q, III - R, IV - P.
d) II - P, III - R, IV - Q.
e) II - Q, III - P, IV - R.

GABARITO – DIVERSAS BANCAS

1. LETRA A
2. LETRA B
3. LETRA B
4. LETRA B
5. LETRA A
6. LETRA D
7. LETRA D
8. LETRA A
9. LETRA D
10. CORRETO
11. LETRA C
12. LETRA C
13. LETRA B
14. LETRA B

DYNAMIC SYSTEMS DEVELOPMENT METHOD (DSDM)

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

O Método de Desenvolvimento de Sistemas Dinâmicos (DSDM) é uma abordagem de desenvolvimento de software ágil que oferece uma metodologia para construir e manter sistemas que atendem restrições de prazo apertado através do uso da prototipagem incremental em um ambiente controlado. **A filosofia DSDM baseia-se em uma versão modificada do Princípio de Pareto**. *Como assim, professor?*

80% de uma aplicação pode ser entregue em 20% do tempo para entregar a aplicação completa (100%). **O DSDM é um processo de software iterativo em que cada iteração segue a regra dos 80%.** Ou seja, somente o trabalho suficiente é requisitado para cada incremento, para facilitar o movimento para o próximo incremento. Funcionalidades atendem aos prazos fixos; e, não, o contrário. *Entendido?*

(COBRA – 2015) Em metodologias de desenvolvimento de sistemas ágeis, existe uma em especial que tem seus esforços focados na conclusão parcial da solução completa do sistema, mesmo que isso pareça contraditório para uma metodologia que pretende resolver o problema de desenvolvimento de sistemas. O texto refere-se a:

a) SCRUM.
b) BSP (Business System Plan).
c) MER (Modelo Entidade Relacionamento).
d) Engenharia de Requisitos.
e) DSDM (Dynamic Systems Development Methodology).

Comentários: como a questão menciona uma metodologia focada na conclusão parcial da solução completa do sistema, estamos falando de DSDM (Letra E).

Os detalhes remanescentes podem ser completados depois, quando outros requisitos de negócio forem conhecidos ou alterações tiverem sido solicitadas e acomodadas. O consórcio DSDM é um grupo mundial de empresas que coletivamente assume o papel de mantenedor do método. Ele definiu um modelo de processos ágeis, chamado ciclo de vida DSDM. **Esse ciclo de vida define três ciclos iterativos diferentes, precedidos por duas atividades de ciclo de vida adicionais:**

ATIVIDADE	DESCRIÇÃO
ESTUDO DE VIABILIDADE	Estabelece os requisitos básicos de negócio e restrições associados à aplicação a ser construída e depois avalia se a aplicação é ou não um candidato viável para o processo de Método de Desenvolvimento de Sistemas Dinâmicos (DSDM).

ESTUDO DE NEGÓCIO	Estabelece os requisitos funcionais e de informação que permitirão à aplicação agregar valor de negócio; define também a arquitetura básica da aplicação e identifica os requisitos de facilidade de manutenção para a aplicação.
ITERAÇÃO DE MODELOS FUNCIONAIS	Produz um conjunto de protótipos incrementais que demonstram funcionalidade para o cliente[1]. Durante esse ciclo iterativo, o objetivo é juntar requisitos adicionais ao se obter feedback dos usuários, conforme testam o protótipo.
ITERAÇÃO DE PROJETO E DESENVOLVIMENTO	Revisita protótipos desenvolvidos durante a iteração de modelos funcionais para assegurar-se de que cada um tenha passado por um processo de engenharia para capacitá-los a oferecer, aos usuários finais, valor de negócio em termos operacionais. Em alguns casos, a Iteração de Modelos Funcionais e a Iteração de Projeto e Desenvolvimento ocorrem ao mesmo tempo.
IMPLEMENTAÇÃO	Aloca a última versão do incremento de software (um protótipo "operacionalizado") no ambiente operacional. Deve-se notar que: (1) o incremento pode estar 100% completo ou (2) alterações podem vir a ser solicitadas conforme o incremento seja alocado. Em qualquer dos casos, o trabalho de desenvolvimento do DSDM contínua, retornando-se à atividade de iteração do modelo funcional.

O DSDM pode ser combinado com o XP para fornecer uma abordagem combinatória que define um modelo de processos consistente (o ciclo de vida do DSDM) com as práticas básicas (XP) necessárias para construir incrementos de software. **Além disso, os conceitos de colaboração e de equipes auto-organizadas do ASD podem ser adaptados a um modelo de processos combinado.**

(UFF – 2015) As metodologias ágeis de desenvolvimento surgiram em meados de 1990, como reação aos chamados métodos pesados de desenvolvimento, que eram caracterizados por muita formalidade nas documentações e regulamentações. Muitos eram gerenciados pelo tradicional modelo em cascata. Em 2001, de fato, após uma reunião no estado de Utah, surgiu, definitivamente, e foi propagado o paradigma de desenvolvimento de softwares ágeis. Muitos foram os motivos que levaram a essa concepção, por exemplo: gestão orientada a pessoas, adaptabilidade de processos, design e construção de software usando uma metodologia adaptativa, entre outros. Uma dessas metodologias ágeis é "centrada em estabelecer os recursos e o tempo fixo para o desenvolvimento de um projeto, ajustando suas funcionalidades de maneira a atender os prazos estipulados". A respeito dessa metodologia, assinale a alternativa correta.

a) SCRUM.
b) Extreme Programming (XP).

[1] Todos os protótipos DSDM são feitos com a intenção de que evoluam para a aplicação final entregue ao cliente.

c) Adaptive Software Development (ASD).
d) Dynamic Systems Development Methodology (DSDM).

_____

Comentários: o DSDM é um processo de software iterativo em que cada iteração segue a regra dos 80%. Ou seja, somente o trabalho suficiente é requisitado para cada incremento, para facilitar o movimento para o próximo incremento. Funcionalidades atendem aos prazos fixos; e, não, o contrário (Letra D).

(CEBRASPE / FINEP – 2024) Para entregar um projeto de software, foi designado um modelo adaptativo capaz de integrar fundamentos de projeto e análise de negócios, com foco na entrega frequente de produtos. Considerando a situação precedente, assinale a opção em que é apresentado o modelo que atende ao referido projeto:

a) DSDM (dynamic systems development method)
b) FDD (feature-driven development)
c) prototipagem
d) XP
e) lean software development

_____

Comentários: (a) Correto. O DSDM é um modelo ágil que enfatiza a colaboração e integração entre análise de negócios e desenvolvimento técnico, visando a entrega frequente de produtos funcionais;

(b) Errado. O FDD foca na modelagem e desenvolvimento orientado por características do software, mas não integra explicitamente a análise de negócios como parte de sua metodologia, o que é menos aderente ao cenário descrito;

(c) Errado. A prototipagem é útil para desenvolver versões iniciais do software para teste e feedback, mas não possui um framework estruturado que integre análise de negócios e não foca necessariamente na entrega frequente de produtos completos;

(d) Errado. XP é centrado no desenvolvimento iterativo e na qualidade do código, mas, apesar de sua eficácia, não enfoca explicitamente a integração entre análise de negócios e desenvolvimento como principal característica;

(e) Errado. Lean Software Development baseia-se em princípios de produção enxuta e eliminação de desperdícios, focando em eficiência e entrega de valor, porém não destaca a integração direta de análise de negócios como elemento central de sua abordagem (Letra A).

NOÇÕES DE DEVOPS

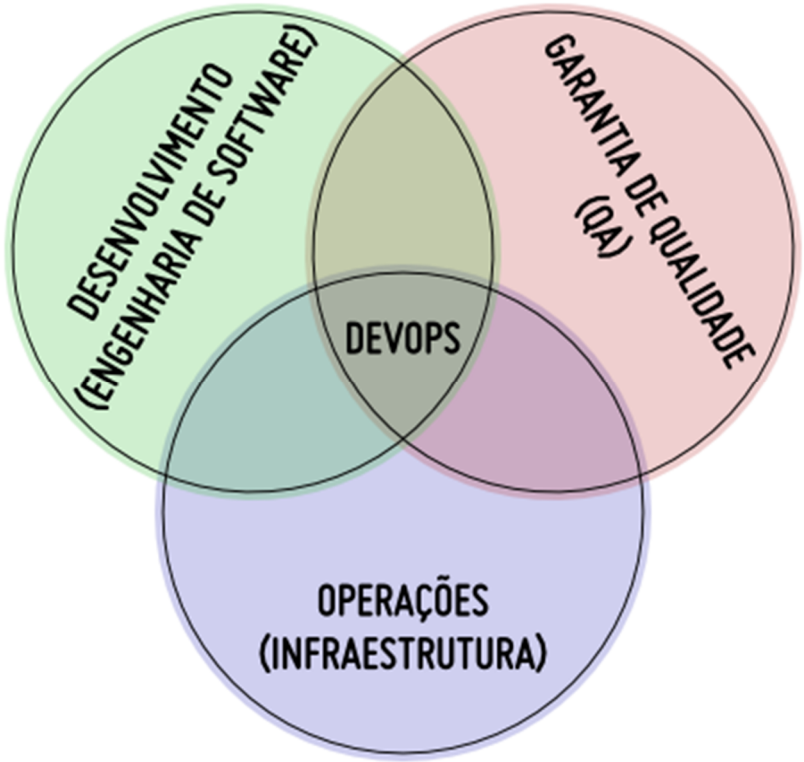
Conceitos Básicos

INCIDÊNCIA EM PROVA: MÉDIA

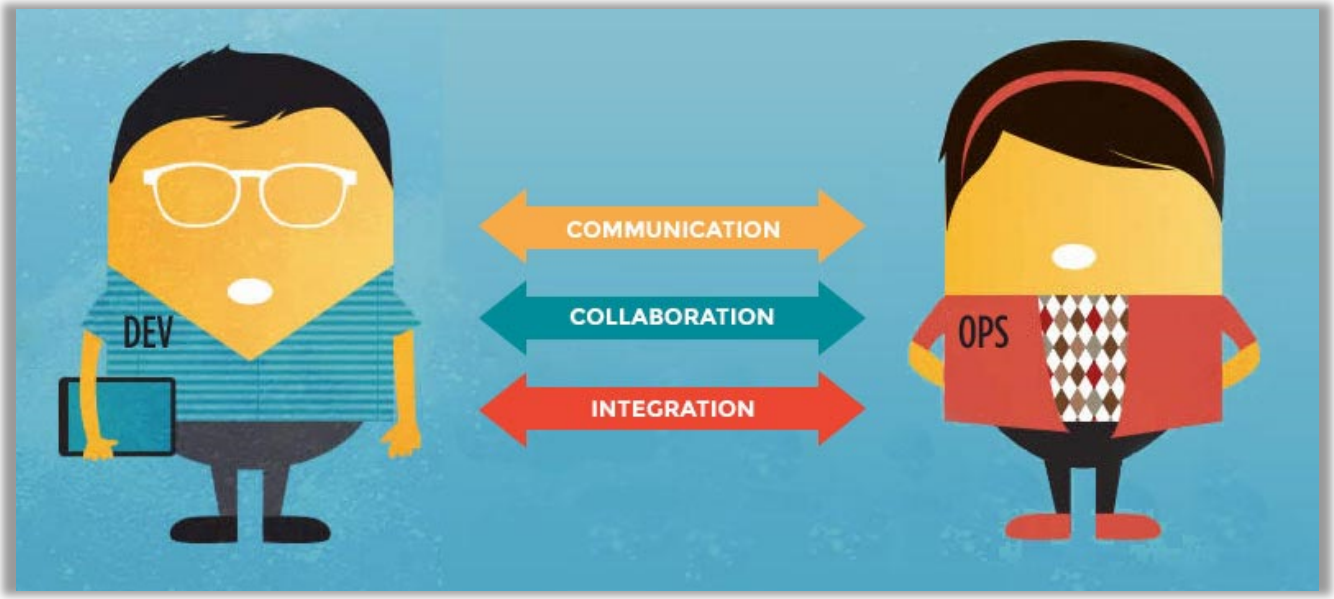
De um tempo para cá, nós temos vivenciado novos paradigmas em tecnologias de software. **As metodologias ágeis, por exemplo, vieram com um conjunto de práticas que desencadearam diversas outras práticas, de forma a obter software de maneira mais ágil e mais adaptável a possíveis mudanças**. No entanto, o grande lance é que as mudanças ocorrem com um intervalo de tempo cada vez menor.

Vocês já devem ter percebido isso! Antigamente, softwares lançavam atualizações em intervalos de 18 a 24 meses (ou até mais). **Atualmente, a dinâmica de consumo de aplicações de tecnologia da informação sofreu uma reviravolta e a demanda dos clientes é insana.** As empresas de software atualmente são duramente pressionadas para lançar e atualizar aplicativos no mercado o mais rápido possível.

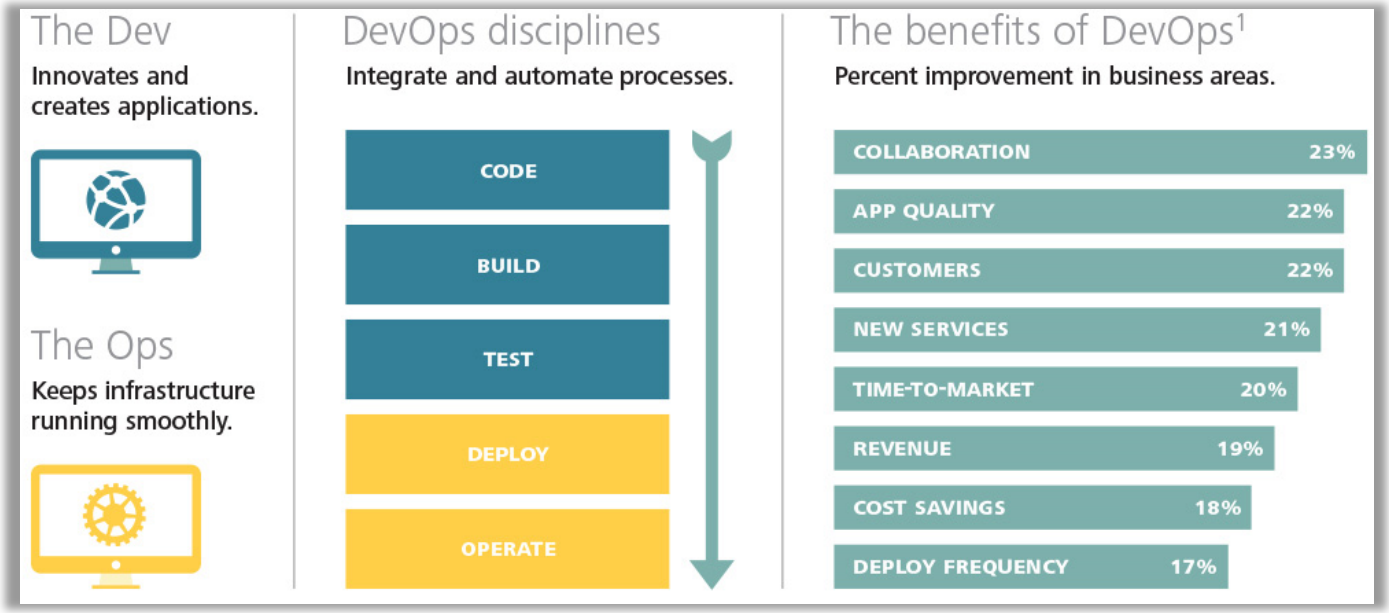
Pois é, o mundo mudou! O ciclo para a criação de novos aplicativos de software dura cerca de três meses para uma versão inicial e mais seis meses para o conjunto completo de recursos**. Não só o ciclo de vida foi encurtado, mas os aplicativos se tornaram muito mais complexos e exigem colaboração e integração cruzada entre os diversos componentes de tecnologia da informação, como Dev, Ops e Q&A.**



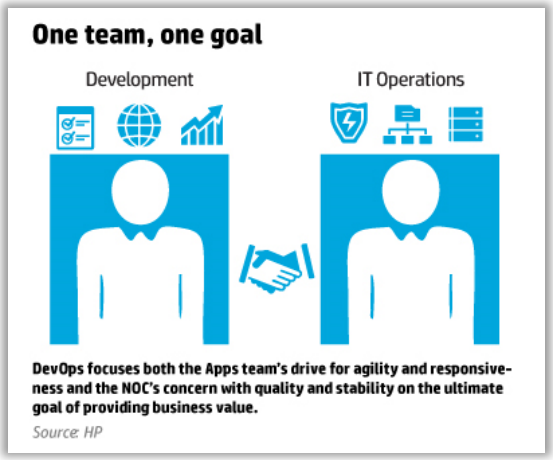
Professor, espera um pouco aí! O que acontece se eu juntar conceitos de Desenvolvimento de Software, Operação de Sistemas e Garantia de Qualidade? Surgirá, então, o conceito de DevOps! **Trata-se de um conceito muito simples, como apresenta a imagem acima. Essas ideias são tratadas em conjunto, em vez de separadas.** O lance é ter uma maior comunicação, colaboração e integração entre essas áreas.



Professor, o que é exatamente DevOps? É uma cultura? É uma técnica? É uma metodologia de desenvolvimento de software? Ainda não existe uma resposta precisa para essas perguntas! *Por que, professor?* **Porque foi um movimento que começou ao mesmo tempo em diversos lugares diferentes, tratando de infraestrutura e desenvolvimento, mas não houve um manifesto formal, como o manifesto ágil.**



Parece simples fazer a infraestrutura conversar de forma harmônica com o desenvolvimento, mas não é tão fácil! *Qual é o papel da infraestrutura?* É sustentar os sistemas em produção; monitorar o funcionamento e a performance; cuidar da estabilidade, segurança, níveis de serviço; planejar mudanças, minimizando riscos; entre outros. *O que acontece se uma aplicação em produção parar de funcionar?*



Isso pode significar prejuízo financeiro ou institucional. Em suma, podemos dizer que a infraestrutura se preocupa em proteger o valor de negócio. *E o desenvolvimento?* Esses caras se preocupam com inovação e criatividade, baseado nos requisitos do usuário. Desenvolvedor fica louco quando sai uma biblioteca, componente ou tecnologia nova. A consequência disso é que cada inovação significa um novo Deploy (feito pela rapaziada da infraestrutura).

E se ocorrer algum problema? Deve ser realizado um rollback (também pelo pessoal da infraestrutura). **Podemos afirmar, então, que o desenvolvedor se preocupa em aumentar o valor do negócio.** Vocês já devem ter notado que há um conflito interessante nessa conversa. Ora, o desenvolvedor quer colocar suas aplicações no ar o mais rápido possível para que fique disponível para o cliente.

No entanto, a galera da infraestrutura quer ter certeza de que a aplicação está suficientemente estável para ir para produção sem gerar incidentes que parem o que já está funcionando. *O que ocorria antigamente?* As empresas permitiam apenas um deploy por semana ou por mês! *Tem coisa mais não-ágil que isso?* Vai totalmente de encontro aos ideais do manifesto ágil. *Lembram-se dos conceitos de entrega, integração e teste contínuos?*

Pois é, a infraestrutura teve que se adaptar a realizar deploys diários. No entanto, os desenvolvedores – muitas vezes – se esqueciam de considerar algumas diferenças importantes entre ambientes de desenvolvimento e produção. **Isso gerava alguns incidentes, o cliente reclamava e começava uma briga muito comum representada pelas duas imagens anteriores.** Sim, galera... rola essa briga!

Desenvolvedores afirmando que a Infraestrutura é engessada, lenta e que não oferece um ambiente adequado para o desenvolvimento de aplicações; já a Infraestrutura afirmava que os desenvolvedores faziam código ruim e instável, e que a culpa não era deles. **Pessoal, voltem agora para a primeira imagem e percebam o que DevOps tenta integrar: Desenvolvimento, Infraestrutura e Qualidade!**

Parece briga de marido e mulher, mas ambos os lados têm que reconhecer seus erros, ceder e se adaptar! **O Desenvolvimento precisa pensar mais na Infraestrutura e controlar as fases de deploy (Ex: Deployment Pipeline). Já a Infraestrutura tem que evoluir para o mundo ágil.** Começar a trabalhar de forma automatizada e dinâmica, ser mais veloz para subir ambientes; reconstruir ou duplicar ambientes de acordo com as necessidades do Desenvolvimento.

Galera, as principais características do DevOps são: colaboração entre equipes; fim de divisões; relação saudável entre áreas; teste, integração e entrega contínuos; automação de deploy; controle e monitoração; gerenciamento de configuração; orquestração de serviços; avaliação de métricas e desempenho; logs e integração; velocidade de entrega; feedback intenso; e comunicação constante.

Em suma, podemos dizer que ele é um movimento, um conceito, uma cultura, uma abordagem que trata do feedback, comunicação e colaboração entre áreas de tecnologia da informação com o objetivo de garantir qualidade do software, com menor custo, mais rapidez, menor risco e maior eficiência. **É como se fosse a utilização de metodologias ágeis no desenvolvimento e na infraestrutura.** *Bacana?*

Para o DevOps, o código gerado pela infraestrutura é apenas mais um artefato qualquer de desenvolvimento e, não, uma parte separada. Trata-se de uma prática importante, já que não basta somente o desenvolvedor escrever o código do sistema, é necessário que a área de infraestrutura também atue para liberar, controlar e entregar a versão do sistema de forma contínua, periódica e preferencialmente automática.

TODAY'S DEVOPS

DEV + OPS

PART DEV

Application Performance

Modern developers use APM tools to decrease latency, have complete visibility into code, databases, caches, queues, and third-party services.

End User Analytics

A great developer understands end users have the best feedback and analytics play an enormous part of understanding users. Developers are constantly monitoring end user latency and checking performance by devices and browsers.

Quality Code

Developers need to ensure their deployments and new releases don't implode or degrade the overall performance.

Code-Level Errors

When you have a large distributed application it is vital to lower MTTR by finding the root cause of errors and exceptions.

PART OPS

Application Availability

The applications need to be up and running and it's Ops responsibility to ensure uptime and SLAs are in order.

Application Performance

Classic Ops generally rely on infrastructure metrics - CPU, memory, network and disk I/O, etc. Modern Ops correlate all of those metrics with application metrics to solve problems 10x faster.

End User Complaints

The goal is to know about and fix problems before end users complain, reduce the number of support tickets, and eliminate false alerts.

Performance Analytics

Automatically generated baselines of all metrics help Ops understand what has changed and where to focus their troubleshooting efforts. Alerts based upon deviation from observed baselines improve alert quality and reduce alert noise.

Vamos resumir: **a preocupação é com os objetivos quase diametralmente opostos da galera de Desenvolvimento (DEV) e Operações (OPS).** Os desenvolvedores querem programar novos recursos, melhorar o produto; corrigir bugs, etc. As operações querem colocar tudo em funcionamento e nunca mudar, já que as alterações causam novos erros, bugs, problemas de desempenho, entre outros.

O objetivo do DevOps é aliviar a tensão entre esses dois campos! **Ter pessoas de Operações nas trincheiras de Desenvolvimento é a principal maneira de alcançar esse objetivo.** Seu trabalho é facilitar ao máximo que desenvolvedores e operações façam o que precisam fazer. *Como assim?* Por exemplo: fornecendo ambientes idênticos de desenvolvimento, testes, homologação, *staging* e qualidade; configuração de pipelines de teste e implementação automáticos.

Além de estar envolvido no processo de desenvolvimento para que eles estejam mais preparados para lidar com erros de produção. **Tradicionalmente, as operações são quase alheias à base de código, uma vez que se preocupam apenas com sua infraestrutura.** O objetivo é tornar todo o processo transparente de forma que você possa fazer milhares de implantações de produção por dia; ao contrário dos métodos tradicionais de configuração de uma janela de implantação uma vez a cada trimestre ou por mais tempo.

Claro que para fazer isso, é necessário utilizar um conjunto de práticas e ferramentas. *Quais, professor?* Ferramentas de Controle de versão (Git, CVS, Tortoise), Servidores de Integração Contínua (Jenkins, Bamboo, Travis), Docker/Vagrant, Gerenciamento de Configuração (SaltStack, Chef, Puppet). Isso reduz a dor de cabeça tanto para os desenvolvedores quanto para os operadores e reduz a quantidade de problemas de desempenho e erros de codificação.

(STF – 2013) Integração contínua, entrega contínua, teste contínuo, monitoramento contínuo e feedback são algumas práticas do DevOps.

_____

Comentários: as principais características do DevOps são: colaboração entre equipes; fim de divisões; relação saudável entre áreas; teste, integração e entrega contínuos; automação de deploy; controle e monitoração; gerenciamento de configuração; orquestração de serviços; avaliação de métricas e desempenho; logs e integração; velocidade de entrega; feedback intenso; e comunicação constante (Correto).

(STF – 2013) Teste contínuo é uma prática do DevOps que, além de permitir a diminuição dos custos finais do teste, ajuda as equipes de desenvolvimento a balancear qualidade e velocidade.

_____

Comentários: teste, integração e entrega contínuos são realmente práticas do DevOps que permitem reduzir custos de teste e ajuda a equipe de desenvolvimento a balancear a qualidade e velocidade (Correto).

(TCU – 2015) De acordo com a abordagem DevOps (development – operations), os desafios da produção de software de qualidade devem ser vencidos com o envolvimento

dos desenvolvedores na operação dos sistemas com os quais colaboraram no desenvolvimento.

_____

Comentários: essa questão permite duas interpretações de 'envolvimento': (1) no sentido de interação e colaboração entre equipes; (2) no sentido de mão na massa mesmo - na operação dos sistemas. Por conta da ambiguidade, a questão foi anulada (Anulada).

(TRE-PE – 2017) O DevOps consiste em:

a) um processo similar ao IRUP (IBM Rational Unified Process), que tem como objetivo dividir o processamento em fases e disciplinas de software para paralelizar as ações de desenvolvimento e de manutenção das soluções.

b) uma plataforma aberta cuja função é substituir a virtualização de aplicações e serviços em containers e, com isso, agilizar a implantação de soluções de software.

c) um aplicativo que permite o gerenciamento de versões de códigos-fonte e versões de programas, bem como a implantação da versão mais recente de um software em caso de falha.

d) um processo de promoção de métodos que objetivam aprimorar a comunicação, tornando a colaboração eficaz especialmente entre os departamentos de desenvolvimento e teste e entre os departamentos de operações e serviço para o negócio.

e) uma metodologia ágil que, assim como a XP (extreme programming) e o Scrum, tem foco na gestão de produtos complexos relativos à equipe de desenvolvimento.

_____

Comentários: DevOps não é um processo similar ao iRUP, não é uma plataforma aberta, não é um aplicativo e não é uma metodologia ágil. DevOps é um processo de promoção de métodos que objetivam aprimorar a comunicação, tornando a colaboração eficaz especialmente entre os departamentos de desenvolvimento e teste e entre os departamentos de operações e serviço para o negócio (Letra D).

(TRT-PA e AP – 2016) Acerca de DevOps, assinale a opção correta.

a) O DevOps concentra-se em reunir diferentes processos e executá-los mais rapidamente e com mais frequência, o que gera baixa colaboração entre equipes.

b) O DevOps tem como princípio produzir, a partir da avaliação dos times de desenvolvimento do serviço, grandes mudanças e farta documentação com valor agregado para os usuários, assemelhando-se, por isso, com objetivos dos métodos iterativos e em cascata.

c) A infraestrutura de nuvem de provedores internos e externos vem restringindo o uso de DevOps pelas organizações.

d) O DevOps parte da premissa de adoção de grandes equipes de especialistas, com a menor interação possível, visando à padronização de processos e à mínima automação de atividades.

e) Atividades típicas em DevOps compreendem teste do código automatizado, automação de fluxos de trabalho e da infraestrutura e requerem ambientes de desenvolvimento e produção idênticos.

_____

Comentários: (a) Errado, isso gera alta colaboração entre equipes; (b) Errado, não se assemelha com objetivos de métodos em cascata, visto que essa metodologia possui entregas menos frequentes e é menos dinâmica; (c) Errado, é justamente o inverso – elas usam com cada vez mais frequência; (d) Errado, recomenda-se a maior interação possível entre as equipes; (e) Correto, testes automatizados, automação de fluxos e infraestrutura são atividades frequentes que realmente exigem ambientes de desenvolvimento e produção idênticos (Letra E).

(STJ – 2015) DevOps é um conceito pelo qual se busca entregar sistemas melhores, com menor custo, em menor tempo e com menor risco.

_____

Comentários: perfeito, perfeito, perfeito – todas são características do DevOps (Correto).

(STJ – 2015) O profissional especialista em DevOps deve atuar e conhecer as áreas de desenvolvimento (engenharia de software), operações e controle de qualidade, além de conhecer, também, de forma ampla, os processos de desenvolvimento ágil.

_____

Comentários: ele é a combinação entre desenvolvimento, operação e controle de qualidade – portanto questão perfeita (Correto).

(SLU-DF – 2019) Em DevOps, o princípio monitorar e validar a qualidade operacional antecipa o monitoramento das características funcionais e não funcionais dos sistemas para o início do seu ciclo de vida, quando as métricas de qualidade devem ser capturadas e analisadas.

_____

Comentários: o princípio monitorar e validar a qualidade operacional transfere o monitoramento para o início do ciclo de vida do software, para identificar antecipadamente problemas de qualidade que podem ocorrer no desenvolvimento. Na implantação e teste, as métricas de qualidade são capturadas e analisadas, sendo que essas métricas devem ser entendidas por todos os stakeholders (Correto).

(MPE-PI – 2018) A infraestrutura como código é uma prática DevOps caracterizada pela infraestrutura provisionada e gerenciada por meio de técnicas de desenvolvimento de código e de software, como, por exemplo, controle de versão e integração contínua.

_____

Comentários: trata-se realmente de uma prática em que a infraestrutura provisionada e gerenciada por meio de técnicas de desenvolvimento de código e de software – ela oferece controle de versão, entrega e integração contínua (Correto).

(STJ – 2018) Apesar de ser um processo com a finalidade de desenvolver, entregar e operar um software, o DevOps é incompatível com a aplicação de métodos ágeis como o Scrum ou, ainda, com o uso de ferramentas que permitam visualizar os fluxos do processo.

_____

Comentários: pelo contrário, é completamente compatível com as aplicações de métodos ágeis como o Scrum ou, ainda, com o uso de ferramentas que permitam visualizar os fluxos do processo (Errado).

(STJ – 2018) O gerenciamento de desenvolvimento de software por meio do Scrum pode ser combinado com o ciclo de vida do DevOps, haja vista que o DevOps combina práticas e ferramentas que aumentam a capacidade de uma organização de distribuir aplicativos e serviços; logo, a integração contínua do software pode ser realizada na sprint do Scrum junto com a operação dos serviços da organização.

_____

Comentários: ele realmente combina práticas e ferramentas que aumentam a capacidade de uma organização de distribuir aplicativos e serviços e pode ser realizado na sprint do Scrum por meio da integração contínua (Correto).

QUESTÕES COMENTADAS

1. **(CESPE / CNPq – 2024)** O conceito de DevOps envolve a automatização de processos e colaboração entre equipes.

Comentários:

O conceito de DevOps envolve a automatização de processos e a colaboração entre as equipes de desenvolvimento (Dev) e operações (Ops). O objetivo é melhorar a eficiência, aumentar a frequência das entregas e assegurar a qualidade e a confiabilidade dos sistemas.

Gabarito: Correto

---

2. **(CESPE / Pref. Cachoeira de Itapemirim-ES – 2024)** Uma das boas práticas do DevOps é a adoção de uma cultura livre de culpa por erros nos processos apresentados pelos desenvolvedores ou pelo pessoal de operações.

Comentários:

Uma das boas práticas do DevOps é fomentar uma cultura livre de culpa (blameless culture). Essa abordagem incentiva a comunicação aberta e a colaboração entre equipes, focando na resolução de problemas e na melhoria contínua em vez de atribuir culpas individuais. Isso ajuda a identificar causas-raiz e a implementar soluções que evitam recorrência de erros, promovendo um ambiente mais produtivo e inovador.

Gabarito: Correto

---

3. **(CESPE / MPE-GO – 2024)** No DevOps, o pipeline de entrega contínua gerencia o processo de desenvolvimento do software, prescindindo-se dos testes, uma vez que a gestão do código-fonte é iterativa com vistas a otimizar a entrega rápida de software de alta qualidade, unindo as equipes de desenvolvimento e de operações.

Comentários:

No DevOps, o pipeline de entrega contínua (continuous delivery pipeline) não prescinde dos testes. Pelo contrário, os testes são uma parte fundamental e integradora do pipeline, garantindo a qualidade e a estabilidade do software em cada etapa do processo de desenvolvimento. A gestão do código-fonte é iterativa, mas a presença contínua de testes automatizados e manuais assegura que o software entregue seja de alta qualidade e pronto para produção, facilitando a colaboração entre as equipes de desenvolvimento e operações.

Gabarito: Errado

---

4. **(CESPE / MPO – 2024)** Logs são registros do que ocorreu em um sistema específico que contém informações importantes que dão à equipe de DevOps uma visão holística sobre como o software está funcionando, fornecendo-lhes os materiais necessários para tomar decisões.

Comentários:

Logs são registros detalhados das atividades e eventos que ocorrem em um sistema. Eles fornecem informações cruciais sobre o funcionamento do software, desempenho e erros, permitindo que a equipe de DevOps tenha uma visão abrangente do sistema. Essas informações são essenciais para monitorar, diagnosticar problemas e tomar decisões informadas sobre a operação e manutenção do software.

Gabarito: Correto

---

5. **(CESPE / MPO – 2024)** Integração contínua é uma prática DevOps em que desenvolvedores compartilham o trabalho regularmente e automatizam a construção, o teste e a validação com a prática de fazer commits.

Comentários:

A integração contínua é uma prática essencial no DevOps que envolve a integração frequente do trabalho dos desenvolvedores no repositório principal, geralmente várias vezes ao dia. Cada commit é automaticamente testado e construído para identificar problemas rapidamente, garantindo que o software seja sempre funcional e de alta qualidade.

Gabarito: Correto

---

6. **(FGV / AL-TO – 2024)** Para integrar um departamento de desenvolvimento de software e um de operações de TI, assinale a ação correta.

a) Implementação da metodologia Agile, concentrando-se na melhoria contínua e colaboração entre as equipes de desenvolvimento de software e operações de TI.

b) Utilização da técnica de Benchmarking, comparando os processos internos com os de organizações líderes do setor para identificar melhores práticas.

c) Adoção do modelo ITIL para a gestão de serviços de TI, focando na padronização dos processos de entrega e suporte de serviços de TI.

d) Aplicação do framework DevOps, integrando os departamentos de desenvolvimento de software e o de operações de TI para agilizar a entrega de projetos e melhorar a comunicação e a colaboração.

e) Introdução da técnica Six Sigma, com o objetivo de reduzir a variabilidade nos processos de desenvolvimento de software e operações de TI, aumentando a qualidade das entregas.

Comentários:

(a) Errado. A metodologia Agile foca em melhoria contínua e colaboração, mas é especificamente voltada para equipes de desenvolvimento de software, não garantindo por si só a integração com operações de TI;

(b) Errado. Benchmarking é uma técnica para identificar melhores práticas, comparando processos internos com os de outras organizações. No entanto, não é diretamente voltada para a integração entre departamentos;

(c) Errado. O modelo ITIL é utilizado para gestão de serviços de TI, focando na padronização de processos de entrega e suporte, mas não se destina especificamente à integração entre desenvolvimento de software e operações de TI;

(d) Correto. DevOps é um framework especificamente criado para integrar os departamentos de desenvolvimento de software e operações de TI, promovendo a comunicação e colaboração contínua para agilizar a entrega de projetos;

(e) Errado. Six Sigma é uma técnica que busca reduzir a variabilidade nos processos para aumentar a qualidade das entregas, mas seu principal foco não é a integração entre diferentes departamentos, e sim a melhoria da qualidade e eficiência.

Gabarito: Letra D

QUESTÕES COMENTADAS

1. **(CESPE / CNPq – 2024)** O conceito de DevOps envolve a automatização de processos e colaboração entre equipes.

2. **(CESPE / Pref. Cachoeira de Itapemirim-ES – 2024)** Uma das boas práticas do DevOps é a adoção de uma cultura livre de culpa por erros nos processos apresentados pelos desenvolvedores ou pelo pessoal de operações.

3. **(CESPE / MPE-GO – 2024)** No DevOps, o pipeline de entrega contínua gerencia o processo de desenvolvimento do software, prescindindo-se dos testes, uma vez que a gestão do código-fonte é iterativa com vistas a otimizar a entrega rápida de software de alta qualidade, unindo as equipes de desenvolvimento e de operações.

4. **(CESPE / MPO – 2024)** Logs são registros do que ocorreu em um sistema específico que contém informações importantes que dão à equipe de DevOps uma visão holística sobre como o software está funcionando, fornecendo-lhes os materiais necessários para tomar decisões.

5. **(CESPE / MPO – 2024)** Integração contínua é uma prática DevOps em que desenvolvedores compartilham o trabalho regularmente e automatizam a construção, o teste e a validação com a prática de fazer commits.

6. **(FGV / AL-TO – 2024)** Para integrar um departamento de desenvolvimento de software e um de operações de TI, assinale a ação correta.

   a) Implementação da metodologia Agile, concentrando-se na melhoria contínua e colaboração entre as equipes de desenvolvimento de software e operações de TI.

   b) Utilização da técnica de Benchmarking, comparando os processos internos com os de organizações líderes do setor para identificar melhores práticas.

   c) Adoção do modelo ITIL para a gestão de serviços de TI, focando na padronização dos processos de entrega e suporte de serviços de TI.

   d) Aplicação do framework DevOps, integrando os departamentos de desenvolvimento de software e o de operações de TI para agilizar a entrega de projetos e melhorar a comunicação e a colaboração.

   e) Introdução da técnica Six Sigma, com o objetivo de reduzir a variabilidade nos processos de desenvolvimento de software e operações de TI, aumentando a qualidade das entregas.

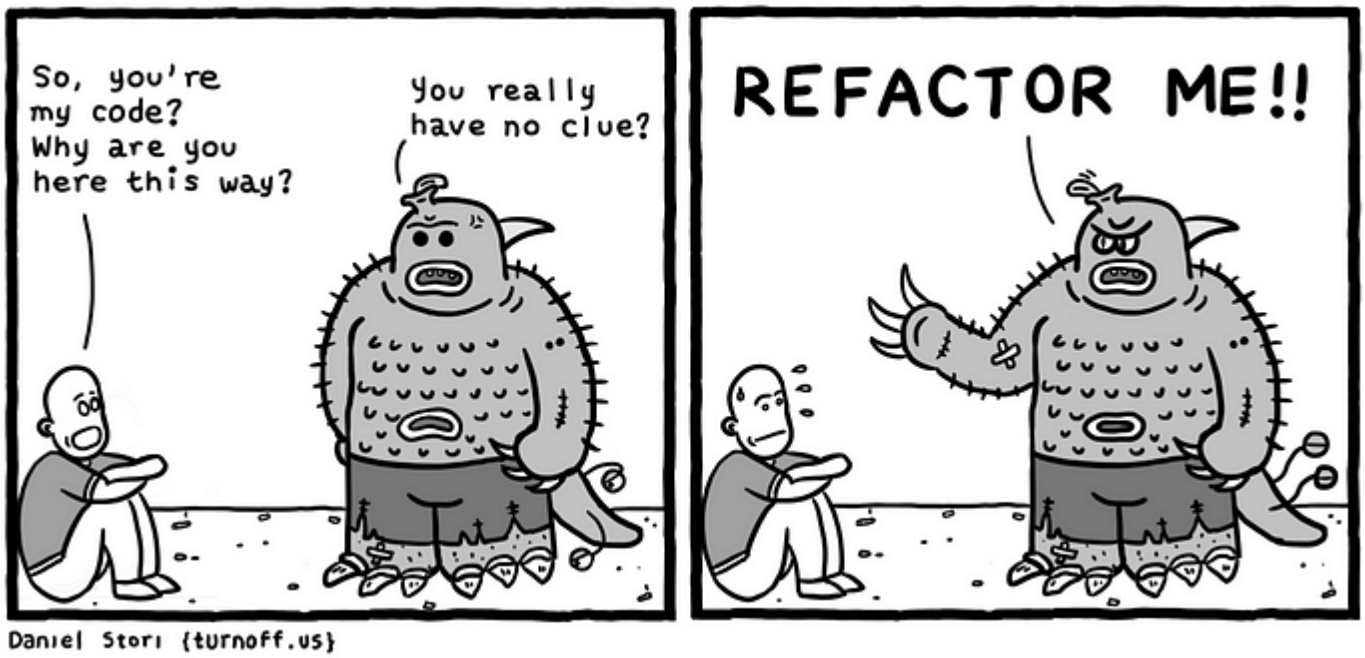
GABARITO

1. CORRETO
2. CORRETO
3. ERRADO
4. CORRETO
5. CORRETO
6. LETRA D

REFATORAÇÃO

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA



Uma importante atividade sugerida por diversas metodologias ágeis de desenvolvimento de software, **a refatoração é uma técnica (inclusive preconizada pelo Extreme Programming) de reorganização que simplifica o projeto (ou código) de um componente de software sem modificar sua função ou seu comportamento**. Martin Fowler define refatoração em seu livro da seguinte maneira:

> *"Refatoração é o processo de alterar um sistema de software de modo que o comportamento externo do código não se altere, mas a estrutura interna se aprimore. É uma forma disciplinada de organizar código [e modificar/simplificar o projeto interno] que minimiza as chances de introdução de bugs. Em resumo, ao se refatorar, se está aperfeiçoando o projeto de codificação depois de este ter sido feito.*

Quando um software é refatorado (também chamado de refabricado), o projeto existente é examinado em termos de redundância, elementos de projeto não utilizados, algoritmos ineficientes ou desnecessários, estruturas de dados mal construídas ou inapropriadas, **ou qualquer outra falha de projeto que possa ser corrigida para produzir um projeto melhor**. *Vamos ver um exemplo mais prático?*

Uma primeira iteração de projeto poderia gerar um componente que apresentasse baixa coesão (realizar três funções que possuem apenas relacionamento limitado entre si). Após cuidadosa

consideração, talvez decidamos que o componente devesse ser refabricado em três componentes distintos, cada um apresentando alta coesão. **O resultado será um software mais fácil de se integrar, testar e manter.**

Dito isso, a refatoração consiste na melhoria da estrutura interna do código-fonte de um programa, enquanto preserva seu comportamento externo. Prestem bastante atenção: refatoração não é reescrever o código; refatoração não é consertar bugs; refatoração não é melhorar aspectos observáveis do software (Ex: interface). Existem diversos outros benefícios por trás da refatoração...

Ela melhora atributos de código-fonte, tais como tamanho, duplicação, acoplamento, redundância, coesão, complexidade ciclomática, entre outros) que estão relacionados com facilidade de manutenção. **Além disso, ela ajuda a compreensão do código-fonte e encoraja o desenvolvedor a pensar sobre suas decisões de projeto**. *Bacana?* Vamos resumir o que vimos sobre refatoração antes de seguir para outros tópicos...

Refatoração é uma técnica utilizada na engenharia de software para melhorar a qualidade e a manutenibilidade do código de um programa. Consiste em modificar o código sem alterar seu comportamento externo, com o objetivo de deixá-lo mais organizado, legível e eficiente. A refatoração é importante porque o código de um programa costuma se tornar cada vez mais complexo e difícil de manter à medida que ele evolui.

A técnica permite que o código seja simplificado e reestruturado, facilitando sua manutenção e evolução. Entre os **benefícios da refatoração**, destacam-se a redução de custos de manutenção, já que um código mais limpo e organizado exige menos tempo e recursos para ser mantido; aumento da produtividade, já que o código é mais fácil de entender e modificar; e redução dos riscos de erros e problemas de segurança, já que o código é mais fácil de auditar e de garantir sua integridade.

Princípios Fundamentais

INCIDÊNCIA EM PROVA: BAIXA

Existem alguns princípios fundamentais que orientam a refatoração de software e ajudam a garantir que a técnica seja aplicada de forma segura e eficaz. Alguns dos principais princípios são:

PRINCÍPIOS	DESCRIÇÃO
MANTENHA O COMPORTAMENTO EXTERNO	Isso significa que, ao realizar uma refatoração, é importante garantir que o comportamento externo do programa não seja alterado. Isso é fundamental para que o programa continue funcionando corretamente após a refatoração.
SIMPLIFIQUE O CÓDIGO	O objetivo da refatoração é simplificar e melhorar o código do programa. Por isso, é importante eliminar código redundante, simplificar expressões complexas e tornar o código mais legível e fácil de entender.
TRABALHE COM TESTES AUTOMATIZADOS	Os testes automatizados são uma parte fundamental da refatoração, pois garantem que o programa continue funcionando corretamente após as mudanças realizadas. É importante criar testes para todas as partes do código que serão modificadas e executá-los antes e depois da refatoração.
REFATORE CONTINUAMENTE	A refatoração é uma atividade contínua, que deve ser realizada ao longo de todo o ciclo de vida do programa. Isso significa que a equipe de desenvolvimento deve estar constantemente procurando maneiras de melhorar o código e simplificá-lo.
USE PADRÕES DE PROJETO	Os padrões de projeto são soluções comprovadas para problemas comuns de design de software. Eles podem ser usados para simplificar o código e torná-lo mais fácil de entender e manter.

Para resumir nosso papo, vamos falar o que é uma refatoração por meio de algumas analogias: digamos que você tenha um guarda-roupa bagunçado, com roupas misturadas e desorganizadas. A refatoração seria como organizar esse guarda-roupa, separando as roupas por tipo, dobrando-as adequadamente e criando categorias para facilitar a busca. Ao refatorar o guarda-roupa, você melhora sua funcionalidade e torna mais fácil encontrar o que precisa.

A minha atividade principal de escrita de aula também passar por refatorações. À medida que eu escrevo, eu vou percebendo que algumas frases podem ser reformuladas para melhorar a clareza ou a coesão do texto de forma que vocês entendam melhor o assunto (essa aula de refatoração mesmo já foi refatorada algumas vezes). Isso é semelhante à refatoração de software, onde você faz alterações no código para torná-lo mais legível, eficiente e fácil de entender.

Identificação de Oportunidades

INCIDÊNCIA EM PROVA: BAIXA

Imagine que você acabou de se mudar para uma nova casa. No começo, você está feliz com a disposição dos móveis e a forma como a casa foi projetada**. No entanto, à medida que você começa a viver nela, você começa a perceber que alguns móveis estão no lugar errado ou não funcionam como deveriam.** Além disso, há algumas áreas que parecem estar subutilizadas ou mal aproveitadas.

Da mesma forma, quando os desenvolvedores de software criam um sistema, eles o projetam com base em seus conhecimentos e suposições naquele momento. **No entanto, à medida que o sistema é usado, os desenvolvedores podem perceber que algumas partes do código estão causando problemas, tornando-se difíceis de manter ou simplesmente não são mais necessárias.**

Como resultado, eles identificam oportunidades de refatoração, ou seja, aprimoramentos no design e no código do sistema para melhorar sua qualidade, manutenibilidade e eficiência. Assim como no exemplo da casa, os desenvolvedores podem utilizar a sua experiência e conhecimentos para identificar áreas problemáticas e oportunidades de melhoria no código-fonte do software.

Eles podem aplicar técnicas de refatoração para reorganizar o código, remover código duplicado, simplificar a lógica e melhorar a estrutura geral do sistema. Isso ajuda a garantir que o software continue atendendo às necessidades dos usuários e da empresa ao longo do tempo. **A identificação de oportunidades de refatoração no código existente é uma atividade importante para garantir a manutenibilidade e a qualidade do software.**

Ao identificar esses sinais de possível refatoração, a equipe de desenvolvimento pode melhorar significativamente a qualidade e a manutenibilidade do código do programa. Vejamos:

OPORTUNIDADES	DESCRIÇÃO
IDENTIFICAÇÃO DE CÓDIGO DUPLICADO	Se houver partes do código que se repetem em vários lugares do programa, pode ser uma oportunidade para refatorar e criar uma função ou classe separada para lidar com essa funcionalidade repetida.
MÉTODOS OU FUNÇÕES MUITO LONGOS	Métodos ou funções muito longos podem ser difíceis de entender e modificar. Se um método ou função for muito grande, pode ser uma oportunidade para dividi-lo em partes menores para melhorar a legibilidade e a manutenibilidade.
COMPLEXIDADE EXCESSIVA	Se um trecho de código é muito complexo e difícil de entender, pode ser uma oportunidade para simplificá-lo e torná-lo mais fácil de entender.
DIFICULDADE P/ ADICIONAR FUNCIONALIDADES	Se for difícil adicionar novas funcionalidades ao código existente, isso pode ser um sinal de que a estrutura do código precisa ser melhorada e simplificada.

CÓDIGO ANTIGO OU LEGADO	Código antigo ou legado pode estar desatualizado e precisar de refatoração para se tornar mais eficiente e manutenível.
PROBLEMAS DE PERFORMANCE	Se o código estiver executando lentamente ou apresentando problemas de performance, pode ser uma oportunidade para refatorar e melhorar o desempenho.

Técnicas de Refatoração

INCIDÊNCIA EM PROVA: BAIXA

Existem diversas técnicas de refatoração que podem ser aplicadas para melhorar o design e a qualidade do código. Algumas das principais técnicas incluem:

TÉCNICAS	DESCRIÇÃO
EXTRAÇÃO DE MÉTODO	Essa técnica envolve a extração de um trecho de código em um método separado. Isso é útil quando você tem um pedaço de código que é usado várias vezes em diferentes partes do seu programa. Ao extrair o código em um método separado, você pode reutilizá-lo em várias partes do seu programa, tornando o código mais limpo e fácil de manter.
MÉTODO INLINE	Essa técnica envolve a substituição de uma chamada de método por seu conteúdo real. Isso pode ser útil em situações em que um método é chamado apenas uma vez e é bastante simples, tornando desnecessária a criação de um método separado para ele. Isso também pode tornar o código mais fácil de entender e depurar.
EXTRAÇÃO DE VARIÁVEL	Essa técnica envolve a extração de uma expressão complexa em uma variável separada. Isso pode tornar o código mais legível e mais fácil de entender, especialmente se a expressão for usada várias vezes no programa. A extração de uma variável também pode tornar mais fácil alterar a expressão, pois você só precisa fazer a mudança em um lugar.
RENOMEAÇÃO	Essa técnica envolve a alteração do nome de uma variável, método ou classe para um nome mais descritivo e significativo. Isso pode tornar o código mais fácil de entender, especialmente se o nome original não for muito claro ou se for ambíguo.
EXTRAÇÃO DE CLASSE	Essa técnica envolve a extração de um conjunto de campos e métodos em uma nova classe separada. Isso pode tornar o código mais organizado e fácil de manter, especialmente se a classe original estiver ficando muito grande. A extração de classe também pode tornar o código mais modular e reutilizável.
DIVISÃO DE VARIÁVEL TEMPORÁRIA	Essa técnica envolve a divisão de uma variável temporária em duas ou mais variáveis separadas, cada uma responsável por uma parte específica da variável original. Isso pode tornar o código mais fácil de entender e mais modular, especialmente se a variável original estiver fazendo muitas coisas diferentes.
REMOÇÃO DE ATRIBUIÇÕES A PARÂMETROS	Essa técnica envolve a remoção de atribuições a parâmetros de um método. Isso pode tornar o código mais fácil de entender e menos propenso a erros, já que as alterações no parâmetro não afetarão a variável original passada ao método. A remoção de atribuições a parâmetros também pode tornar o código mais fácil de testar, pois não há efeitos colaterais inesperados no parâmetro passado.
SUBSTITUIÇÃO DE CÓDIGO DUPLICADO	Essa técnica consiste em identificar trechos de código que se repetem e substituí-los por uma função ou classe separada que possa ser reutilizada. Isso reduz a quantidade de código e torna o programa mais fácil de manter.

ENCAPSULAMENTO DE DADOS	Essa técnica consiste em tornar as variáveis privadas e criar métodos de acesso (*getters* e *setters*) para manipulá-las. Isso ajuda a garantir a integridade dos dados e torna o código mais fácil de entender e modificar.
DECOMPOSIÇÃO DE CLASSES	Essa técnica consiste em identificar classes que estão realizando muitas funcionalidades diferentes e dividi-las em classes menores e mais específicas. Isso ajuda a tornar o código mais modular e reutilizável.
GENERALIZAÇÃO DE CÓDIGO	Essa técnica consiste em identificar trechos de código que realizam a mesma funcionalidade em diferentes partes do programa e criar uma classe ou função genérica que possa ser reutilizada. Isso reduz a quantidade de código e torna o programa mais fácil de manter.

Teste de Refatoração

INCIDÊNCIA EM PROVA: BAIXA

Imagine que você tenha um carro antigo que precisa de alguns consertos e algumas melhorias para continuar funcionando bem. Para fazer isso, você leva seu carro a um mecânico especializado em carros antigos. O mecânico verifica as diferentes partes do carro e identifica as áreas que precisam ser atualizadas ou substituídas. **Antes de realizar essas atualizações, o mecânico faz alguns testes no carro para ter certeza de que o motor está funcionando corretamente.**

Ele testa se as luzes estão funcionando adequadamente e se o carro pode ser conduzido com segurança. Esses testes são necessários para garantir que o carro não apresente nenhum problema que possa colocar o motorista ou outras pessoas em risco. **Da mesma forma, os testes de refatoração são feitos em software para garantir que as atualizações e melhorias não tenham efeitos colaterais indesejados.**

Os testes verificam se o software ainda atende aos requisitos originais e se todas as funcionalidades ainda estão funcionando conforme o esperado. **Eles ajudam a garantir que o software seja seguro e confiável após as atualizações.** Assim como o mecânico verifica o carro antes e depois de fazer as atualizações, os desenvolvedores de software fazem testes antes e depois de realizar as refatorações.

Isso ajuda a garantir que o software continue a ser confiável e eficaz, mesmo após as atualizações e melhorias. Logo, testar a refatoração é uma etapa essencial, pois **garante que as alterações realizadas no código não introduzam novos erros e que o programa continue funcionando corretamente. Independentemente da abordagem escolhida**, é importante garantir que o teste de refatoração seja realizado de forma completa e rigorosa para garantir a integridade do código.

TESTE DE REFATORAÇÃO	DESCRIÇÃO
TESTE MANUAL	Essa abordagem envolve a execução manual do programa após a refatoração para verificar se todas as funcionalidades ainda estão funcionando corretamente. Essa abordagem é útil para testar pequenas alterações, mas pode ser demorada e propensa a erros.
TESTE DE REGRESSÃO	Essa abordagem envolve a criação de uma suíte de testes automatizados antes da refatoração e a execução desses testes após a refatoração para verificar se todas as funcionalidades ainda estão funcionando corretamente. Essa abordagem é mais eficiente e menos propensa a erros do que o teste manual.
TESTE DE UNIDADE	Essa abordagem envolve a criação de testes de unidade para as partes do código que foram alteradas durante a refatoração. Esses testes são executados automaticamente para garantir que as alterações não introduziram novos erros. Essa abordagem é mais precisa e eficiente do que o teste manual ou de regressão.

Além disso, é importante documentar todas as alterações realizadas durante a refatoração e manter um histórico de versões do código para facilitar a manutenção e evolução do software.

Code/Bad Smells

INCIDÊNCIA EM PROVA: BAIXA

Nós vimos insistentemente que a refatoração é o processo de melhorar a estrutura interna do código sem alterar seu comportamento externo. Nesse contexto, identificar indícios de código mal estruturado é fundamental para decidir quando e onde a refatoração é necessária. Esses "indícios" são conhecidos como **"Code Smells"** ou **"Bad Smells"**, e são sinais de que o código pode se beneficiar de uma reorganização para melhorar sua legibilidade, manutenção e extensibilidade.

Além dos testes de software, que verificam o comportamento funcional do código, ferramentas de análise estática – que buscam inspecionar o código-fonte sem executá-lo – desempenham um papel fundamental ao analisar o código-fonte em busca de code smells e outras potenciais falhas que possam não ser imediatamente visíveis através dos testes. Os Code Smells podem ser categorizados conforme veremos a seguir:

CATEGORIAS DE CODE SMELL	DESCRIÇÃO
BLOATERS	Código que cresce demais e se torna inchado. Essas são estruturas que acabam se tornando grandes e complexas demais, dificultando a compreensão e a manutenção do código.
OBJECT-ORIENTATION ABUSERS	Mau uso de conceitos de orientação a objetos. Ocorre quando os princípios de orientação a objetos são violados ou mal aplicados, resultando em um design fraco ou inadequado.
CHANGE PREVENTERS	Estruturas que dificultam a modificação do código. Essas smells tornam o código rígido, dificultando as mudanças e aumentando o risco de erros ao modificar o sistema.
DISPENSABLES	Código que pode ser removido sem impacto negativo. Trata-se de elementos que não têm utilidade real e que só adicionam complexidade desnecessária ao código.
COUPLERS	Acoplamento excessivo entre classes ou módulos. Isso ocorre quando há dependências fortes entre diferentes partes do sistema, dificultando a reutilização e a modificação.

Bloaters

Código que cresce demais e se torna inchado. Essas são estruturas que acabam se tornando grandes e complexas demais, dificultando a compreensão e a manutenção do código.

CATEGORIAS DE BLOATERS	DESCRIÇÃO
LONG METHOD	Métodos que são muito longos e fazem muitas coisas. Métodos com muitas linhas de código tornam-se difíceis de entender, testar e manter. Eles geralmente violam o princípio de responsabilidade única (SRP). Refatorar dividindo o método em vários métodos menores e mais focados é recomendado.
LARGE CLASS	Classes que têm muitas responsabilidades e contêm uma grande quantidade de código. Isso faz com que a classe seja difícil de entender e manter. Pode indicar que a classe está assumindo mais responsabilidades do que deveria, o que viola o princípio de

	responsabilidade única. Refatorar dividindo a classe em várias classes menores com responsabilidades específicas pode melhorar a manutenibilidade.
PRIMITIVE OBSESSION	Uso excessivo de tipos primitivos (como int, string, etc.) em vez de criar classes ou objetos mais ricos. Isso ocorre quando dados simples são usados para representar conceitos complexos, levando a código duplicado e dificuldade em manter a lógica associada a esses dados. Refatorar criando tipos ou classes específicos que encapsulam o comportamento associado pode resolver esse problema.
LONG PARAMETER LIST	Métodos que recebem muitos parâmetros. Uma lista longa de parâmetros pode indicar que o método faz mais do que deveria ou que há dados relacionados que não estão encapsulados em um objeto. Refatorar encapsulando parâmetros em objetos ou reduzindo o número de parâmetros pode melhorar a clareza e a facilidade de uso do método.
DATA CLUMPS	Grupos de dados que frequentemente aparecem juntos em vários lugares do código. Esses grupos de dados podem indicar que eles estão relacionados e deveriam ser encapsulados em uma classe ou objeto, em vez de serem passados separadamente. Refatorar para agrupar esses dados em uma única entidade pode reduzir a duplicação e aumentar a coesão.

Object-Orientation Abusers

Mau uso de conceitos de orientação a objetos. Ocorre quando os princípios de orientação a objetos são violados ou mal aplicados, resultando em um design fraco ou inadequado.

CATEGORIAS DE OO ABUSERS	DESCRIÇÃO
ALTERNATIVE CLASSES WITH DIFERENTE INTERFACES	Classes que fazem coisas similares, mas têm interfaces diferentes. Esse code smell ocorre quando duas ou mais classes realizam funções semelhantes, mas possuem interfaces diferentes, o que pode confundir e dificultar o uso consistente dessas classes. Refatorar unificando as interfaces ou utilizando herança/polimorfismo pode resolver esse problema.
REFUSED BEQUEST	Subclasses que herdam código ou comportamento de uma superclasse, mas não o utilizam ou o utilizam de maneira inadequada. Isso sugere que a herança foi mal aplicada e que a subclasse pode não ser realmente um tipo da superclasse, o que viola o princípio da substituição de Liskov. Refatorar a hierarquia de classes ou utilizar composição em vez de herança pode ser necessário.
SWITCH STATEMENTS	Uso excessivo de declarações switch ou if-else para lidar com diferentes comportamentos baseados em tipos. Esse code smell indica que o polimorfismo poderia ser usado em vez de condicionais para evitar duplicação de código e melhorar a extensibilidade. Refatorar utilizando o padrão de projeto Strategy ou criando classes específicas para cada caso pode resolver esse problema.
TEMPORARY FIELDS	Campos em uma classe que só são usados em certas circunstâncias. Esses campos tornam o código mais confuso, pois aumentam a complexidade sem adicionar valor quando não são utilizados. Refatorar movendo esses campos para classes mais apropriadas ou criando classes específicas para os casos em que esses campos são relevantes pode ajudar a manter o código mais limpo e compreensível.

Change Preventers

Estruturas que dificultam a modificação do código. Essas smells tornam o código rígido, dificultando as mudanças e aumentando o risco de erros ao modificar o sistema.

CATEGORIAS DE CHANGE PREVENTERS	DESCRIÇÃO
DIVERGENTE CHANGE	Ocorre quando uma única classe precisa ser modificada por vários motivos diferentes. Isso geralmente acontece quando uma classe assume muitas responsabilidades diferentes, resultando em múltiplas alterações em resposta a diferentes tipos de mudanças no software. Esse code smell viola o princípio de responsabilidade única (SRP). Refatorar dividindo a classe em várias classes menores, cada uma com uma responsabilidade específica, pode resolver o problema.
PARALLEL INHERITANCE HIERARCHIES	Ocorre quando há uma correspondência direta entre as hierarquias de classes em dois ou mais pacotes, e alterações em uma hierarquia exigem mudanças na outra. Isso indica que as hierarquias estão fortemente acopladas, tornando difícil modificar uma sem afetar a outra. Refatorar usando composição em vez de herança ou unificando as hierarquias pode ajudar a reduzir esse acoplamento e simplificar o código.
SHOTGUN SURGERY	Ocorre quando uma pequena mudança em um sistema exige modificações em muitos lugares diferentes. Isso torna a manutenção do código complexa e propensa a erros, pois é fácil esquecer de modificar uma das partes necessárias. Refatorar centralizando a responsabilidade em uma única classe ou módulo, ou melhorando a coesão, pode reduzir a necessidade de mudanças espalhadas e facilitar a manutenção.

Dispensables

Código que pode ser removido sem impacto negativo. Trata-se de elementos que não têm utilidade real e que só adicionam complexidade desnecessária ao código.

CATEGORIAS DE DISPENSABLES	DESCRIÇÃO
COMMENTS	Uso excessivo de comentários para explicar o código. Comentários muitas vezes indicam que o código é confuso ou difícil de entender. Em vez de depender de comentários, o código deve ser refatorado para ser mais claro e autoexplicativo. Bons nomes de variáveis, métodos e uma estrutura de código limpa geralmente tornam os comentários desnecessários.
DUPLICATE CODE	Blocos de código idênticos ou muito semelhantes aparecem em vários lugares. Código duplicado aumenta a dificuldade de manutenção, pois qualquer alteração precisa ser replicada em vários locais, aumentando o risco de inconsistências. Refatorar o código para eliminar duplicações, criando métodos ou classes reutilizáveis, é a melhor prática.
DATA CLASS	Classes que contêm apenas campos de dados e métodos getters e setters. Essas classes não têm comportamento real e geralmente são usadas apenas para transportar dados. Isso pode indicar que a lógica associada a esses dados está espalhada por várias partes do código.

	Refatorar movendo métodos relevantes para essas classes, tornando-as mais coesas e encapsulando o comportamento associado aos dados.
DEAD CODE	Código que nunca é usado ou não tem impacto no funcionamento do programa. Dead code aumenta a complexidade do código sem adicionar valor. Ele pode confundir os desenvolvedores e dificultar a manutenção. A remoção de código morto simplifica o código e reduz o risco de bugs.
LAZY CLASS	Classes que não fazem muito ou não têm uma responsabilidade clara. Essas classes foram criadas com algum propósito, mas acabaram sendo subutilizadas ou tornaram-se obsoletas. Manter classes preguiçosas pode adicionar desordem ao código. Refatorar para eliminar a classe ou combinar suas responsabilidades com outra classe relevante pode ser a solução.
SPECULATIVE GENERALITY	Código escrito para acomodar futuras necessidades que nunca se concretizam. Isso geralmente acontece quando os desenvolvedores tentam antecipar requisitos que podem nunca surgir, resultando em complexidade desnecessária. A melhor prática é remover ou simplificar essas generalizações, mantendo o código focado nas necessidades atuais.

Couplers

Acoplamento excessivo entre classes ou módulos. Isso ocorre quando há dependências fortes entre diferentes partes do sistema, dificultando a reutilização e a modificação.

CATEGORIAS DE COUPLERS	DESCRIÇÃO
FEATURE ENVY	Métodos em uma classe que têm mais interesse em dados de outra classe do que em seus próprios dados. Isso ocorre quando um método em uma classe acessa os campos ou métodos de outra classe com muita frequência, sugerindo que a funcionalidade deveria estar localizada na classe que contém os dados. Refatorar movendo o método para a classe que contém os dados que ele está manipulando pode resolver esse problema.
INAPPROPRIATE INTIMACY	Duas classes que interagem de maneira muito próxima, conhecendo detalhes internos uma da outra. Isso cria um forte acoplamento entre as classes, tornando-as dependentes e dificultando a manutenção. Refatorar utilizando princípios de encapsulamento e separação de responsabilidades, como dividir funcionalidades em classes auxiliares ou usar herança, pode reduzir essa intimidade inadequada.
INCOMPLETE LIBRARY CLASS	Bibliotecas de classes fornecidas por terceiros que não oferecem toda a funcionalidade necessária, levando a extensões ou trabalho adicional para compensar. Isso pode resultar em uma dependência de código externo que não é facilmente mantido. Uma solução pode ser encapsular a biblioteca em uma classe adaptadora que fornece a funcionalidade necessária ou, se possível, contribuir para a biblioteca para adicionar a funcionalidade ausente.
MESSAGE CHAINS	Cadeias longas de chamadas de métodos, onde um objeto chama um método em outro objeto, que por sua vez chama um método em outro objeto, e assim por diante. Isso cria um acoplamento frágil entre várias classes e pode dificultar a manutenção e a depuração do código. Refatorar para reduzir a cadeia de mensagens, talvez utilizando um padrão como o Facade, pode tornar o código mais claro e menos dependente.
MIDDLE MAN	Classes que adicionam pouco ou nenhum valor ao delegar todas as suas funções para outra classe. Quando uma classe simplesmente repassa métodos para outra classe, sem adicionar lógica ou comportamento, ela pode ser considerada desnecessária. Refatorar eliminando a

	classe intermediária e permitindo que as classes consumidoras acessem diretamente a classe final pode simplificar o código.

Refatoração Contínua

INCIDÊNCIA EM PROVA: BAIXA

Um conceito inovador é a refatoração contínua, que é uma prática que envolve incorporar a refatoração no processo de desenvolvimento contínuo de software, de forma a manter o código limpo, organizado e fácil de manter. **A ideia é que, ao invés de realizar grandes refatorações em momentos pontuais do projeto, a equipe realize pequenas refatorações constantemente, à medida que trabalha no código.**

Para incorporar a refatoração contínua no processo de desenvolvimento, é importante que a equipe de desenvolvimento tenha um bom entendimento dos princípios de refatoração, bem como das técnicas e ferramentas disponíveis. **Além disso, é importante que a equipe esteja comprometida em manter o código limpo e fácil de manter, e que estejam dispostos a dedicar tempo para realizar pequenas refatorações ao longo do desenvolvimento.**

Algumas práticas comuns para incorporar a refatoração contínua no processo de desenvolvimento incluem:

- Realizar pequenas refatorações frequentemente, sempre que for identificado um código duplicado, um método muito grande ou uma classe mal organizada, por exemplo.
- Utilizar ferramentas de refatoração para facilitar o processo de refatoração contínua, automatizando tarefas repetitivas e tornando o processo mais eficiente.
- Integrar a refatoração no processo de revisão de código, de forma que o código refatorado seja revisado e testado antes de ser incorporado ao código-base.
- Utilizar testes automatizados para validar as refatorações, de forma a garantir que as alterações não tenham introduzido novos erros ou comportamentos inesperados no código.

Com a refatoração contínua, a equipe de desenvolvimento pode manter o código sempre limpo e organizado, reduzindo a quantidade de tempo e esforço necessários para realizar grandes refatorações em momentos pontuais do projeto. **Além disso, um código bem organizado e fácil de manter pode ajudar a reduzir os custos de manutenção do software, melhorar a qualidade do produto final e aumentar a satisfação do usuário.**

Existem algumas práticas recomendadas que podem ajudar a maximizar os benefícios da refatoração e minimizar os riscos envolvidos no processo. Algumas dessas práticas incluem:

PRÁTICAS RECOMENDADAS	DESCRIÇÃO
PLANEJE A REFATORAÇÃO	Antes de começar a refatorar, planeje cuidadosamente o que você quer fazer e como irá fazer. Isso pode incluir a identificação das áreas que precisam de refatoração, a escolha

	das técnicas de refatoração apropriadas e a elaboração de um plano para minimizar os riscos.
REFATORE EM PEQUENOS PASSOS	Refatorar em pequenos passos ajuda a minimizar os riscos envolvidos no processo, permitindo que você teste as alterações e valide-as antes de avançar para a próxima etapa. Além disso, isso também ajuda a manter o código sempre funcional, evitando que o processo de refatoração cause interrupções no desenvolvimento.
USE TÉCNICAS DE VERSIONAMENTO	Antes de iniciar a refatoração, crie uma nova branch do seu código e trabalhe nela. Dessa forma, você pode testar e validar as alterações sem afetar a versão principal do código, garantindo que você possa reverter facilmente para uma versão anterior, caso algo dê errado.
TESTE RIGOROSAMENTE	Testar rigorosamente é fundamental para garantir que a refatoração não introduza novos erros ou comportamentos inesperados no código. Certifique-se de criar testes automatizados para validar as alterações e execute testes manuais sempre que possível.
ENVOLVER TODA A EQUIPE	Refatorar é um trabalho que deve ser feito em equipe. Envolver todos os membros da equipe pode ajudar a identificar áreas que precisam de refatoração, garantir que as alterações sejam compreendidas por todos e ajudar a minimizar os riscos envolvidos no processo.
APRENDER E PRATICAR CONSTANTEMENTE	Refatoração é uma habilidade que requer prática constante. Aprenda novas técnicas de refatoração, pratique sempre que possível e busque feedback da equipe para melhorar continuamente.

Estudos de Caso

INCIDÊNCIA EM PROVA: BAIXA

Existem estudos de caso de refatoração que mostram como a aplicação de técnicas de refatoração pode melhorar significativamente a qualidade do código e a eficiência do desenvolvimento:

ESTUDOS DE CASO	DESCRIÇÃO
REFATORAÇÃO DO CÓDIGO-FONTE DO ECLIPSE	O Eclipse é uma plataforma de desenvolvimento popular que é construída em Java. Em 2005, o time de desenvolvimento do Eclipse realizou uma grande refatoração do código-fonte para melhorar sua qualidade. Eles usaram diversas técnicas de refatoração para simplificar e limpar o código, e conseguiram reduzir o tamanho do código em mais de 20%.
REFATORAÇÃO DO CÓDIGO DO KENT BECK	Kent Beck, um dos criadores do Extreme Programming (XP), escreveu um livro chamado "Test Driven Development: By Example". Neste livro, ele descreveu como refatorou um sistema bancário legado para melhorar sua qualidade e desempenho. Ele usou diversas técnicas de refatoração para simplificar o código, remover duplicação e melhorar a estrutura do sistema.
REFATORAÇÃO DO CÓDIGO DO LINKEDIN	O LinkedIn, uma rede social para profissionais, realizou uma grande refatoração do seu código em 2014. Eles usaram diversas técnicas de refatoração para melhorar a qualidade do código e torná-lo mais fácil de manter. Como resultado, eles conseguiram reduzir o número de erros e melhorar significativamente a performance do site.
REFATORAÇÃO DO CÓDIGO DO GOOGLE	O Google é conhecido por sua abordagem rigorosa para a qualidade do código. Eles realizam regularmente grandes refatorações em seus sistemas para melhorar sua qualidade e desempenho. Por exemplo, em 2006, eles realizaram uma grande refatoração no Google Maps, que melhorou significativamente sua performance e tornou-o mais fácil de manter.

Reengenharia x Refatoração

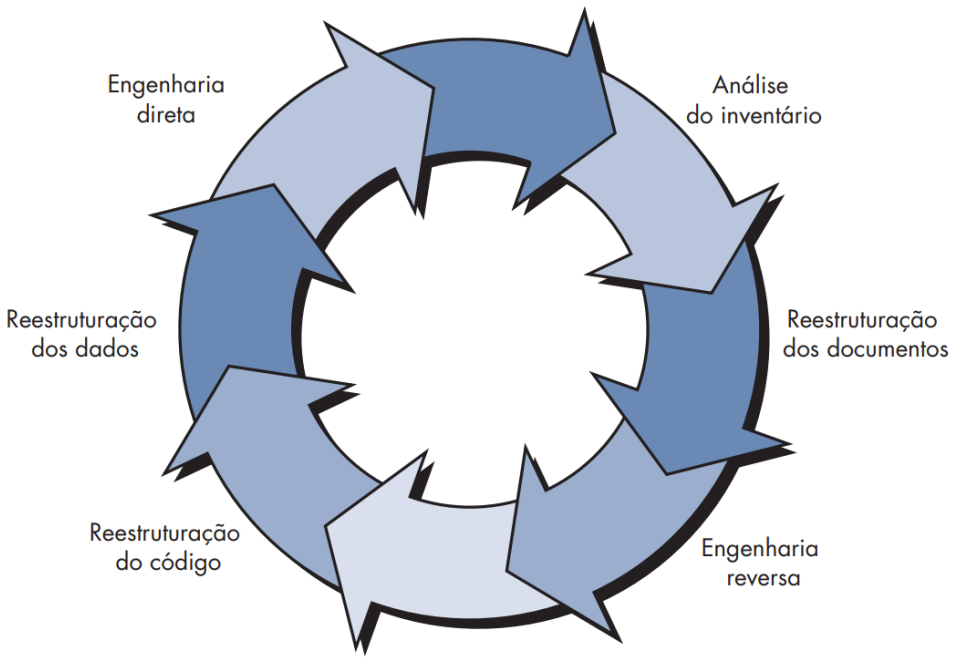
INCIDÊNCIA EM PROVA: BAIXA

Por fim, é importante diferenciar a refatoração de reengenharia. Pressman explica assim: imaginem que um aplicativo atendeu às necessidades de negócio de uma empresa por dez ou quinze anos. **Durante esse tempo, ele foi corrigido, adaptado e aperfeiçoado em diversas oportunidades.** Profissionais realizaram este trabalho com as melhores intenções, mas as boas práticas de engenharia de software foram sempre deixadas de lado (devido à pressão por outros aspectos).

Agora o aplicativo está instável. **Ainda funciona, mas sempre que se tenta fazer uma alteração, ocorrem efeitos colaterais sérios e inesperados. No entanto, o aplicativo deve continuar evoluindo.** *O que fazer?* Software que não pode ser mantido não é novidade. Na verdade, a ênfase cada vez maior sobre a engenharia de software foi motivada pelos problemas de manutenção criados por mais de quatro décadas.

Reengenharia toma tempo, tem um custo significativo em dinheiro e absorve recursos que poderiam de outra forma ser usados em necessidades mais imediatas. Por todas essas razões, a reengenharia não é realizada em alguns meses ou mesmo anos. A reengenharia dos sistemas de informação é uma atividade que absorverá recursos da tecnologia da informação por muito tempo, logo todas as organizações precisam de uma estratégia pragmática para reengenharia de software

Segue abaixo um modelo de processo de reengenharia de software. Em alguns casos, essas atividades ocorrem em sequência linear, mas nem sempre é o caso. Por exemplo: pode acontecer de a engenharia reversa ter de ocorrer antes do início da reestruturação dos documentos. Deve-se realizar uma análise de inventário, isto é, toda organização de software deve ter um inventário de todos os aplicativos com informações detalhadas (tamanho, idade, criticidade no negócio, etc).



Assim que surgem candidatos à reengenharia**. Deve ocorrer uma reestruturação dos documentos – isso não significa que se deve documentar absolutamente tudo.** No entanto, há um mínimo necessário e, de preferência, deve estar atualizado. A engenharia reversa daria um livro inteiro, mas trata do processo de analisar um programa na tentativa de criar uma representação do programa em um nível mais alto de abstração do que o código-fonte.

É como se eu pegasse um aplicativo pronto e chegasse em sua documentação de análise, projeto, requisitos, entre outros. Em seguida, temos a reestruturação do código, que reestrutura ou reescreve o código em uma tecnologia mais moderna ou de forma mais eficiente, preservando a funcionalidade e a arquitetura global. Depois, temos a reestruturação dos dados, em que se identificam objetos de dados, e as estruturas de dados existentes são revisadas quanto à qualidade.

Por fim, a engenharia direta recupera as informações do projeto de software existente e usa as informações para alterar ou reconstituir o sistema existente em um esforço para melhorar sua qualidade geral. Em muitos casos, o software que passou pela reengenharia reimplementa a função do sistema existente e também acrescenta novas funções e/ou melhora o desempenho geral da aplicação ou componente.

Reengenharia (também chamada Manutenção Preventiva) é, portanto, o exame, análise e reestruturação de um sistema de software existente para reconstituí-lo em uma nova forma. O objetivo da reengenharia é: compreender os atuais artefatos de software, isto é, especificação, projeto, implementação e documentação; e melhorar a funcionalidade e qualidade de atributos do sistema.

Alguns exemplos de atributos de qualidade são: capacidade de evolução, desempenho e capacidade de reutilização. Fundamentalmente, um novo sistema é gerado a partir de um sistema operacional, de tal modo que o sistema de destino possua melhores fatores de qualidade, como confiabilidade, exatidão, integridade, eficiência, facilidade de manutenção, facilidade de utilização, flexibilidade, capacidade de teste, a interoperabilidade, reusabilidade e portabilidade.

Em outras palavras, reengenharia é feita para converter um sistema de "ruim" em um sistema de "bom". Claro que existem riscos envolvidos nesta transformação (Ex: o sistema novo não pode ter qualidade inferior ao anterior). Sistemas de software são redesenhados, mantendo um ou mais dos quatro objetivos gerais seguintes: melhorar manutenibilidade; migração para uma nova tecnologia; melhorar a qualidade do software; e preparar para melhorias funcionais.

Uma boa compreensão dos processos de desenvolvimento de software é útil em fazer um plano de reengenharia. Vários conceitos aplicados durante o desenvolvimento de software são fundamentais para a reengenharia. Por exemplo: abstração e requinte são os principais conceitos utilizados no desenvolvimento de software, e ambos os conceitos são igualmente úteis na reengenharia.

O princípio da abstração afirma que o nível de abstração da representação de um sistema pode ser aumentado gradualmente substituindo sucessivamente os detalhes com informações abstratas, isto é, retirando detalhes menos importantes. **Por meio da abstração, pode-se produzir uma visão que incide sobre as características selecionadas do sistema ao esconder informações sobre outras características.**

Já o princípio do refinamento afirma que o nível de abstração da representação do sistema é gradualmente reduzido por sucessivas substituições de alguns aspectos do sistema com mais detalhes. Enfim, galera... a definição que eu mais gosto é do Sommerville e diz assim: **reengenharia é a reorganização e modificação de sistemas de software existentes, parcial ou totalmente, para torná-los mais manuteníveis.**

QUESTÕES COMENTADAS – DIVERSAS BANCAS

1. **(CESPE / SEFIN de Fortaleza-CE – 2023)** O processo de refatoração de software melhora o design do código existente, mas não substitui nem altera o seu comportamento.

Comentários:

A refatoração de software envolve a melhoria do design do código existente, como a reorganização de estruturas e a eliminação de duplicações, sem alterar o comportamento externo ou funcionalidade do software. O objetivo é tornar o código mais limpo, eficiente e fácil de manter, mantendo o mesmo resultado final.

Gabarito: Correto

---

2. **(CESPE / CAU-BR – 2024)** Um dos princípios da componentização de software é o OCP (open-closed principle), que define que um componente pode ser estendido sem necessidade de modificações internas.

Comentários:

O princípio do Open-Closed Principle (OCP) é um dos princípios fundamentais da programação orientada a objetos e também se aplica à componentização de software. Ele estabelece que um componente deve ser aberto para extensão, mas fechado para modificação. Isso significa que a funcionalidade de um componente pode ser estendida sem alterar seu código fonte, promovendo a reutilização e a manutenção do software sem introduzir novos erros.

Gabarito: Correto

---

3. **(FUNDATEC / BRDE - 2023)** Avalie as assertivas abaixo:

I. Melhoria do design interno (arquitetura) do software.
II. Código mais legível.
III. Localização de bugs.
IV. Mudança do comportamento externo do software.

Quantas podem vir a ser benefícios do processo de refatoração de código?

a) 0
b) 1
c) 2
d) 3
e) 4

Comentários:

Podem ser benefícios do processo de refatoração de código:

I. Melhoria do design interno (arquitetura) do software.
II. Código mais legível.
III. Localização de bugs.

A assertiva IV não é um benefício da refatoração, dado que ela busca alterar um software de uma maneira que não mude o seu comportamento externo e ainda melhore a sua estrutura interna.

Gabarito: Letra D

---

4. **(AOCP / CODEBA – 2023)** No contexto do desenvolvimento de software, refatoração é uma técnica que visa melhorar a estrutura interna do código sem alterar seu comportamento externo. Assinale a alternativa correta sobre refatoração.

a) Refatoração torna o código mais difícil de entender, aumentando a complexidade.
b) Ao refatorar, os testes existentes são irrelevantes, pois a refatoração não afeta o comportamento do software.
c) Refatoração pode ajudar a simplificar o código, tornando-o mais modular e reutilizável.
d) Refatoração frequentemente resulta em uma completa reescrita do código, alterando seu comportamento externo.
e) Code smells são indícios de que o código está bem otimizado e não requer refatoração.

Comentários:

(a) Errado. A refatoração visa justamente tornar o código mais claro e simples, reduzindo a complexidade;

(b) Errado. Testes são fundamentais durante a refatoração para garantir que o comportamento externo do software permaneça inalterado;

(c) Correto. A refatoração tem como objetivo simplificar o código, tornando-o mais modular e facilitando sua reutilização;

(d) Errado. A refatoração melhora a estrutura interna do código sem alterar seu comportamento externo, não envolve reescrita completa;

(e) Errado. Code smells são sinais de que o código pode estar mal estruturado e pode se beneficiar de refatoração.

Gabarito: Letra C

---

5. **(CESPE / PETROBRÁS – 2022)** De acordo com o conceito de code smell, a categoria bloaters engloba os trechos de código que são irrelevantes e podem ser excluídos sem afetar a aplicação.

Comentários:

A categoria bloaters no conceito de code smell refere-se a trechos de código que se tornaram grandes e complexos demais, dificultando a manutenção e a compreensão. Exemplos incluem classes e métodos muito longos ou com muitas responsabilidades. Esses códigos não são irrelevantes, mas precisam ser refatorados para melhorar a clareza e eficiência, sem serem simplesmente excluídos.

Gabarito: Errado

---

6. **(IDECAN / UFBA – 2022)** Além do teste de software, podem ser identificados problemas no código através de ferramentas de análise estática em busca de code smells.

Comentários:

Ferramentas de análise estática de código são usadas para identificar problemas potenciais no código, como code smells, que são indicadores de fragilidade ou má qualidade no design. Essas ferramentas analisam o código-fonte sem executá-lo, detectando possíveis defeitos, vulnerabilidades de segurança, violações de padrões de codificação e outras questões que podem afetar a manutenibilidade e a qualidade do software.

Gabarito: Correto

---

7. **(QUADRIX / PRODAM-AM - 2022)** Assinale a alternativa que apresenta a prática de XP (Extreme Programming) que é definida como uma técnica disciplinada para reestruturar um corpo de código existente, alterando a sua estrutura interna sem alterar seu comportamento externo. Essa prática mantém a semântica do código, ou seja, após as mudanças, o código ainda funciona da mesma forma.

a) refatoração
b) metáfora
c) *test-driven development*
d) *coding standards*
e) *on-site customer*.

Comentários:

A prática descrita na questão é a refatoração. A refatoração é uma prática da metodologia XP (Extreme Programming) que consiste em melhorar a estrutura interna do código sem alterar o seu comportamento externo, mantendo a sua semântica. A refatoração é uma técnica disciplinada que ajuda a manter o código limpo, legível, fácil de manter e evoluir.

Gabarito: Letra A

---

8. **(FUNDEP / UFJF - 2022)** Considere o trecho de código a seguir, que acabou de ser refatorado.

```
delta = b*b-4*a*c; // nova variável
x1 = (-b + sqrt(delta)) / (2*a);
x2 = (b + sqrt(delta)) / (2*a);
```

Assinale a refatoração aplicada para essa situação.

a) Inline de método.
b) Extração de método.
c) Extração de variável.
d) Renomeação.
e) Extração de classe.

Comentários:

A refatoração aplicada para esse trecho de código foi a extração de variável. Foi criada uma nova variável chamada **delta** para armazenar o resultado do cálculo **b*b-4*a*c**. Essa refatoração tem como objetivo tornar o código mais legível e fácil de entender, além de evitar a repetição do cálculo em ambos os cálculos de raízes.

Gabarito: Letra C

---

9. **(CESPE / BANRISUL - 2022)** A filosofia da modelagem ágil não admite decisões que levem um projeto a sua rejeição e a sua refatoração, pois os profissionais de tecnologia não possuem todas as respostas e outros stakeholders envolvidos no negócio devem ser respeitados e integrados ao processo.

Comentários:

Opa! A filosofia da modelagem ágil, ao contrário do que foi afirmado, considera que é normal e esperado que as decisões tomadas possam levar a ajustes, refatorações ou até mesmo rejeição de um projeto, pois o objetivo é desenvolver soluções iterativamente e de forma incremental, com a participação dos stakeholders e validações constantes. A metodologia ágil valoriza a adaptabilidade e a flexibilidade para lidar com mudanças, por isso a refatoração é vista como uma prática importante para melhorar a qualidade do software.

Gabarito: Errado

---

10. (CESPE / MC - 2022) Refatorar um *software* consiste em modificar o seu comportamento interno e externo, mantendo-se inalterada sua estrutura interna.

Comentários:

Opa! Refatorar um software consiste em melhorar sua estrutura interna sem alterar o comportamento externo, mantendo a mesma semântica e funcionalidade do software. Ou seja, a refatoração não deve modificar o comportamento externo do software, pois isso poderia afetar a sua integridade e desestabilizar a aplicação.

Gabarito: Errado

---

11. (CONSULPAM / Prefeitura de Irauçuba - CE - 2022) Dentro das metodologias ágeis, o processo de desenvolvimento de software especificado pela Programação Extrema (eXtreme Programming, XP) possui algumas características específicas. Uma das características do XP versa sobre as necessidades de melhoria no projeto, que devem ser realizadas através de um tipo de processo específico para este fim. Assinale a alternativa com o nome deste tipo de processo.

a) Testes.
b) Refatoração.
c) Histórias do Usuário.
d) Programação em Pares.

Comentários:

A característica descrita na questão refere-se à prática da refatoração na metodologia XP (Extreme Programming). A refatoração é um processo específico para a melhoria contínua do projeto de software, que consiste em reestruturar o código sem alterar seu comportamento externo, mantendo a semântica e funcionalidade do software.

Gabarito: Letra B

---

12. (IDECAN / IF-CE - 2021) No que diz respeito à manutenção e reengenharia de software, um termo define o processo de alterar o código-fonte, de modo que não altere o comportamento externo e ainda melhore a sua estrutura interna. É uma técnica disciplinada de limpar e organizar o código, e por consequência, minimizar a chance de introduzir novos bugs. Esse termo é conhecido como

a) elicitação.
b) refatoração.
c) recodificação.

d) replicação.

Comentários:

O termo que define o processo de alterar o código-fonte, de modo que não altere o comportamento externo e ainda melhore a sua estrutura interna é a refatoração. A refatoração é uma técnica disciplinada de limpar e organizar o código, a fim de torná-lo mais legível, fácil de entender e de manter, além de minimizar a chance de introduzir novos bugs.

Gabarito: Letra B

---

13. **(IBADE / Prefeitura de Vilhena - 2019)** Em Orientação a Objetos define-se o processo de Refatoração como:

a) processo de criação de entidades com herança e polimorfismo.
b) classe abstrata que permite a criação de métodos fatorados.
c) classe que permite herdar da classe pai todos os seus objetos.
d) processo para melhorar a legibilidade ou a eficiência do código, preservando seu comportamento.
e) objeto que permite acessar classes fatoradas.

Comentários:

Em Orientação a Objetos, o processo de Refatoração é definido como o processo para melhorar a legibilidade ou a eficiência do código, preservando seu comportamento. A Refatoração é uma técnica disciplinada de limpar e organizar o código, reestruturando sua estrutura interna, sem alterar seu comportamento externo, mantendo a semântica e funcionalidade do software.

Gabarito: Letra D

---

14. **(CESPE / MPC-PA - 2019)** No contexto da manutenção preventiva de sistemas no paradigma orientado a objetos, a refatoração é uma técnica empregada com o objetivo de:

a) adicionar uma nova funcionalidade no sistema.
b) incluir uma generalidade que possa ser necessária no futuro.
c) realizar a reengenharia de um sistema.
d) reduzir o risco da introdução de novos erros no programa.
e) substituir métodos similares em subclasses por um único método em uma superclasse.

Comentários:

(a) Errado. A refatoração não tem como objetivo adicionar novas funcionalidades ao sistema. Ela se concentra em melhorar a qualidade do código existente, sem adicionar novas funcionalidades;

(b) Errado. Desconheço o sentido de generalidade apresentada no texto desse item. De toda forma, a refatoração busca melhorar a qualidade do código;

(c) Errado. Refatoração é diferente de reengenharia – a primeira não busca mudar o comportamento externo do software; já a segunda, sim;

(d) Correto. Ao melhorar a qualidade do código existente, tornando-o mais legível, eficiente e fácil de entender, a refatoração ajuda a prevenir a introdução de novos erros no programa.

(e) Correto. A substituição de métodos similares em subclasses por um único método em uma superclasse é uma técnica que pode ser aplicada para fazer a refatoração.

A banca considerou o item (d) como errado, mas eu discordo. Após a refatoração, é interessante fazer testes de regressão para verificar se novos erros não foram inseridos. Logo, é claro que a refatoração ajuda a reduzir o risco da introdução de novos erros por conta da melhoria da qualidade do software. Enfim, divirjo respeitosamente da banca e acredito que a questão deveria ser anulada.

Gabarito: Letra E

---

15. **(IF-PA / IF-PA - 2019)** Ao analisarmos uma classe Java, nos deparamos com um método que implementa diversas funcionalidades, tornando-se um método com muitas linhas de código, de difícil compreensão e manutenção. Para melhorar essa situação, decidimos dividi-lo em métodos menores, mais fáceis de entender e de efetuar manutenções. A esse processo de organizar e melhorar a estrutura interna de uma aplicação, denominamos de:

a) indentação.
b) depuração.
c) inspeção.
d) integração.
e) refatoração.

Comentários:

A esse processo de organizar e melhorar a estrutura interna de uma aplicação, dividindo um método em métodos menores, mais fáceis de entender e de efetuar manutenções, denominamos de refatoração. A indentação se refere à organização do código para melhorar sua legibilidade e compreensão. Depuração é o processo de encontrar e corrigir erros em um programa. A inspeção é uma atividade de revisão do código realizada por um ou mais desenvolvedores. A integração se refere ao processo de combinar diferentes partes de um sistema de software para formar um todo coeso.

Gabarito: Letra E

---

16. (CESPE / SLU – 2019) Refactoring (refatoração) é o processo utilizado para reescrever aplicações desatualizadas, com a finalidade de incrementar e melhorar suas funcionalidades; o uso dessa técnica normalmente aprimora aplicações para disponibilizá-las na Internet.

Comentários:

A refatoração é uma técnica de reorganização que simplifica um projeto, sem modificar suas funções ou seu comportamento. Logo, a questão erra ao afirmar que a finalidade é incrementar e melhorar as funcionalidades – isso não é possível na refatoração, de modo que o comportamento deve ser preservado.

Gabarito: Errado

---

17. (AOCP / PRODEB – 2018) "Processo de alteração de um sistema de software de tal forma que não se altere o comportamento externo do código, mas se aprimore a estrutura interna". O enunciado se refere a:

a) Reúso.
b) Teste Unitário.
c) Teste de Integração.
d) Refatoração.
e) Prototipação.

Comentários:

(a) Errado, o reúso é uma estratégia para reutilização de um software existente; (b) Errado, o teste unitário é um teste que foca na lógica interna de processamento; (c) Errado, o teste de integração verifica se os componentes de um sistema trabalham corretamente juntos; (d) Correto, trata-se da refatoração; (e) Errado, a prototipação se baseia na ideia de construção de um protótipo (esboços, desenhos ou imagens) que auxiliem a construção do produto.

Gabarito: Letra D

---

18. (CS-UFG / Câmara de Goiânia – 2018) Sejam as classes A e B tais que o relacionamento entre elas é dado pelo fato de A usar (referenciar) a classe B. Dessa forma, qual das refatorações a seguir implementa o princípio da inversão de dependência?

a) Cria interface para serviços oferecidos por B; a classe A passa a usar a interface criada; a classe B passa a implementar a interface criada; a classe A não usa mais a classe B.

b) Cria interface para serviços oferecidos por A; a classe A passa a implementar a interface criada; a classe B passa a usar a interface criada; a classe A não usa mais a classe B.

c) Cria um relacionamento de herança entre as classes A e B (A torna-se uma especialização de B); métodos da classe B empregados pela classe A são migrados para a classe A; a classe A não usa mais a classe B.

d) Cria uma referência para a classe B na classe A; cria um método para receber uma instância de B (injeção de dependência) e guarda-a na referência criada; a classe A não usa mais a classe B.

Comentários:

Basicamente o princípio da inversão de dependência consiste em: Uma interface deve ser projetada pela classe que a usará, e não pela classe que a implementará. Além disso, ele afirma que: (1) Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações; (2) Abstrações não devem depender de detalhes. Os detalhes devem depender das abstrações. Dito isso, vamos analisar as alternativas.

(a) Correto, pois a classe A não irá mais depender da classe B, ela irá usar a interface criada; (b) Errado, pois B não pode depender da classe A; (c) Errado, pois deve-se utilizar uma interface; (d) Errado, pois a classe B não pode depender da classe A.

Gabarito: Letra A

---

19. (FAURGS / TJ-RS – 2018) Em relação à refatoração, assinale com V (verdadeiro) ou F (falso) as afirmações abaixo.

( ) O melhor momento para se refatorar um código é durante os testes de aceitação, pois o cliente tem interesse em um código de qualidade.

( ) Um dos passos da refatoração é a aplicação dos testes que verificarão sua implementação.

( ) Rotinas muito longas e código duplicado são exemplos de bad smells.

( ) Refatorações são modificações no código que são simples a ponto de não gerarem nenhum efeito prático.

( ) Um código que já foi refatorado uma vez não precisará ser refatorado no futuro, pois já atende aos critérios de qualidade exigidos.

( ) A refatoração de um código implica apenas a melhoria de sua qualidade interna e não deve afetar sua funcionalidade original.

A sequência correta de preenchimento dos parênteses, de cima para baixo, é

a) F – V – F – V – F – V.

b) V – F – V – F – V – F.
c) F – V – V – F – F – V.
d) F – V – F – V – V – F.
e) V – F – V – F – V – V.

Comentários:

(F) Errado, não é feita durante os testes de aceitação, mas sim durante os testes unitários; (V) Correto, a refatoração por realizar mudanças na estrutura interna do código necessita de uma bons testes de implementação, principalmente os testes de unidade; (V) Correto, bas smell é uma situação no qual a estrutura do programa pode ser melhorada com refatoração. Ademais, rotinas longas e códigos duplicados devem ser melhorados com a refatoração; (F) Errado, nem sempre as refatorações são simples; (F) Errado, a refatoração deve ser feita sempre que necessário; (V) Correto, a refatoração não deve alterar o comportamento do software.

Gabarito: Letra C

---

20. (CEPS-UFPA / UFPA – 2018) Acerca do tema refatoração de software, considere as afirmativas.

I A refatoração busca evoluir o projeto e código-fonte de um sistema de software para se alcançar alta coesão, isto é, suas classes devem possuir conjuntos extensos de responsabilidades.

II A refatoração busca evoluir o projeto e código-fonte de um sistema de software para alcançar baixo acoplamento, isto é, a colaboração entre as classes deve ser mantida em um nível mínimo aceitável.

III A refatoração é o processo de mudar um sistema de software de tal forma que não altere o comportamento externo do código-fonte, embora melhore sua estrutura interna.

Está(ão) correta(s)

a) I, II e III.
b) I e II, somente.
c) I e III, somente.
d) III, somente.
e) II e III, somente.

Comentários:

(I) Errado, de fato, a refatoração busca que se tenha mais coesão no software, mas ela não busca a necessidade de se possuir conjuntos extensos de responsabilidade; (II) Correto, a refatoração busca baixo acoplamento e alta coesão; (III) Correto, é a definição perfeita de refatoração.

Gabarito: Letra E

---

21. (CESPE / STJ – 2018) A refatoração de um código escrito em Delphi pode levar um método a ser separado e transformado em alguns outros métodos.

Comentários:

Perfeito! A refatoração consiste na melhoria da estrutura interna do código-fonte de um programa, enquanto preserva seu comportamento externo. Essa melhoria na estrutura interna pode gerar a separação e a transformação de métodos.

Gabarito: Correto

---

22. (COPERVE-UFSC / UFSC – 2018) Considere os seguintes exemplos de procedimentos de manutenção, no contexto da necessidade de alteração de um programa hipotético de controle acadêmico de cursos de graduação da UFSC:

I. fazer com que o resultado da matrícula passe a ter a opção de gerar o resultado em formato PDF, além da atual possibilidade de informar na tela;

II. incluir funcionalidade para permitir que o trancamento de matrícula possa ser feito on-line;

III. reorganização da hierarquia de herança das classes do programa;

IV. criar classes no programa;

V. remover classes do programa;

Assinale a alternativa que relaciona apenas procedimentos de manutenção que podem ser classificados como ações de refatoração (refactoring).

a) III, IV e V.
b) I e II.
c) I, III e IV.
d) II, III e IV.
e) I, II e V.

Comentários:

Vamos relembrar o conceito de refatoração: a refatoração consiste na melhoria da estrutura interna do código-fonte de um programa, enquanto preserva seu comportamento externo.

(I) Errado, a refatoração não permite a incorporação de novas funcionalidades; (II) Errado, na refatoração não se pode incluir novas funcionalidades; (III) Correto, essa reorganização preserva o

comportamento externo do software; (IV) Correto, trata-se de uma reorganização interna, sem afetar o comportamento externo; (V) Correto, também se trata de uma reorganização interna.

Gabarito: Letra A

---

23. (CESPE / CGM-JP – 2018) A refatoração recomendada pela metodologia XP consiste na reorganização interna do código-fonte sem alteração no seu comportamento, o que permite melhorias no projeto, mesmo após o início da implementação.

Comentários:

Perfeito! A refatoração consiste na melhoria da estrutura interna do código-fonte de um programa, enquanto preserva seu comportamento externo.

Gabarito: Correto

---

24. (CESPE / DPE-RO – 2021) No contexto das metodologias ágeis, o conceito de refatoração compreende:

a) o desenvolvimento de testes incrementais a partir de novos cenários.
b) a renomeação de atributos e métodos para implementar melhorias no software.
c) a decomposição de histórias de usuário em uma série de tarefas de desenvolvimento.
d) a substituição do resultado de uma sprint inteira para atender a requisitos diferentes dos originais.
e) a junção de alterações de código às funcionalidades do software já entregue.

Comentários:

A refatoração é o processo de alteração de um sistema de software de tal forma que não se altere o comportamento externo do código, mas se aprimore a estrutura interna. A alternativa (e) pode gerar dúvida, mas na refatoração não há – de nenhuma forma – alterações nas funcionalidades do software. As demais alternativas não fazem sentido...

Gabarito: Letra B

---

25. (FCC / DPE-SP – 2010) A refatoração é o processo de modificar um sistema de software para melhorar a estrutura interna do código sem alterar seu comportamento externo.

Comentários:

Perfeito! Refatoração consiste na melhoria da estrutura interna do código-fonte de um programa, enquanto preserva seu comportamento externo.

Gabarito: Correto

---

26. (CESPE / CENSIPAM – 2006) A refatoração modifica a estrutura interna de um software visando facilitar o entendimento e as futuras modificações sem alterar o comportamento apresentado pelo software. Não é uma prática que possa ser aplicada em processos de desenvolvimento ágeis, pois requer a construção de modelos tanto para o projeto de alto nível quanto para o projeto detalhado.

Comentários:

A segunda sentença está incorreta – ele tanto é aplicado que é explicitamente referenciado pelo XP.

Gabarito: Errado

---

27. (CESPE / CENSIPAM – 2006) A refatoração é aplicável quando são identificados fragmentos de código que podem ser agrupados, expressões complicadas, atributos acessados mais por outras classes que pelas classes das quais são membros, enunciados condicionais complexos, códigos duplicados, longos métodos, longas classes, muitos parâmetros, métodos ou classes pouco usadas.

Comentários:

Perfeito! Entre os benefícios esperados, podemos afirmar que a refatoração melhora atributos objetos de código (tamanho, duplicação, acoplamento, coesão, complexidade ciclomática, entre outros) que estão relacionados com facilidade de manutenção. Além disso, ele ajuda a compreensão do código-fonte e encoraja o desenvolvedor a pensar sobre suas decisões de projeto

Gabarito: Correto

---

28. (CESPE / INMETRO – 2009) As técnicas de refatoração de código compreendem, entre outras, a remoção de números mágicos e a introdução de padrões de desenho.

Comentários:

Perfeito! A refatoração é a reorganização interna do código, logo bastante ligada aos padrões de projeto. Já os números mágicos são aqueles não tem um significado documentado e esclarecido no código e que, em geral, não estão atribuídos diretamente a uma variável. Eles foram ficando por conta de modificações no sistema e atualmente ninguém sabe o que eles significam.

Exemplo: Suponha que um programa de cálculo trigonométrico faça uso do número π em diversos lugares. A princípio o programador usou a aproximação 3.14 e a colocou numericamente em todos os lugares que ela era necessária. O número 3.14 a princípio é facilmente reconhecível como π por

qualquer pessoa com algum conhecimento de matemática. Porém nos testes o programador descobriu que precisaria de uma aproximação melhor, como 3.1415926. Agora ele tem que procurar todas as ocorrências de 3.14 no programa e substituí-la pela nova aproximação. Este procedimento é trabalhoso e sujeito a erros. Se o programador tivesse usado uma constante com o nome PI, em vez do número mágico 3.14, bastaria mudar a aproximação na definição da constante.

Gabarito: Correto

---

29. (CESPE / INMETRO – 2010) A técnica de refatoração, utilizada no paradigma de orientação a objetos, é mais bem enquadrada como uma técnica de reengenharia de software, isto é, que altera um software para reconstituí-lo em uma nova forma, função e implementação, que como uma técnica de engenharia reversa de software, isto é, uma técnica que analisa um software e cria novas representações abstratas do mesmo.

Comentários:

Trata-se do exato oposto: lembrem-se que a refatoração muda dentro sem mudar fora, logo não se encaixa como uma técnica de reengenharia, porque essa altera o software para reconstruí-lo de uma nova forma.

Gabarito: Errado

---

30. (CESPE / BASA – 2012) Denomina-se refatoração a atividade de reestruturação de programas, classes e métodos existentes para adaptá-los a alterações de funcionalidades e requisitos.

Comentários:

A questão está errada (mas a banca não entendeu dessa forma). Se houve alterações de funcionalidade, não é uma refatoração! Isso talvez seria uma reengenharia de software. Enfim, discordo...

Gabarito: Correto

---

31. (CESPE / BASA – 2012) A refatoração objetiva tornar o código mais claro e limpo.

Comentários:

Perfeito! Ela ajuda a compreensão do código-fonte e encoraja o desenvolvedor a pensar sobre suas decisões de projeto.

Gabarito: Correto

---

32. (CESPE / BASA – 2012) Ao refatorar um código, altera-se a funcionalidade do sistema.

Comentários:

Na verdade, a funcionalidade permanece a mesma.

Gabarito: Errado

---

33. **(CESPE / UNIPAMPA – 2013)** No que concerne às mudanças futuras, a refatoração de um programa orientado a objetos e a manutenção preventiva têm propostas opostas.

Comentários:

Não, têm propostas similares – ambas buscam melhorar a estrutura do código sem alterar seu comportamento externo.

Gabarito: Errado

---

34. **(CESPE / ANAC – 2009)** A técnica conhecida como refactoring é constantemente aplicada no desenvolvimento baseado no método ágil extreme programming.

Comentários:

Perfeito! Ela é realmente preconizada pelo XP para reorganizar e simplificar o projeto (ou código) de um componente de software sem modificar sua função ou seu comportamento.

Gabarito: Correto

---

35. **(CESPE / INMETRO – 2009)** A refabricação (ou refactoring) significa que primeiro deve ser desenvolvido um conjunto mínimo de funcionalidades que agreguem valor ao negócio e, depois, novas funcionalidades devem ser incrementadas ao produto já entregue.

Comentários:

Não faz o menor sentido. Nada de novas funcionalidades, é apenas otimização do comportamento interno.

Gabarito: Errado

---

36. **(CESPE / SAD-PE – 2010)** Em ferramentas CASE, como refactoring, é melhor adotar-se uma abordagem formal que uma abordagem heurística.

Comentários:

Bem... a despeito de a questão tratar *refactoring* como uma ferramenta e, não, como uma técnica, é melhor adotar uma abordagem heurística, devido a imensa variedade de algoritmos, entradas, saídas, etc. Além disso, é uma abordagem puramente empírica, baseada em fatos reais, na procura de possíveis melhorias.

Gabarito: Errado

---

37. (CESPE / STF – 2013) O refactoring aprimora o design de um software, reduz a complexidade da aplicação, remove redundâncias desnecessárias, reutiliza código, otimiza o desempenho e evita a deterioração durante o ciclo de vida de um código.

Comentários:

Perfeito! Ele melhora o design, reduz a complexidade, remove redundâncias, aumento reuso e otimiza o desempenho do software – isso auxilia a evitar a deterioração durante o ciclo de vida de um código.

Gabarito: Correto

---

38. (CESPE / TCU – 2015) A cada nova funcionalidade de software adicionada na prática de refactoring (refatoração) em XP, a chance, o desafio e a coragem de alterar o código-fonte de um software são aproveitados como oportunidade para que o design do software adote uma forma mais simples ou em harmonia com o ciclo de vida desse software, ainda que isso implique a alteração de um código com funcionamento correto.

Comentários:

A questão está completamente errada! Vimos insistentemente que a refatoração não adiciona novas funcionalidades. Quando vi que o gabarito preliminar veio como correto, achei um absurdo. Eu tinha certeza de que a banca mudaria o gabarito, mas ela não deu o braço a torcer e apenas anulou a questão. Menos mal...

Gabarito: Anulada

---

39. (CESPE / TRE-PE – 2017) Refactoring é o processo que:

a) implementa todas as funcionalidades da camada de model para depois implementar as camadas de controller e de viewer, nos casos em que a arquitetura MVC é utilizada.

b) efetua mudanças em um código existente e funcional sem alterar seu comportamento externo, com o objetivo de aprimorar a estrutura interna do código.

c) inclui funcionalidades extras no código, com o intuito de aprimorá-lo (rich source-code).

d) aprimora a extração e o refinamento iterativo dos requisitos do produto ainda na fase de planejamento do software, sendo considerado um valor na XP (extreme programming).

e) estabelece os métodos, um após o outro, para depois definir as classes e suas abstrações e implementar as interfaces.

Comentários:

Refatoração é o processo de mudar um sistema de software de tal forma que não altere o comportamento externo do código, embora melhore sua estrutura interna – nenhum dos outros itens faz qualquer sentido.

Gabarito: Letra B

---

40. (FCC / MPE-SE – 2009) Quanto à caracterização, a reengenharia de software é classificada como manutenção:

a) preventiva.
b) criptográfica.
c) de melhoria.
d) adaptativa.
e) corretiva.

Comentários:

A reengenharia de software é também conhecida como manutenção preventiva.

Gabarito: Letra A

---

41. (CESPE / PEFOCE – 2012) Como regra geral, não se deve tentar reestruturar um sistema com o uso da reengenharia se a abordagem inicial do sistema legado for funcional e a versão melhorada desejada for orientada a objetos.

Comentários:

O problema com a reengenharia de software é que existem limites práticos para o quanto você pode melhorar um sistema por meio da reengenharia. Não é possível, por exemplo, converter um sistema escrito por meio de uma abordagem funcional para um sistema orientado a objetos. As principais mudanças de arquitetura ou a reorganiza-ção radical do sistema de gerenciamento de dados não podem ser feitas automaticamente, pois são muito caras. Embora a reengenharia possa melhorara manutenibilidade, o sistema reconstruído provavelmente não será tão manutenível como um novo sistema, desenvolvido por meio de métodos modernos de engenharia de software.

Gabarito: Correto

---

42. (CESPE / PEFOCE – 2012) Reestruturação de software é uma atividade do processo de reengenharia de software voltada para a modificação da arquitetura global do programa, cujo objetivo consiste em tornar mais fácil o entendimento, os testes e a manutenção dos softwares.

Comentários:

Na verdade, não se modifica a arquitetura global. A reestruturação de software modifica o código-fonte e/ou dados para torná-lo mais amigável para futuras alterações. Lembrando que a reestruturação de software é composta pela reestruturação de código e a reestruturação de dados.

Gabarito: Errado

---

43. (CESPE / PC-ES – 2011) A reengenharia procura introduzir melhorias em processos já existentes, reformulando o que já existe ou fazendo pequenas mudanças que deixem as estruturas básicas intactas.

Comentários:

A reengenharia reconstitui o sistema em uma nova forma, logo não se trata de pequenas mudanças que deixem as estruturas básicas intactas.

Gabarito: Errado

---

44. (CESPE / TRE-BA – 2010) Das várias estratégias de mudança de software, realizar alterações significativas na arquitetura do sistema de software diz respeito a reengenharia de software.

Comentários:

Discordo do gabarito dessa questão! Para mim, ela está correta. Não vislumbro qualquer erro – caso alguém saiba, favor informar :)

Gabarito: Errado

LISTA DE QUESTÕES – DIVERSAS BANCAS

1. **(CESPE / SEFIN de Fortaleza-CE – 2023)** O processo de refatoração de software melhora o design do código existente, mas não substitui nem altera o seu comportamento.

2. **(CESPE / CAU-BR – 2024)** Um dos princípios da componentização de software é o OCP (open-closed principle), que define que um componente pode ser estendido sem necessidade de modificações internas.

3. **(FUNDATEC / BRDE - 2023)** Avalie as assertivas abaixo:

   I. Melhoria do design interno (arquitetura) do software.
   II. Código mais legível.
   III. Localização de bugs.
   IV. Mudança do comportamento externo do software.

   Quantas podem vir a ser benefícios do processo de refatoração de código?

   a) 0
   b) 1
   c) 2
   d) 3
   e) 4

4. **(AOCP / CODEBA – 2023)** No contexto do desenvolvimento de software, refatoração é uma técnica que visa melhorar a estrutura interna do código sem alterar seu comportamento externo. Assinale a alternativa correta sobre refatoração.

   a) Refatoração torna o código mais difícil de entender, aumentando a complexidade.
   b) Ao refatorar, os testes existentes são irrelevantes, pois a refatoração não afeta o comportamento do software.
   c) Refatoração pode ajudar a simplificar o código, tornando-o mais modular e reutilizável.
   d) Refatoração frequentemente resulta em uma completa reescrita do código, alterando seu comportamento externo.
   e) Code smells são indícios de que o código está bem otimizado e não requer refatoração.

5. **(CESPE / PETROBRÁS – 2022)** De acordo com o conceito de code smell, a categoria bloaters engloba os trechos de código que são irrelevantes e podem ser excluídos sem afetar a aplicação.

6. **(IDECAN / UFBA – 2022)** Além do teste de software, podem ser identificados problemas no código através de ferramentas de análise estática em busca de code smells.

7. **(QUADRIX / PRODAM-AM - 2022)** Assinale a alternativa que apresenta a prática de XP (Extreme Programming) que é definida como uma técnica disciplinada para reestruturar um corpo de código existente, alterando a sua estrutura interna sem alterar seu comportamento externo. Essa prática mantém a semântica do código, ou seja, após as mudanças, o código ainda funciona da mesma forma.

a) refatoração
b) metáfora
c) *test-driven development*
d) *coding standards*
e) *on-site customer*.

8. **(FUNDEP / UFJF - 2022)** Considere o trecho de código a seguir, que acabou de ser refatorado.

```
delta = b*b-4*a*c; // nova variável
x1 = (-b + sqrt(delta)) / (2*a);
x2 = (b + sqrt(delta)) / (2*a);
```

Assinale a refatoração aplicada para essa situação.

a) Inline de método.
b) Extração de método.
c) Extração de variável.
d) Renomeação.
e) Extração de classe.

9. **(CESPE / BANRISUL - 2022)** A filosofia da modelagem ágil não admite decisões que levem um projeto a sua rejeição e a sua refatoração, pois os profissionais de tecnologia não possuem todas as respostas e outros stakeholders envolvidos no negócio devem ser respeitados e integrados ao processo.

10. **(CESPE / MC - 2022)** Refatorar um *software* consiste em modificar o seu comportamento interno e externo, mantendo-se inalterada sua estrutura interna.

11. **(CONSULPAM / Prefeitura de Irauçuba - CE - 2022)** Dentro das metodologias ágeis, o processo de desenvolvimento de software especificado pela Programação Extrema (eXtreme Programming, XP) possui algumas características específicas. Uma das características do XP versa sobre as necessidades de melhoria no projeto, que devem ser realizadas através de um tipo de processo específico para este fim. Assinale a alternativa com o nome deste tipo de processo.

a) Testes.
b) Refatoração.
c) Histórias do Usuário.

d) Programação em Pares.

12. **(IDECAN / IF-CE - 2021)** No que diz respeito à manutenção e reengenharia de software, um termo define o processo de alterar o código-fonte, de modo que não altere o comportamento externo e ainda melhore a sua estrutura interna. É uma técnica disciplinada de limpar e organizar o código, e por consequência, minimizar a chance de introduzir novos bugs. Esse termo é conhecido como

a) elicitação.
b) refatoração.
c) recodificação.
d) replicação.

13. **(IBADE / Prefeitura de Vilhena - 2019)** Em Orientação a Objetos define-se o processo de Refatoração como:

a) processo de criação de entidades com herança e polimorfismo.
b) classe abstrata que permite a criação de métodos fatorados.
c) classe que permite herdar da classe pai todos os seus objetos.
d) processo para melhorar a legibilidade ou a eficiência do código, preservando seu comportamento.
e) objeto que permite acessar classes fatoradas.

14. **(CESPE / MPC-PA - 2019)** No contexto da manutenção preventiva de sistemas no paradigma orientado a objetos, a refatoração é uma técnica empregada com o objetivo de:

a) adicionar uma nova funcionalidade no sistema.
b) incluir uma generalidade que possa ser necessária no futuro.
c) realizar a reengenharia de um sistema.
d) reduzir o risco da introdução de novos erros no programa.
e) substituir métodos similares em subclasses por um único método em uma superclasse.

15. **(IF-PA / IF-PA - 2019)** Ao analisarmos uma classe Java, nos deparamos com um método que implementa diversas funcionalidades, tornando-se um método com muitas linhas de código, de difícil compreensão e manutenção. Para melhorar essa situação, decidimos dividi-lo em métodos menores, mais fáceis de entender e de efetuar manutenções. A esse processo de organizar e melhorar a estrutura interna de uma aplicação, denominamos de:

a) indentação.
b) depuração.
c) inspeção.
d) integração.
e) refatoração.

16. **(CESPE / SLU – 2019)** Refactoring (refatoração) é o processo utilizado para reescrever aplicações desatualizadas, com a finalidade de incrementar e melhorar suas funcionalidades; o uso dessa técnica normalmente aprimora aplicações para disponibilizá-las na Internet.

17. **(AOCP / PRODEB – 2018)** "Processo de alteração de um sistema de software de tal forma que não se altere o comportamento externo do código, mas se aprimore a estrutura interna". O enunciado se refere a:

a) Reúso.
b) Teste Unitário.
c) Teste de Integração.
d) Refatoração.
e) Prototipação.

18. **(CS-UFG / Câmara de Goiânia – 2018)** Sejam as classes A e B tais que o relacionamento entre elas é dado pelo fato de A usar (referenciar) a classe B. Dessa forma, qual das refatorações a seguir implementa o princípio da inversão de dependência?

a) Cria interface para serviços oferecidos por B; a classe A passa a usar a interface criada; a classe B passa a implementar a interface criada; a classe A não usa mais a classe B.

b) Cria interface para serviços oferecidos por A; a classe A passa a implementar a interface criada; a classe B passa a usar a interface criada; a classe A não usa mais a classe B.

c) Cria um relacionamento de herança entre as classes A e B (A torna-se uma especialização de B); métodos da classe B empregados pela classe A são migrados para a classe A; a classe A não usa mais a classe B.

d) Cria uma referência para a classe B na classe A; cria um método para receber uma instância de B (injeção de dependência) e guarda-a na referência criada; a classe A não usa mais a classe B.

19. **(FAURGS / TJ-RS – 2018)** Em relação à refatoração, assinale com V (verdadeiro) ou F (falso) as afirmações abaixo.

() O melhor momento para se refatorar um código é durante os testes de aceitação, pois o cliente tem interesse em um código de qualidade.

() Um dos passos da refatoração é a aplicação dos testes que verificarão sua implementação.

() Rotinas muito longas e código duplicado são exemplos de bad smells.

() Refatorações são modificações no código que são simples a ponto de não gerarem nenhum efeito prático.

() Um código que já foi refatorado uma vez não precisará ser refatorado no futuro, pois já atende aos critérios de qualidade exigidos.

() A refatoração de um código implica apenas a melhoria de sua qualidade interna e não deve afetar sua funcionalidade original.

A sequência correta de preenchimento dos parênteses, de cima para baixo, é

a) F – V – F – V – F – V.
b) V – F – V – F – V – F.
c) F – V – V – F – F – V.
d) F – V – F – V – V – F.
e) V – F – V – F – V – V.

20. (CEPS-UFPA / UFPA – 2018) Acerca do tema refatoração de software, considere as afirmativas.

I A refatoração busca evoluir o projeto e código-fonte de um sistema de software para se alcançar alta coesão, isto é, suas classes devem possuir conjuntos extensos de responsabilidades.

II A refatoração busca evoluir o projeto e código-fonte de um sistema de software para alcançar baixo acoplamento, isto é, a colaboração entre as classes deve ser mantida em um nível mínimo aceitável.

III A refatoração é o processo de mudar um sistema de software de tal forma que não altere o comportamento externo do código-fonte, embora melhore sua estrutura interna.

Está(ão) correta(s)

a) I, II e III.
b) I e II, somente.
c) I e III, somente.
d) III, somente.
e) II e III, somente.

21. (CESPE / STJ – 2018) A refatoração de um código escrito em Delphi pode levar um método a ser separado e transformado em alguns outros métodos.

22. (COPERVE-UFSC / UFSC – 2018) Considere os seguintes exemplos de procedimentos de manutenção, no contexto da necessidade de alteração de um programa hipotético de controle acadêmico de cursos de graduação da UFSC:

I. fazer com que o resultado da matrícula passe a ter a opção de gerar o resultado em formato PDF, além da atual possibilidade de informar na tela;

II. incluir funcionalidade para permitir que o trancamento de matrícula possa ser feito on-line;

III. reorganização da hierarquia de herança das classes do programa;

IV. criar classes no programa;

V. remover classes do programa;

Assinale a alternativa que relaciona apenas procedimentos de manutenção que podem ser classificados como ações de refatoração (refactoring).

a) III, IV e V.
b) I e II.
c) I, III e IV.
d) II, III e IV.
e) I, II e V.

23. **(CESPE / CGM-JP – 2018)** A refatoração recomendada pela metodologia XP consiste na reorganização interna do código-fonte sem alteração no seu comportamento, o que permite melhorias no projeto, mesmo após o início da implementação.

24. **(CESPE / DPE-RO – 2021)** No contexto das metodologias ágeis, o conceito de refatoração compreende:

a) o desenvolvimento de testes incrementais a partir de novos cenários.
b) a renomeação de atributos e métodos para implementar melhorias no software.
c) a decomposição de histórias de usuário em uma série de tarefas de desenvolvimento.
d) a substituição do resultado de uma sprint inteira para atender a requisitos diferentes dos originais.
e) a junção de alterações de código às funcionalidades do software já entregue.

25. **(FCC / DPE-SP – 2010)** A refatoração é o processo de modificar um sistema de software para melhorar a estrutura interna do código sem alterar seu comportamento externo.

26. **(CESPE / CENSIPAM – 2006)** A refatoração modifica a estrutura interna de um software visando facilitar o entendimento e as futuras modificações sem alterar o comportamento apresentado pelo software. Não é uma prática que possa ser aplicada em processos de desenvolvimento ágeis, pois requer a construção de modelos tanto para o projeto de alto nível quanto para o projeto detalhado.

27. **(CESPE / CENSIPAM – 2006)** A refatoração é aplicável quando são identificados fragmentos de código que podem ser agrupados, expressões complicadas, atributos acessados mais por outras classes que pelas classes das quais são membros, enunciados condicionais complexos, códigos duplicados, longos métodos, longas classes, muitos parâmetros, métodos ou classes pouco usadas.

28. **(CESPE / INMETRO – 2009)** As técnicas de refatoração de código compreendem, entre outras, a remoção de números mágicos e a introdução de padrões de desenho.

29. **(CESPE / INMETRO – 2010)** A técnica de refatoração, utilizada no paradigma de orientação a objetos, é mais bem enquadrada como uma técnica de reengenharia de software, isto é, que altera um software para reconstituí-lo em uma nova forma, função e implementação, que como uma técnica de engenharia reversa de software, isto é, uma técnica que analisa um software e cria novas representações abstratas do mesmo.

30. **(CESPE / BASA – 2012)** Denomina-se refatoração a atividade de reestruturação de programas, classes e métodos existentes para adaptá-los a alterações de funcionalidades e requisitos.

31. **(CESPE / BASA – 2012)** A refatoração objetiva tornar o código mais claro e limpo.

32. **(CESPE / BASA – 2012)** Ao refatorar um código, altera-se a funcionalidade do sistema.

33. **(CESPE / UNIPAMPA – 2013)** No que concerne às mudanças futuras, a refatoração de um programa orientado a objetos e a manutenção preventiva têm propostas opostas.

34. **(CESPE / ANAC – 2009)** A técnica conhecida como refactoring é constantemente aplicada no desenvolvimento baseado no método ágil extreme programming.

35. **(CESPE / INMETRO – 2009)** A refabricação (ou refactoring) significa que primeiro deve ser desenvolvido um conjunto mínimo de funcionalidades que agreguem valor ao negócio e, depois, novas funcionalidades devem ser incrementadas ao produto já entregue.

36. **(CESPE / SAD-PE – 2010)** Em ferramentas CASE, como refactoring, é melhor adotar-se uma abordagem formal que uma abordagem heurística.

37. **(CESPE / STF – 2013)** O refactoring aprimora o design de um software, reduz a complexidade da aplicação, remove redundâncias desnecessárias, reutiliza código, otimiza o desempenho e evita a deterioração durante o ciclo de vida de um código.

38. **(CESPE / TCU – 2015)** A cada nova funcionalidade de software adicionada na prática de refactoring (refatoração) em XP, a chance, o desafio e a coragem de alterar o código-fonte de um software são aproveitados como oportunidade para que o design do software adote uma forma mais simples ou em harmonia com o ciclo de vida desse software, ainda que isso implique a alteração de um código com funcionamento correto.

39. (CESPE / TRE-PE – 2017) Refactoring é o processo que:

a) implementa todas as funcionalidades da camada de model para depois implementar as camadas de controller e de viewer, nos casos em que a arquitetura MVC é utilizada.

b) efetua mudanças em um código existente e funcional sem alterar seu comportamento externo, com o objetivo de aprimorar a estrutura interna do código.

c) inclui funcionalidades extras no código, com o intuito de aprimorá-lo (rich source-code).

d) aprimora a extração e o refinamento iterativo dos requisitos do produto ainda na fase de planejamento do software, sendo considerado um valor na XP (extreme programming).

e) estabelece os métodos, um após o outro, para depois definir as classes e suas abstrações e implementar as interfaces.

40. (FCC / MPE-SE – 2009) Quanto à caracterização, a reengenharia de software é classificada como manutenção:

a) preventiva.
b) criptográfica.
c) de melhoria.
d) adaptativa.
e) corretiva.

41. (CESPE / PEFOCE – 2012) Como regra geral, não se deve tentar reestruturar um sistema com o uso da reengenharia se a abordagem inicial do sistema legado for funcional e a versão melhorada desejada for orientada a objetos.

42. (CESPE / PEFOCE – 2012) Reestruturação de software é uma atividade do processo de reengenharia de software voltada para a modificação da arquitetura global do programa, cujo objetivo consiste em tornar mais fácil o entendimento, os testes e a manutenção dos softwares.

43. (CESPE / PC-ES – 2011) A reengenharia procura introduzir melhorias em processos já existentes, reformulando o que já existe ou fazendo pequenas mudanças que deixem as estruturas básicas intactas.

44. (CESPE / TRE-BA – 2010) Das várias estratégias de mudança de software, realizar alterações significativas na arquitetura do sistema de software diz respeito a reengenharia de software.

GABARITO – DIVERSAS BANCAS

1. CORRETO
2. CORRETO
3. LETRA D
4. LETRA C
5. ERRADO
6. CORRETO
7. LETRA A
8. LETRA C
9. ERRADO
10. ERRADO
11. LETRA B
12. LETRA B
13. LETRA D
14. LETRA E
15. LETRA E
16. ERRADO
17. LETRA D
18. LETRA A
19. LETRA C
20. LETRA E
21. CORRETO
22. LETRA A
23. CORRETO
24. LETRA B
25. CORRETO
26. ERRADO
27. CORRETO
28. CORRETO
29. ERRADO
30. CORRETO
31. CORRETO
32. ERRADO
33. ERRAD
34. CORRETO
35. ERRADO
36. ERRADO
37. CORRETO
38. ANULADA
39. LETRA B
40. LETRA A
41. CORRETO
42. ERRADO
43. ERRADO
44. ERRADO

INTEGRAÇÃO, ENTREGA E IMPLANTAÇÃO CONTÍNUA

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

A integração contínua é um termo originado na metodologia ágil XP (Extreme Programming), apesar de ser utilizada em diversos outros contextos. Ela consiste em um processo bastante simples: o desenvolvedor integra o código alterado e/ou desenvolvido ao projeto principal na mesma frequência com que as funcionalidades são desenvolvidas, sendo feito idealmente muitas vezes ao dia ao invés de apenas uma vez.

O objetivo principal de utilizar a integração contínua é verificar se as alterações ou novas funcionalidades implementadas não criaram novos defeitos no projeto já existente. A prática da integração contínua pode ser feita através de processos manuais ou automatizados, utilizando ferramentas como o Jenkins, Hudson, entre outros. Um dos nossos maiores guias, Martin Fowler, já dizia sobre esse tema:

> *"A Integração Contínua se trata de uma prática de desenvolvimento de software em que os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por um build automatizado (incluindo testes) para detectar erros de integração o mais rápido possível. Muitos times acham que essa abordagem leva a uma significante redução nos problemas de integração e permite que um time desenvolva software coeso mais rapidamente".*

Basicamente, a grande vantagem da integração contínua está no feedback instantâneo. A prática da integração contínua se mostra muito eficaz nas equipes de desenvolvimento e está sendo amplamente utilizada. Inúmeros projetos *open-source* utilizam várias máquinas dedicadas a serem servidores de integração, rodando muitas vezes em softwares/plataformas diferentes. Ela é recomendada tanto com equipes distantes geograficamente quanto com equipes no mesmo local.

No primeiro caso, recomenda-se o uso de integração contínua assíncrona; e no segundo caso, recomenda-se o uso de integração contínua síncrona. **Em ambos os casos, é importante ter pelo menos uma máquina dedicada à integração que seja uma cópia idêntica do ambiente de produção, pois quanto mais rápido for o feedback melhor**. Dessa forma, a integração contínua busca alcançar alguns objetivos...

Primeiro, minimizar a duração e esforço requeridos por cada episódio de integração; e segundo, ser capaz de entregar uma versão do produto que possa ser lançada a qualquer momento. **É requerido um procedimento de integração que seja reproduzível no mínimo e, em grande parte, automatizável**. Essa automatização não visa apenas evitar trabalho repetitivo. Ela na verdade permite aos desenvolvedores alcançarem um nível muito maior de produtividade e qualidade.

Um problema recorrente no desenvolvimento de software é que os erros ocultos nos programas que desenvolvemos são cada vez mais caros e difíceis de corrigir à medida que o tempo avança. Qualquer pessoa que trabalhe com programação já deve ter passado pela situação de achar que um programa estava quase pronto e "só faltava testar". **Depois descobre-se que, na verdade, muita coisa ainda estava errada ou até mesmo faltando no código**.

Outra situação recorrente é a "preguiça" de testar o software, ou pelo menos a prática de postergar os testes. Sem testes e *deploys* automatizados, o desenvolvedor tende a escrever uma grande quantidade de código antes de realmente colocar o programa para executar, afinal se o processo é trabalhoso e toma muito tempo, ele vai evitar de fazer isso ao máximo. **Só que no final, gasta-se muito tempo para corrigir todos os "detalhes" que não estavam corretos**.

Para mitigar todos esses problemas, surgiu a Integração Contínua – justamente para automatizar o processo para que, com bastante frequência, uma ou mais vezes ao dia, seja possível integrar todas as alterações de todos os desenvolvedores envolvidos no projeto e realizar um teste geral. **Tem uma metáfora que eu gosto de usar bastante para explicar esse tema: um filme de ação**. Ora, todo filme é composto de várias cenas. Em nosso caso, uma aplicação é composta de vários builds.

Toda vez que eu adiciono uma nova cena em um filme, o editor faz um teste rodando todo o filme novamente para ver se o filme faz sentido com a nova cena. Caso o teste falhe, i.e., a nova cena não se encaixe na história, ela deve ser refeita! **Em nosso caso, dizemos que ele a build foi quebrada**. **Dessa maneira, em vez de construir vários componentes separadamente e depois juntá-los em um software final, nós vamos integrar continuamente**.

Novos builds só são integrados quando a build anterior for corrigida. Esses procedimentos auxiliam a reduzir riscos e a entregar soluções livres de defeitos.

AS PRÁTICAS...
- Mantenha um único repositório de código-fonte.
- Automatize um build.
- Faça um build auto-testável.
- Todo commit devem ser um build na máquina de integração.
- Mantenha os builds rápidos.
- Teste em uma cópia do ambiente de produção.
- Mantenha fácil que todos consigam o último executável.
- Todos consegue visualizar o processo.
- Automatização do deployment.

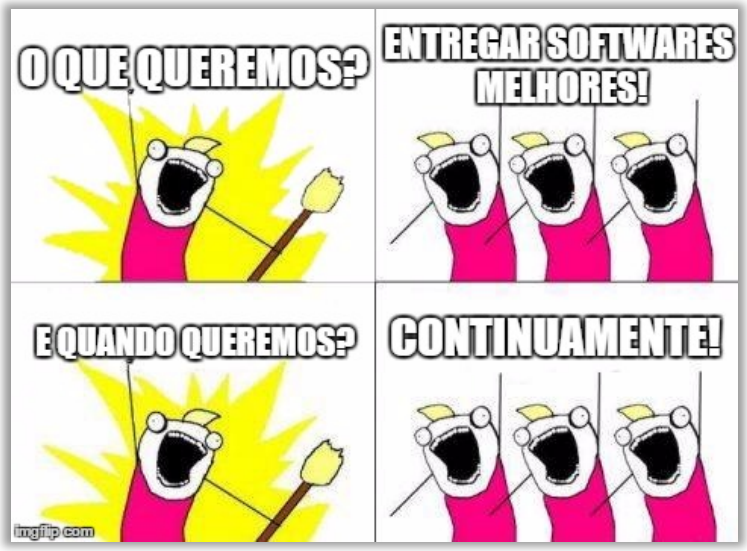
COMO FAZER...
- Desenvolvedores devem fazer checkout do código em seus workspaces privados.
- Quando finalizado, o commit modifica o repositório.
- O Servidor de Integração Contínua monitora o repositório e faz checkout das mudanças quando elas ocorrem.
- O Servidor de Integração Contínua constrói o sistema e roda testes de integração e testes de unidade.
- O Servidor de Integração Contínua lança artefatos implantáveis para testes.
- O Servidor de Integração Contínua atribui um rótulo de build para a versão do código que ele construiu.
- O Servidor de Integração Contínua informa ao time sobre o sucesso do build.
- Se a build ou teste falhar, o Servidor de Integração Contínua alerta a equipe.
- A equipe corrige o problema na melhor oportunidade.
- Continue a integrar continuamente e a testar durante o projeto.

Vamos falar um pouco sobre a entrega contínua. Uma equipe ágil geralmente entrega seu produto nas mãos dos usuários finais, ouvindo seus feedbacks (críticas e elogios). A frequência de entregas varia de acordo com aspectos técnicos e negociais do contexto que se encontra, mas se nós tivéssemos que chutar um número, diríamos que há uma entrega a cada quatro a seis iterações, no máximo.

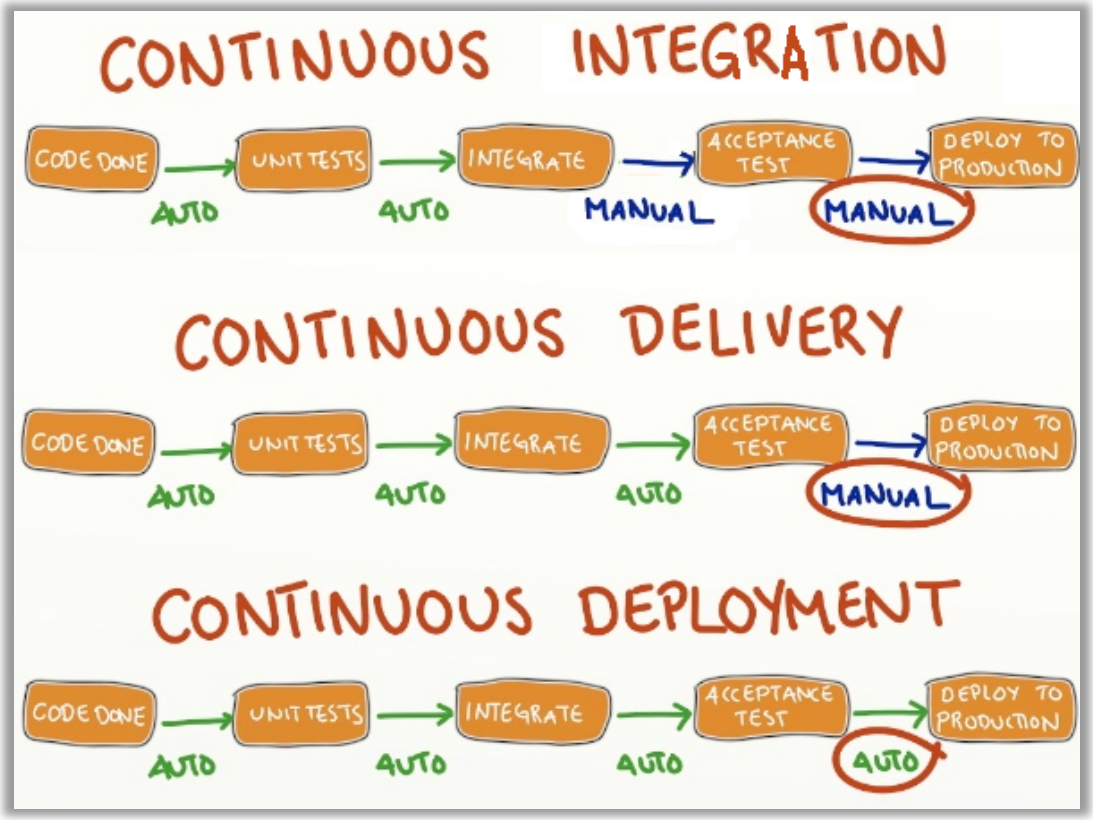
Em contextos técnicos favoráveis, como desenvolvimento web, um ritmo mais frequente de entregas pode ser alcançado (Ex: uma entrega a cada iteração). Já algumas equipes vão ao limite dessa prática por meio de *continous deployments*. **Apresentar a última versão do produto para o gerente de projetos/produtos para teste não é suficiente, nem entregar a uma equipe de garantia de qualidade**.

Uma entrega, em seu sentido primário, deve ser pelo menos uma versão beta avaliada por usuários representativos. Em alguns casos (como em softwares embarcados), não é possível organizar entregas contínuas para todos os usuários, mas isso não deve se tornar um pretexto para desistir das entregas contínuas mesmo que somente para alguns usuários. Entre os benefícios esperados, busca-se mitigar a falha de planejamento bastante conhecida de descobrir atrasos muito tarde.

Ela ajuda a validar o ajuste do produto ao mercado mais cedo, fornece feedbacks adiantados sobre a qualidade e estabilidade, e permite um retorno mais rápido do investimento sobre o produto.



Professor, qual a diferença entre Integração, Entrega e Implantação Contínua? **A primeira é o processo de testes e integração de um novo código com a base de código já existente – geralmente são feitos testes automatizados**. A segunda é o processo de garantir que o novo código pode ser implantado no ambiente de produção a qualquer momento – geralmente são feitos testes de comportamento.



Por fim, o terceiro trata do processo de publicação (*deploy*) no ambiente de produção, tão logo você esteja certo de que o código passou por todos os testes e está pronto para ser publicado. **Se o código já passou pelos testes automatizados e pelos testes manuais, visuais e de comportamento, então ele pode ser publicado de forma automatizada**. Observem a imagem anterior...

Quem ainda não entendeu, vai entender agora! **Imaginem que temos uma equipe de desenvolvimento formada por quatro pessoas trabalhando em um sistema**. Cada desenvolvedor está focado em implementar uma funcionalidade diferente. Caso o primeiro desenvolvedor termine sua parte na codificação, ele pode fazer um *commit* do código em uma ferramenta de controle de versão (Ex: Git).

A Integração Contínua é uma prática que pode ser aplicada por meio de uma ferramenta (Ex: Jenkins). Essa ferramenta pode ser configurada para, de tempos em tempos, buscar tudo que foi *comitado* pelos desenvolvedores e realizar testes de unidade e de integração automaticamente. **Caso algum problema seja encontrado, temos um feedback imediato**. Os desenvolvedores são notificados e ninguém faz check-in do repositório até que o problema seja resolvido.

Se os testes passarem, o primeiro desenvolvedor saberá que o código que ele implementou não quebrou o sistema. Então, percebam que a integração contínua consiste em, periodicamente, realizar testes automatizados e integrar novos códigos ao sistema (via de regra, no ambiente de desenvolvimento e, não, de produção). Já a diferença entre entrega contínua e implantação contínua é um pouco mais sutil...

A primeira é uma prática ágil na qual o software é construído de tal forma que ele pode ser colocado em produção a qualquer momento. A segunda é uma prática ágil na qual o software é construído de tal forma que ele é colocado em produção em determinado momento (isso é configurável). A grande diferença é por conta do último ponto no nosso desenho. A entrega contínua apresenta uma versão candidata para publicação após cada funcionalidade passar por todos os estágios.

Após isso, o negócio poderá decidir quando colocá-la efetivamente em produção. Já na implantação contínua, a entrada em produção ocorre automaticamente. *Do início, vamos andar pelo desenho?* Idealmente, o terceiro desenvolvedor pode terminar uma nova funcionalidade e realizar o *commit* do seu código para sua ferramenta de controle de versão. **Serão realizados testes de unidade e testes de integração automatizados**.

Passando por esses estágios, o código é integrado ao ambiente de desenvolvimento. Agora, todos os desenvolvedores poderão usufruir dessa nova funcionalidade implementada pelo terceiro desenvolvedor. Caso tenhamos também Entrega Contínua, o fluxo não para por aí! **De tempos em tempos, o sistema integrado passará por testes de aceitação automatizados**. Caso não haja erros, ele estará pronto para ser implantado em um ambiente de produção.

É comum que esse código fique em um ambiente de homologação, aguardando o negócio decidir quando ele deverá ser implantado, finalmente, no ambiente de produção. Caso tenhamos também a implantação contínua, o fluxo continua e, não mais será uma decisão do negócio a implantação ou não do software em produção – a implantação ocorrerá automaticamente. Fim ;)

QUESTÕES COMENTADAS – DIVERSAS BANCAS

1. **(CESPE / CAPES – 2024)** Um dos objetivos do processo de entrega contínua é garantir a continuidade da aplicação em produção, por meio de aumento do tempo entre os deploys, minimizando o seu impacto no que está funcionando.

Comentários:

No processo de entrega contínua (continuous delivery), um dos principais objetivos é garantir a continuidade e a confiabilidade da aplicação em produção através de deploys frequentes e automáticos, com mudanças pequenas e graduais. A ideia é reduzir o tempo entre os deploys e minimizar o impacto de cada mudança ao integrar e testar continuamente, assegurando que cada atualização cause o menor impacto possível no ambiente de produção.

Gabarito: Errado

---

2. **(CESPE / DATAPREV – 2023)** A implantação contínua se refere ao lançamento automático das mudanças feitas por um desenvolvedor, do repositório à produção, as quais podem ser utilizadas por clientes.

Comentários:

A implantação contínua (continuous deployment) se refere ao processo de lançamento automático de mudanças feitas por desenvolvedores, do repositório à produção, permitindo que essas alterações sejam imediatamente disponibilizadas e utilizadas pelos clientes. Este processo requer uma robusta automação de testes para garantir que apenas código de alta qualidade seja implantado.

Gabarito: Correto

---

3. **(CESPE / MPO – 2024)** O processo de CD apresenta como uma de suas vantagens à possibilidade de atualização anual concomitantemente à atualização do sistema operacional.

Comentários:

O processo de CD (Continuous Delivery ou Continuous Deployment) tem como principal vantagem a capacidade de lançar atualizações de software de forma contínua e automatizada, em curtos ciclos de tempo. Ele permite a liberação frequente de novas versões para produção, não se restringindo a ciclos de atualização anuais ou sincronizados com a atualização do sistema operacional. A ideia central é liberar software com maior rapidez e segurança, facilitando a adaptação rápida a mudanças.

Gabarito: Errado

---

4. **(CESPE / MPO – 2024)** Os recursos de monitoramento são de responsabilidade privativa, nessa abordagem, da equipe de operações no processo de CI/CD (continuous integration / continuous delivery).

Comentários:

No processo de CI/CD, os recursos de monitoramento não são exclusivamente de responsabilidade da equipe de operações. Devem ser compartilhados entre as equipes de desenvolvimento, operações e qualidade para garantir uma integração contínua e uma entrega contínua eficaz, promovendo a colaboração e a resposta rápida a problemas.

Gabarito: Errado

---

5. **(CESPE / MPE-GO – 2024)** Para uma integração contínua efetiva, um dos princípios recomendados é realizar o teste dos builds no ambiente de produção no qual o sistema final será executado.

Comentários:

Para uma integração contínua efetiva, os builds devem ser testados em um ambiente que imite o ambiente de produção, mas não no ambiente de produção real. Testar no ambiente de produção pode causar problemas se houver falhas ou bugs não detectados, além de possivelmente afetar a disponibilidade do serviço para os usuários finais. O ideal é usar um ambiente de staging que replique o ambiente de produção para realizar esses testes de forma segura.

Gabarito: Errado

---

6. **(FGV / Câmara de Fortaleza – 2024)** A automação de testes desempenha um papel fundamental em pipelines de Integração Contínua (CI) e Entrega Contínua (CD), ao permitir que equipes de desenvolvimento detectem e corrijam problemas rapidamente, mantendo a qualidade do software em níveis adequados. Uma característica importante da automação de testes CI/CD é a capacidade de executar testes automaticamente a cada novo commit no repositório, garantindo que alterações recentes não quebrem funcionalidades existentes.

   Com base nesse contexto, assinale a opção que descreve corretamente a importância da automação de testes em ambientes CI/CD.

   a) A automação de testes em CI/CD é opcional, visto que testes manuais podem ser eficientemente agendados para cobrir todas as mudanças de código.

   b) A principal vantagem da automação de testes em CI/CD é reduzir o custo dos testes ao eliminar a necessidade de ferramentas de automação.

c) A automação de testes em CI/CD permite a execução frequente de testes, ajudando a identificar e corrigir defeitos rapidamente, o que é essencial para manter a qualidade do software.

d) Em ambientes de CI/CD, a automação de testes é usada apenas para testes de performance e carga, enquanto testes funcionais devem ser realizados manualmente.

e) A automação de testes em CI/CD foca exclusivamente em testes de segurança, deixando outros tipos de testes para serem executados em ambientes de teste dedicados.

Comentários:

(a) Errado. A automação de testes em CI/CD não é opcional, pois permite a detecção rápida de defeitos em mudanças de código frequentes, algo que testes manuais não conseguem fazer de maneira eficiente e contínua;

(b) Errado. A automação de testes em CI/CD não visa eliminar a necessidade de ferramentas de automação, mas sim aumentar a eficiência e frequência dos testes, ajudando a detectar defeitos rapidamente;

(c) Correto. A automação de testes em CI/CD permite a execução frequente de testes a cada nova alteração de código, facilitando a identificação e correção de defeitos e mantendo a qualidade do software;

(d) Errado. A automação de testes em CI/CD é usada para diversos tipos de testes, incluindo funcionais, unitários, integração e não apenas para testes de performance e carga;

(e) Errado. A automação de testes em CI/CD abrange uma ampla gama de testes, como funcionais, integração, e não é focada exclusivamente em testes de segurança.

Gabarito: Letra C

---

7. **(FGV / TJ-RJ – 2024)** O Desenvolvimento de Software tem sido impactado pela condução de projetos empregando métodos ágeis. São alguns dos principais métodos: Scrum; Kanban; Lean; Feature Driven Development (FDD); eXtreme Programming (XP); e, Microsoft Solutions Framework (MSF).

Entre as técnicas aplicadas, destaca-se a prática que torna a integração de código mais eficiente por meio de builds e testes automatizados denominada:

a) Iterative Development / Continuous Delivery (ID/CD)
b) Continuous Delivery / Continuous Feedback (CD/CF)
c) Continuous Integration / Continuous Delivery (CI/CD)

d) Flow Production / Continuous Integration (FP/CI)
e) Continuous Integration / Continuous Feedback (CI/CF)

Comentários:

(a) Errado. Iterative Development não é uma prática específica de integração e entrega contínua de código;

(b) Errado. Continuous Feedback é importante, mas não é a prática específica para integração e entrega contínua de código;

(c) Correto. Continuous Integration (CI) e Continuous Delivery (CD) são práticas que tornam a integração de código mais eficiente por meio de builds e testes automatizados;

(d) Errado. Flow Production não está relacionado diretamente com práticas de integração e entrega contínua de código;

(e) Errado. Continuous Feedback não é a prática específica para integração e entrega contínua de código;

Gabarito: Letra C

---

8. **(CESPE / DATAPREV – 2023)** O processo de entrega contínua envolve a automação de processos de desenvolvimento, testes e integração dos códigos de forma frequente e em um repositório compartilhado.

Comentários:

Opa! Essa é a definição de Integração Contínua e, não, Entrega Contínua.

Gabarito: Errado

---

9. **(CESPE / SEFIN de Fortaleza-CE – 2023)** Embora as técnicas de integração contínua sejam essenciais para garantir a qualidade do software e a eficiência do processo de desenvolvimento, elas não possibilitam a realização de testes automatizados de aceitação.

Comentários:

As técnicas de integração contínua (CI) incluem a realização de testes automatizados, inclusive testes de aceitação. Na CI, o código é frequentemente integrado e testado automaticamente, o que pode incluir uma ampla gama de testes: unitários, de integração, de aceitação, entre outros. Esses testes ajudam a garantir que o software funcione conforme esperado em cada etapa do desenvolvimento, contribuindo para a qualidade e a eficiência do processo.

Gabarito: Errado

---

10. (CESGRANRIO / Petrobrás– 2010) É comum, na Engenharia de Software, o uso de ferramentas de software que auxiliam na realização de diversas atividades do desenvolvimento. Nesse contexto, ferramentas de integração contínua são destinadas a automatizar a implantação do produto de software no ambiente de produção.

Comentários:

Não se trata de automatizar a implantação, mas de oferecer feedbacks tempestivos por meio da integração a todo momento. Automatizar a implantação é apenas um detalhe!

Gabarito: Errado

---

11. (CESPE / TCE-PE – 2004) A prática de integração contínua depende fortemente do uso de ferramentas de build e controle de versão.

Comentários:

Perfeito! Necessita de ferramentas de build e controle de versão. Cada integração é verificada por um build automatizado (incluindo testes) para detectar erros de integração o mais rápido possível.

Gabarito: Correto

---

12. (FCC / BACEN – 2006) A técnica de *Continuous Integration* diz que o código desenvolvido por cada par de desenvolvedores deve ser integrado ao código base constantemente. Quanto menor o intervalo entre cada integração, menor a diferença entre os códigos desenvolvidos e maior a probabilidade de identificação de erros, pois cada vez que o código é integrado, todos os *unit tests* devem ser executados, e, se algum deles falhar, é porque o código recém integrado foi o responsável por inserir erro no sistema.

Comentários:

Perfeito! O desenvolvedor integra o código alterado e/ou desenvolvido ao projeto principal na mesma frequência com que as funcionalidades são desenvolvidas, sendo feito idealmente muitas vezes ao dia ao invés de apenas uma vez.

Gabarito: Correto

---

13. (CESPE / TCU – 2015) Para que a prática de integração contínua seja eficiente, é necessário parametrizar e automatizar várias atividades relativas à gerência da configuração, não somente do código-fonte produzido, mas também de bibliotecas e componentes externos.

Comentários:

Perfeito! Parametrizam-se e automatizam-se componentes como código-fonte, bibliotecas, scripts de build, etc – ajustando a dependência entre eles.

Gabarito: Correto

LISTA DE QUESTÕES – DIVERSAS BANCAS

1. **(CESPE / CAPES – 2024)** Um dos objetivos do processo de entrega contínua é garantir a continuidade da aplicação em produção, por meio de aumento do tempo entre os deploys, minimizando o seu impacto no que está funcionando.

2. **(CESPE / DATAPREV – 2023)** A implantação contínua se refere ao lançamento automático das mudanças feitas por um desenvolvedor, do repositório à produção, as quais podem ser utilizadas por clientes.

3. **(CESPE / MPO – 2024)** O processo de CD apresenta como uma de suas vantagens à possibilidade de atualização anual concomitantemente à atualização do sistema operacional.

4. **(CESPE / MPO – 2024)** Os recursos de monitoramento são de responsabilidade privativa, nessa abordagem, da equipe de operações no processo de CI/CD (continuous integration / continuous delivery).

5. **(CESPE / MPE-GO – 2024)** Para uma integração contínua efetiva, um dos princípios recomendados é realizar o teste dos builds no ambiente de produção no qual o sistema final será executado.

6. **(FGV / Câmara de Fortaleza – 2024)** A automação de testes desempenha um papel fundamental em pipelines de Integração Contínua (CI) e Entrega Contínua (CD), ao permitir que equipes de desenvolvimento detectem e corrijam problemas rapidamente, mantendo a qualidade do software em níveis adequados. Uma característica importante da automação de testes CI/CD é a capacidade de executar testes automaticamente a cada novo commit no repositório, garantindo que alterações recentes não quebrem funcionalidades existentes.

   Com base nesse contexto, assinale a opção que descreve corretamente a importância da automação de testes em ambientes CI/CD.

   a) A automação de testes em CI/CD é opcional, visto que testes manuais podem ser eficientemente agendados para cobrir todas as mudanças de código.

   b) A principal vantagem da automação de testes em CI/CD é reduzir o custo dos testes ao eliminar a necessidade de ferramentas de automação.

   c) A automação de testes em CI/CD permite a execução frequente de testes, ajudando a identificar e corrigir defeitos rapidamente, o que é essencial para manter a qualidade do software.

   d) Em ambientes de CI/CD, a automação de testes é usada apenas para testes de performance e carga, enquanto testes funcionais devem ser realizados manualmente.

e) A automação de testes em CI/CD foca exclusivamente em testes de segurança, deixando outros tipos de testes para serem executados em ambientes de teste dedicados.

7. **(FGV / TJ-RJ – 2024)** O Desenvolvimento de Software tem sido impactado pela condução de projetos empregando métodos ágeis. São alguns dos principais métodos: Scrum; Kanban; Lean; Feature Driven Development (FDD); eXtreme Programming (XP); e, Microsoft Solutions Framework (MSF).

   Entre as técnicas aplicadas, destaca-se a prática que torna a integração de código mais eficiente por meio de builds e testes automatizados denominada:

   a) Iterative Development / Continuous Delivery (ID/CD)
   b) Continuous Delivery / Continuous Feedback (CD/CF)
   c) Continuous Integration / Continuous Delivery (CI/CD)
   d) Flow Production / Continuous Integration (FP/CI)
   e) Continuous Integration / Continuous Feedback (CI/CF)

8. **(CESPE / DATAPREV – 2023)** O processo de entrega contínua envolve a automação de processos de desenvolvimento, testes e integração dos códigos de forma frequente e em um repositório compartilhado.

9. **(CESPE / SEFIN de Fortaleza-CE – 2023)** Embora as técnicas de integração contínua sejam essenciais para garantir a qualidade do software e a eficiência do processo de desenvolvimento, elas não possibilitam a realização de testes automatizados de aceitação.

10. **(CESGRANRIO / Petrobrás– 2010)** É comum, na Engenharia de Software, o uso de ferramentas de software que auxiliam na realização de diversas atividades do desenvolvimento. Nesse contexto, ferramentas de integração contínua são destinadas a automatizar a implantação do produto de software no ambiente de produção.

11. **(CESPE / TCE-PE – 2004)** A prática de integração contínua depende fortemente do uso de ferramentas de build e controle de versão.

12. **(FCC / BACEN – 2006)** A técnica de *Continuous Integration* diz que o código desenvolvido por cada par de desenvolvedores deve ser integrado ao código base constantemente. Quanto menor o intervalo entre cada integração, menor a diferença entre os códigos desenvolvidos e maior a probabilidade de identificação de erros, pois cada vez que o código é integrado, todos os *unit tests* devem ser executados, e, se algum deles falhar, é porque o código recém integrado foi o responsável por inserir erro no sistema.

13. **(CESPE / TCU – 2015)** Para que a prática de integração contínua seja eficiente, é necessário parametrizar e automatizar várias atividades relativas à gerência da configuração, não somente do código-fonte produzido, mas também de bibliotecas e componentes externos.

GABARITO – DIVERSAS BANCAS

1. ERRADO
2. CORRETO
3. ERRADO
4. ERRADO
5. ERRADO
6. LETRA C
7. LETRA C
8. ERRADO
9. ERRADO
10. ERRADO
11. CORRETO
12. CORRETO
13. CORRETO

MODELAGEM ÁGIL

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

A modelagem ágil é uma metodologia baseada em práticas para modelagem e documentação de sistemas baseados em software. De modo geral, é um conjunto de melhores práticas. De modo específico é um conjunto de valores, princípios e práticas para modelagem de software que pode combinada com outras metodologias. Os valores da modelagem ágil estendem os valores do XP, sendo comunicação, simplicidade, feedback, coragem e adiciona: **humildade**.

Admitir que você não sabe de tudo e que os outros têm valor a adicionar ao projeto. **As chaves para o sucesso são uma comunicação eficaz entre os participantes do projeto no esforço para desenvolver a solução mais simples possível.** Possui basicamente três objetivos: (1) definir como colocar em prática os valores, princípios e práticas relativas a uma modelagem eficaz e leve; (2) discutir como aplicar técnicas de modelagem de software adotando uma perspectiva ágil;

(3) e discutir como melhorar suas atividades adotando uma perspectiva quase ágil para desenvolvimento de software e equipes de projeto que adotaram Processo Unificado. **Um conceito importante para compreender cerca de modelagem ágil é de que não é um processo completo de software.** Ele não inclui as atividades de programação e não inclui as atividades de teste. *Bacana?*

Não abrange gerenciamento de projetos, implantação do sistema, operações do sistema, suporte do sistema, ou várias outras questões. **Para projetos XP, a modelagem ágil descreve explicitamente como melhorar a produtividade através da adição de atividades de modelagem**, enquanto que – para projetos RUP – ela descreve como agilizar modelagem e documentação para melhorar a produtividade.

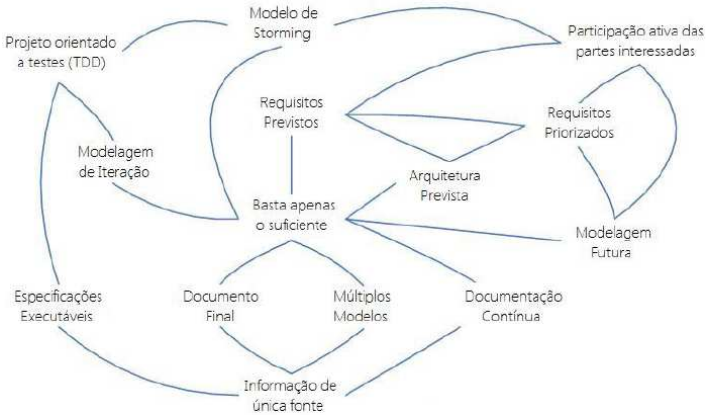
Pessoal, é claro que o XP requer menos modelagem, mas em nenhum momento ele a descarta. A modelagem ágil **não é um processo de desenvolvimento prescritivo**, portanto ela não define procedimentos detalhados de como criar um tipo de modelo, mas ela define sugestões sobre como realizar uma modelagem efetiva e ser um modelador eficiente. Ela mistura o caos de práticas simples de modelagem com a ordem inerente a artefatos de modelagem de software.

Entre os benefícios da modelagem ágil, temos: relação custo-benefício, que aumenta à medida que maximiza o retorno sobre investimento das partes interessadas; definição de técnicas explícitas para projetos ágeis, em que são apresentadas sugestões para utilização com Extreme Programming (XP), Rational Unified Process (RUP), entre outros; modelagem ágil é compreensível para o público ao qual se reportam.

Além disso, existe melhoria da documentação e modelagem de processos prescritivos, também específicos para cada modelo de processo. Entretanto, a modelagem ágil tem dificuldades de ser aplicada por equipes grandes (exemplo: mais de 30 pessoas) sem suporte de ferramentas adequadas. Eles são suficientemente precisos, detalhados e consistentes e agregam valor ao projeto. *Entendido?*

Craig Larman diz que a finalidade da modelagem é principalmente entender, não documentar. Sabe-se que a adoção de um método ágil não evita a modelagem, pelo contrário. Muito métodos ágeis normalmente incluem significativas sessões de modelagem. A modelagem ágil recomenda aplicação da Unified Modeling Language (UML) em partes incomuns e complexas do projeto e, não, em todo projeto.

Por fim, deve-se ter em mente que a finalidade primária da modelagem é apoiar o entendimento e a comunicação e não documentar o processo de desenvolvimento de software. Recomenda-se utilizar a ferramenta mais simples possível, incentivando a criatividade e apoiando a compreensão e a modificação rápidas. A Modelagem Ágil apresenta uma série de melhores práticas apresentadas a seguir:



PRÁTICA	DESCRIÇÃO
ARQUITETURA DE PREVISÃO	No início de um projeto ágil, a equipe vai precisar fazer alguns trabalhos como modelo arquitetônico inicial, de alto nível para identificar uma estratégia técnica viável para sua solução.
MODELAGEM DE ITERAÇÃO	No início de cada iteração, você vai fazer um pouco de modelagem, como parte de suas atividades de planejamento de iteração.
BASTA APENAS O SUFICIENTE	Um modelo ou documento deve ser suficiente para a situação na mão e nada mais.
MODELAGEM FUTURA	Às vezes, é necessária para reduzir o risco global.
MODELO DE STORMING	Ao longo da iteração, modela-se por alguns minutos para explorar os detalhes por trás de um requisito ou para pensar em um problema de design.

MÚLTIPLOS MODELOS	Cada tipo de modelo tem seus pontos fortes e fracos. Um desenvolvedor eficaz terá uma gama de modelos que lhes permita aplicar o modelo certo de maneira mais apropriada para a situação na mão.
REQUISITOS PRIORIZADOS	Equipes ágeis implementam os requisitos em ordem de prioridade, conforme definido por suas partes interessadas, de modo a proporcionar o maior retorno sobre investimento possível.
REQUISITOS PREVISTOS	No início de um projeto ágil, há de se investir algum tempo para identificar o escopo do projeto e criar a pilha de prioridade inicial de requisitos.
PARTICIPAÇÃO ATIVA DAS PARTES INTERESSADAS	Os interessados deverão fornecer informações em tempo hábil, tomar decisões em tempo hábil e ser tão ativamente envolvidos no processo de desenvolvimento através do uso de ferramentas e técnicas, inclusive.

Em suma: **modelagem ágil é uma metodologia prática** e, não, teórica, apoiada por valores e princípios, não prescritiva, com foco apenas no que é suficiente, que deve ser utilizada em conjunto com outras metodologias, estimulada por um trabalho em equipe. Scott W. Ambler diz que é mais uma arte que uma ciência; que recomenda a criação de um modelo correto que faça sentido na situação. É isso! :-)

(CODENI-RJ – 2010) Trata-se de um modo comum de aplicar a UML, frequentemente com alto retorno no investimento de tempo. Essa definição refere-se a:

a) Case
b) Objetos Syntropy
c) Modelagem Ágil
d) MDA

Comentários: Craig Larman diz: "*Modelagem ágil enfatiza a UML como rascunho; trata-se de um modo comum de aplicar a UML, frequentemente com algo retorno no investimento de tempo (que é tipicamente curto). As ferramentas UML podem ser úteis, mas eu incentivo as pessoas a também considerar uma abordagem ágil de modelagem para aplicar a UML*" (Letra C).

(SERPRO – 2013) A modelagem ágil é uma metodologia que apresenta ciclo de vida e processo que podem ser claramente seguidos pelo operador.

Comentários: a modelagem ágil não é um processo de desenvolvimento prescritivo, portanto ela não define procedimentos detalhados de como criar um tipo de modelo, mas ela define sugestões sobre como realizar uma modelagem efetiva. Define-se o que fazer e, não, como fazer (Letra E).

(CGM-JP – 2018) Na modelagem tradicional, parte significativa do tempo da equipe de desenvolvimento e dos recursos é despendida com manutenção e ajustes de modelos e diagramas; na modelagem ágil, por princípio, mudanças de requisitos ocorridas ao longo do processo de desenvolvimento ensejarão o descarte dos modelos e diagramas que não forneçam valor em longo prazo, ainda que sejam relacionados aos requisitos afetados.

Comentários: a modelagem tradicional despende uma boa quantidade de tempo com manutenção e ajustes de modelos e diagramas (documentação); já na modelagem ágil, mudanças de requisitos podem sim ensejar o descarte de modelos que não agreguem valor a longo prazo ou que não sejam mais úteis (Correto).

(BANRISUL – 2022) A filosofia da modelagem ágil não admite decisões que levem um projeto a sua rejeição e a sua refatoração, pois os profissionais de tecnologia não possuem todas as respostas e outros stakeholders envolvidos no negócio devem ser respeitados e integrados ao processo.

_____

Comentários: pelo contrário, a filosofia da modelagem ágil admite - sim - decisões que levem um projeto a sua rejeição e a sua refatoração. Ela é baseada na agilidade e flexibilidade. Por exemplo: se for identificado que um projeto não atenderá às necessidades do cliente, a equipe pode optar por rejeitá-lo e refatorá-lo para atender melhor a essas necessidades (Errado).

QUESTÕES COMENTADAS – DIVERSAS BANCAS

1. **(MS-CONCURSOS / CODENI-RJ – 2010)** Trata-se de um modo comum de aplicar a UML, frequentemente com alto retorno no investimento de tempo. Essa definição refere-se a:

a) Case
b) Objetos Syntropy
c) Modelagem Ágil
d) MDA

Comentários:

(a) Errado. CASE (Computer-Aided Software Engineering) refere-se ao uso de software para ajudar no desenvolvimento de software, mas não especificamente à aplicação da UML com foco em alto retorno do investimento de tempo;

(b) Errado. Objeto Syntropy é um método de desenvolvimento de software que usa modelagem, mas a definição não se alinha especificamente com o conceito de alto retorno no investimento de tempo ao aplicar a UML;

(c) Correto. Trata-se de uma abordagem que aplica princípios ágeis ao processo de modelagem, incluindo o uso de UML. Pode ser associada a um alto retorno no investimento de tempo devido à sua natureza iterativa e adaptativa, focando em entregar valor rapidamente. No entanto, a definição da questão não se alinha perfeitamente apenas com a modelagem ágil, pois várias metodologias podem buscar alto retorno no investimento de tempo;

(d) Errado. MDA (Model-Driven Architecture) é uma abordagem para o desenvolvimento de software que se baseia fortemente na modelagem e na transformação de modelos. MDA utiliza UML como parte de seu processo de modelagem para definir os requisitos e a estrutura do software de maneira abstrata antes de se mover para a implementação. O foco está na abstração e na automação para gerar código, o que pode levar a um alto retorno no investimento de tempo. Porém, a definição fornecida na questão não é exclusiva para descrever MDA.

Craig Larman diz: "*Modelagem ágil enfatiza a UML como rascunho; trata-se de um modo comum de aplicar a UML, frequentemente com algo retorno no investimento de tempo (que é tipicamente curto). As ferramentas UML podem ser úteis, mas eu incentivo as pessoas a também considerar uma abordagem ágil de modelagem para aplicar a UML*".

Gabarito: Letra C

---

2. **(CESPE / SERPRO – 2013)** A modelagem ágil é uma metodologia que apresenta ciclo de vida e processo que podem ser claramente seguidos pelo operador.

Comentários:

A modelagem ágil não é um processo de desenvolvimento prescritivo, portanto ela não define procedimentos detalhados de como criar um tipo de modelo, mas ela define sugestões sobre como realizar uma modelagem efetiva. Define-se o que fazer e, não, como fazer.

Gabarito: Errado

---

3. **(CESPE / CGM-JP – 2018)** Na modelagem tradicional, parte significativa do tempo da equipe de desenvolvimento e dos recursos é despendida com manutenção e ajustes de modelos e diagramas; na modelagem ágil, por princípio, mudanças de requisitos ocorridas ao longo do processo de desenvolvimento ensejarão o descarte dos modelos e diagramas que não forneçam valor em longo prazo, ainda que sejam relacionados aos requisitos afetados.

Comentários:

Na modelagem tradicional, uma quantidade significativa de esforço é, de fato, dedicada à manutenção e ajustes de modelos e diagramas para garantir que eles reflitam com precisão os requisitos do sistema ao longo do tempo. Isso pode ser devido ao rigoroso processo de documentação e à natureza menos flexível dessa abordagem, onde as mudanças são mais custosas e demoradas para serem implementadas.

Por outro lado, a modelagem ágil adota uma filosofia de adaptabilidade e flexibilidade. Mudanças nos requisitos são vistas como uma parte natural do desenvolvimento de software, e a metodologia ágil encoraja a revisão contínua dos modelos e diagramas para garantir que eles continuem a fornecer valor. Modelos e diagramas que não fornecem valor em longo prazo ou que se tornam irrelevantes devido a mudanças nos requisitos podem ser descartados. Isso reflete o princípio ágil de valorizar "software em funcionamento mais que documentação abrangente", conforme expresso no Manifesto Ágil.

A modelagem tradicional despende uma boa quantidade de tempo com manutenção e ajustes de modelos e diagramas (documentação); já na modelagem ágil, mudanças de requisitos podem sim ensejar o descarte de modelos que não agreguem valor a longo prazo ou que não sejam mais úteis.

Gabarito: Correto

---

4. **(CESPE / BANRISUL – 2022)** A filosofia da modelagem ágil não admite decisões que levem um projeto a sua rejeição e a sua refatoração, pois os profissionais de tecnologia não possuem todas as respostas e outros stakeholders envolvidos no negócio devem ser respeitados e integrados ao processo.

Comentários:

A modelagem ágil, de fato, enfatiza a importância da colaboração entre todos os stakeholders do projeto, incluindo os profissionais de tecnologia, clientes e usuários finais. Ela reconhece que os profissionais de tecnologia não possuem todas as respostas e que o envolvimento contínuo de outros stakeholders é crucial para o sucesso do projeto. Isso é verdade e está em linha com os princípios do Manifesto Ágil, que valoriza a colaboração com o cliente mais do que a negociação de contratos.

No entanto, a primeira parte da afirmação sugere que a modelagem ágil não admite decisões que levem à rejeição ou refatoração de um projeto, o que não é correto. Na verdade, a adaptabilidade é um dos pilares da modelagem ágil. A abordagem ágil é flexível e permite, e muitas vezes encoraja, a refatoração como uma forma de melhorar e adaptar o software às necessidades em mudança.

Se durante o processo de desenvolvimento ágil, for identificado que certas partes do projeto não estão agregando valor ou não estão atendendo às necessidades do negócio como esperado, a refatoração ou até a rejeição dessas partes pode ser considerada uma prática válida e alinhada com os princípios ágeis.

A filosofia ágil enfatiza a entrega iterativa de valor, o aprendizado contínuo e a capacidade de responder a mudanças rapidamente, o que inclui fazer ajustes significativos no projeto, se necessário, para melhor atender aos objetivos do negócio.

Gabarito: Errado

---

5. **(UFMT / UFMT – 2016 – Letra A)** A modelagem Ágil tem foco na codificação de artefatos de software e, por este motivo, dispensa completamente a documentação de sistema.

Comentários:

O Manifesto Ágil, que serve como fundamento para as metodologias ágeis, afirma: "Valorizamos mais software em funcionamento do que documentação abrangente". Isso não significa que a documentação seja completamente dispensada, mas sim que o foco está em criar software funcional que atenda às necessidades dos usuários.

A documentação ainda tem seu lugar no desenvolvimento ágil, mas é tratada de maneira pragmática. A documentação necessária é produzida, preferencialmente de maneira enxuta e suficiente para o propósito, sem excessos que não agreguem valor ao cliente ou ao processo de desenvolvimento.

Logo, enquanto a modelagem ágil prioriza a entrega de software funcional, ela não elimina a necessidade de documentação. Em vez disso, incentiva uma abordagem mais balanceada, onde a documentação é criada conforme necessário, de forma a apoiar o desenvolvimento do projeto sem sobrecarregá-lo com artefatos desnecessários.

Gabarito: Errado

---

6. **(UFMG / UFMG – 2016 – Letra C)** A Modelagem Ágil (AM) sugere que vários modelos, como os modelos da UML, modelos de dados, protótipos de interface do usuário e modelos navegacionais, dentre outros, sejam construídos e utilizados em paralelo.

Comentários:

A Modelagem Ágil (Agile Modeling - AM) é uma metodologia que complementa as práticas de desenvolvimento ágil de software. Ela defende a utilização de modelagem e documentação suficiente, mas de maneira eficaz e eficiente para satisfazer as necessidades atuais do projeto. Dentro do contexto da AM, é encorajado o uso de vários modelos em paralelo, conforme necessário para o projeto. Isso pode incluir:

- **Modelos da UML:** para a representação visual de sistemas, incluindo estruturas, comportamentos e interações;
- **Modelos de Dados:** utilizado para descrever a estrutura lógica dos dados que um sistema pode usar;
- **Protótipos de Interface do Usuário:** para explorar e comunicar como os usuários interagirão com o sistema;
- **Modelos Navegacionais:** para descrever como os usuários navegam pelas funcionalidades do sistema.

A ideia é que estes modelos ajudem a equipe de desenvolvimento a entender melhor os requisitos, a estrutura e o comportamento do sistema que está sendo desenvolvido, bem como a comunicar esses aspectos entre os membros da equipe e stakeholders. Além disso, ela incentiva a revisão e a refatoração contínuas desses modelos à medida que novas informações são descobertas e os requisitos do projeto evoluem, alinhando-se assim com os princípios ágeis de desenvolvimento iterativo e adaptativo.

Gabarito: Correto

---

7. **(CESPE / SERPRO – 2021)** Um dos princípios da modelagem ágil é a abstração das ferramentas que serão utilizadas para criar os modelos e suas notações particulares.

Comentários:

Na verdade, um dos princípios fundamentais da modelagem ágil não é especificamente a abstração das ferramentas usadas para criar modelos ou suas notações particulares. Em vez disso, a modelagem ágil enfatiza a eficácia e eficiência na comunicação e na compreensão dos requisitos e do design do sistema, independentemente das ferramentas ou notações específicas utilizadas. A modelagem ágil valoriza:

- A comunicação eficaz entre os membros da equipe e com os stakeholders, mais do que o uso de ferramentas ou notações específicas;

- A criação de modelos que são "bons o suficiente" para atender às necessidades atuais, em vez de buscar a perfeição ou completude;

- A adaptabilidade/capacidade de responder a mudanças, o que pode incluir a alteração de modelos ou a troca de ferramentas conforme necessário para melhor atender aos requisitos do projeto;

Embora a abstração em relação às ferramentas de modelagem possa ser uma consequência dos princípios da modelagem ágil, devido à sua ênfase na simplicidade e na necessidade, não é correto afirmar que abstrair as ferramentas e notações é um princípio central por si só. A escolha de ferramentas e notações deve servir ao objetivo de facilitar a compreensão e a colaboração dentro da equipe de desenvolvimento e com os stakeholders, e não ser um fim em si mesma.

Logo, embora a modelagem ágil possa encorajar a flexibilidade na escolha e uso de ferramentas e notações, a abstração das ferramentas não é um princípio declarado da modelagem ágil.

Gabarito: Errado

---

8. **(FUNCERN / Câmara de Natal-RN – 2023)** É correto afirmar que a modelagem ágil é uma abordagem de desenvolvimento de software que enfatiza a entrega contínua de software funcional e:

a) adota uma abordagem sequencial, na qual o modelo é desenvolvido em fases distintas, tais como: análise de requisitos, projeto, implementação, testes e manutenção.

b) favorece a criação de modelos em partes incrementais, investindo semanas ou até meses na criação de modelos e documentos para cada entrega.

c) incentiva a comunicação constante e a colaboração entre membros da equipe, bem como com os stakeholders do projeto.

d) valoriza a documentação em detalhes, mesmo quando os requisitos do software não estão claramente definidos.

Comentários:

(a) Errado. A modelagem ágil não adota uma abordagem sequencial rígida (também conhecida como modelo em cascata). Em vez disso, foca em iterações curtas e adaptáveis que permitem mudanças rápidas baseadas no feedback.

(b) Errado. Embora a modelagem ágil favoreça a criação de modelos em partes incrementais, a ideia de investir semanas ou meses na criação de modelos detalhados para cada entrega contradiz a natureza ágil de trabalhar com entregas rápidas e eficientes, ajustando-se conforme necessário.

(c) Correto. A modelagem ágil definitivamente incentiva a comunicação constante e a colaboração entre todos os membros da equipe e com os stakeholders. Esta é uma das pedras angulares da abordagem ágil, promovendo um ambiente de trabalho inclusivo e responsivo às mudanças de requisitos.

(d) Errado. A modelagem ágil valoriza a documentação, mas apenas na medida em que ela oferece valor ao processo de desenvolvimento. A abordagem ágil prefere software em funcionamento a documentação abrangente, especialmente em estágios iniciais quando os requisitos podem ainda não estar claramente definidos.

Gabarito: Letra C

LISTA DE QUESTÕES – DIVERSAS BANCAS

1. **(MS-CONCURSOS / CODENI-RJ – 2010)** Trata-se de um modo comum de aplicar a UML, frequentemente com alto retorno no investimento de tempo. Essa definição refere-se a:

   a) Case
   b) Objetos Syntropy
   c) Modelagem Ágil
   d) MDA

2. **(CESPE / SERPRO – 2013)** A modelagem ágil é uma metodologia que apresenta ciclo de vida e processo que podem ser claramente seguidos pelo operador.

3. **(CESPE / CGM-JP – 2018)** Na modelagem tradicional, parte significativa do tempo da equipe de desenvolvimento e dos recursos é despendida com manutenção e ajustes de modelos e diagramas; na modelagem ágil, por princípio, mudanças de requisitos ocorridas ao longo do processo de desenvolvimento ensejarão o descarte dos modelos e diagramas que não forneçam valor em longo prazo, ainda que sejam relacionados aos requisitos afetados.

4. **(CESPE / BANRISUL – 2022)** A filosofia da modelagem ágil não admite decisões que levem um projeto a sua rejeição e a sua refatoração, pois os profissionais de tecnologia não possuem todas as respostas e outros stakeholders envolvidos no negócio devem ser respeitados e integrados ao processo.

5. **(UFMT / UFMT – 2016 – Letra A)** A modelagem Ágil tem foco na codificação de artefatos de software e, por este motivo, dispensa completamente a documentação de sistema.

6. **(UFMG / UFMG – 2016 – Letra C)** A Modelagem Ágil (AM) sugere que vários modelos, como os modelos da UML, modelos de dados, protótipos de interface do usuário e modelos navegacionais, dentre outros, sejam construídos e utilizados em paralelo.

7. **(CESPE / SERPRO – 2021)** Um dos princípios da modelagem ágil é a abstração das ferramentas que serão utilizadas para criar os modelos e suas notações particulares.

8. **(FUNCERN / Câmara de Natal-RN – 2023)** É correto afirmar que a modelagem ágil é uma abordagem de desenvolvimento de software que enfatiza a entrega contínua de software funcional e:

   a) adota uma abordagem sequencial, na qual o modelo é desenvolvido em fases distintas, tais como: análise de requisitos, projeto, implementação, testes e manutenção.

b) favorece a criação de modelos em partes incrementais, investindo semanas ou até meses na criação de modelos e documentos para cada entrega.

c) incentiva a comunicação constante e a colaboração entre membros da equipe, bem como com os stakeholders do projeto.

d) valoriza a documentação em detalhes, mesmo quando os requisitos do software não estão claramente definidos.

GABARITO – DIVERSAS BANCAS

1. LETRA C
2. ERRADO
3. CORRETO
4. ERRADO
5. ERRADO
6. CORRETO
7. ERRADO
8. LETRA C

MINIMUM VIABLE PRODUCT (MVP)

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Meus queridos alunos, vocês concordam comigo que é natural que patrocinadores de projetos queiram lançar um produto totalmente completo para seus clientes. *É o que os clientes esperam, não é?* Esta é a pergunta errada a ser feita! A pergunta certa seria: *qual é o mínimo que meus clientes mais valiosos precisam primeiro?* Pessoal, raramente eles precisarão de tudo! Exemplo: nesse momento, eu estou escrevendo essa aula no Microsoft Word 2019.

Em 2006, a equipe de experiência de usuário do Microsoft Office publicou resultados de uma pesquisa que descobriu que apenas cinco comandos representavam 32% de todo o uso do Microsoft Word (Copiar, Salvar, Colar, Desfazer e Negritar). *Dá para acreditar nisso?* **Esse software tem uma infinidade absurda de comandos, mas 1/3 do uso dos clientes trata apenas de cinco comandos bastante simples.**

O Minimum Viable Product (MVP) é um conceito (inicialmente cunhado e definido por Frank Robinson no contexto de desenvolvimento de software) **que ajuda a identificar e alcançar a atividade mínima viável** que inicialmente pode provar e, posteriormente, entregar os benefícios necessários para alcançar uma solução viável. *Difícil de entender?* **Basicamente, MVP não é uma versão mais barata ou diluída do produto final.**

Trata-se de uma representação dele – no todo ou em parte – que oferece valor ou representa a atividade tangível ou solução de alguma forma. Um grande autor chamado Eric Ries o define como **a versão de um novo produto que permite que uma equipe colete o máximo de aprendizado sobre os clientes com a menor quantidade de esforço**. O objetivo dessa abordagem é construir rapidamente um software que apresente seus aspectos mais importantes.

O MVP poderá ajudá-lo a obter um feedback inicial e, em seguida, corrigir – conforme a necessidade – antes de construir um produto mais pesado. Ele pode ser um serviço completo que atende a grupos de usuários limitados ou serviços parciais que abordam grupos de usuários mais amplos. O feedback dos clientes é extremamente importante, portanto é interessante lançar o MVP para usuários reais.

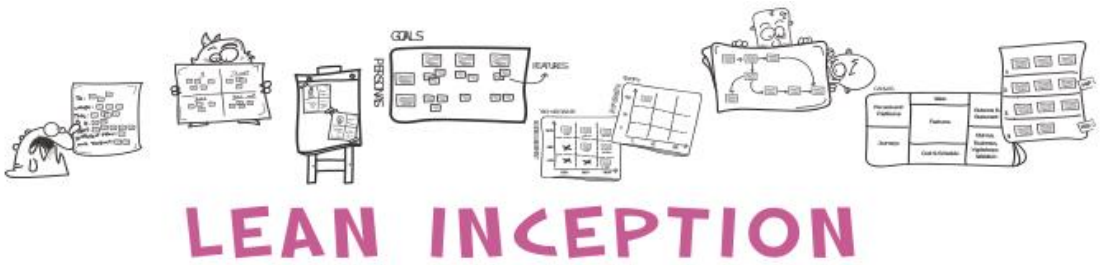
Se você se concentrar no mínimo produto viável, significa que você dará o mínimo nas mãos dos clientes e começará a, de fato, aprender com seu comportamento real dos usuários. Aquele cara lá de cima chamado Eric Ries conta uma história muito interessante sobre um não-sucesso. Ele conta que trabalhou por cinco anos em uma startup, com uma equipe escolhida à mão e tudo do bom e do melhor.

Quando ele se juntou à equipe, eles não lhe disseram o que era o produto, apenas os nomes das pessoas legais com quem ele trabalharia. A equipe ficava em um hangar aeronaves. O projeto era todo secreto: eles estavam construindo um produto chamado Big Idea – o futuro da Internet. Eles geraram entusiasmo no Vale do Silício, a imprensa ficava curiosa e quando o produto finalmente foi lançado, houve linhas em linhas em jornais e revistas.

No entanto, o que produto não obteve foi o cliente. Cinco anos para construir o Big Idea com todos os recursos, industrialmente fortalecidos e depois simplesmente falhar. *Não há muita sabedoria nisso tudo, vocês concordam comigo?* Esta é uma lição repetida de novo e de novo e de novo. As organizações continuam envolvendo o desenvolvimento de seus produtos em segredo, esperando até que eles tenham o conjunto completo de recursos antes de bater no mercado.

Galera, um projeto ágil é conduzido através de entregas rápidas e frequentes. Um projeto bem executado coloca ênfase em entregas de valor de acordo com objetivos de negócios e os usuários de destaque. **O MVP – versão mais simples de um produto que pode ser disponibilizada para o negócio – promove essa liberação incremental de software.** É diferente de produtos criados da forma tradicional, tipicamente com um período longo de criação de protótipo, análise e elaboração.

Como já vimos, ele foca no produto mínimo, validação de pedaços menores – bem menos elaborados do que uma versão final –, mas viável para verificar se o direcionamento está correto.



Com base no entendimento comum dos objetivos do negócio e dos usuários de destaque, busca-se o esclarecimento dos pedaços menores que compõem o produto. Por fim, vamos falar rapidamente um pouquinho sobre o Lean Inception! *O que diabos é isso, Diego?* Galera, não existe ninguém melhor para explicar esse conceito do que o próprio criador dele. E adivinhem só: ele é brasileiro e se chama Paulo Caroli:

> *Projetos ágeis enfatizam a entrega precoce e contínua de software valioso, cujo valor vem dos objetivos de negócios e das necessidades dos clientes. A criação do produto Lean StartUp ajuda nessa direção, promovendo a liberação incremental de um MVP (Minimum Viable Product) – uma versão simples de um produto que é dada aos usuários para validar as principais premissas do negócio.*
>
> *Mas como nós resolvemos o que deve estar em MVP e começamos um projeto ágil o mais rapidamente possível? Como garantir que a equipe comece a criar o produto com um entendimento compartilhado e um plano eficaz? Eu projetei o Lean Inception para responder a essas perguntas.*
>
> *O ágil simples não tem um trabalho inicial, mas na prática percebemos que temos de fazer um pouco disso. Para a ThoughtWorks, Inception é esse "pouco disso". Desde que ingressei na ThoughtWorks, percebi que todos os projetos*

ágeis da empresa começaram de forma semelhante. A equipe do projeto se reuniria por algumas semanas, passando por muitas atividades antes de iniciar o trabalho de entrega: isso foi a Inception. Nossas Inceptions variam de projeto para projeto, mas elas geralmente geram alinhamento entre o negócio e as pessoas técnicas, e criam uma lista ordenada de estórias de usuários com estimativas juntamente com um plano de lançamento.

Eu fiquei muito satisfeito facilitando essas Inceptions ágeis dessa forma até 2011 – ano em que meu filho nasceu. A questão é que eu era o facilitador da Inception e ela levaria de duas a quatro semanas. E eu não poderia ficar longe de casa por mais de uma semana. Eu tive que fazer as Inceptions mais enxutas, de alguma forma fazê-las caber em uma semana. Eu estava indo fazer a minha primeira viagem após meu filho ter nascido. Em um longo voo de São Paulo a São Francisco, li o livro Lean StartUp, de Eric Ries. A partir disso, eu encontrei a desculpa perfeita para reduzir o comprimento do Inception e voltar para casa depois de uma semana.

Por que é chamado de Lean Inception?

Esse novo estilo de inception é definitivamente uma mudança em relação a Inception de antigamente. A equipe não mais escreve e estima histórias de usuários. Ao experimentar esse novo estilo, o nome "Inception" deu a todos a mensagem errada. Eu precisava de um nome diferente. O novo estilo de Inception é Lean (leve) por duas razões:

- A duração da Inception é menor, removendo tudo o que não se relacionava ao produto (como arquitetura, projeto etc.), tornando-se Lean;

- O resultado final da Inception é a compreensão do MVP, um conceito fundamental do movimento Lean StartUp. Portanto, o novo estilo de Inception tinha um nome claro: The Lean Inception.

Por que um Lean Inception?

Uma Lean Inception é útil quando a equipe precisa desenvolver iterativamente um MVP. Embora o termo seja muitas vezes mal compreendido, a propriedade central de um MVP é que é algo que construímos para saber se vale a pena continuar construindo um produto. Portanto, escolhemos recursos baseados em testar nossas premissas sobre o que é valioso para nossos usuários. Para isso, precisamos entender quem são nossos usuários, que atividade eles fazem que o produto suporta e como medir se eles acharem o produto útil. Achamos que o workshop é valioso em duas circunstâncias principais.

- Grandes projetos acham que uma Lean Inception é valiosa para começar rapidamente e para ser orientada a trabalhar em um estilo Lean. Esse começo constrói iterações iniciais projetadas para descobrir e testar quais recursos são verdadeiramente valorizados pelos seus usuários.

- Organizações menores (como as Startups) usam Lean Inceptions para pegar uma ideia que já foi testada por alguns MVPs pré-software e evoluí-lo em um produto de software.

Esse workshop é especificamente sobre a compreensão de um MVP. Ele não substitui sessões de ideação, pesquisa de clientes, revisão de arquitetura ou análise competitiva. É uma técnica específica que faz parte da compreensão do que é preciso para construir um produto de sucesso. Como ele se encaixa exatamente com essas outras atividades depende muito do contexto específico da sua organização e do esforço de desenvolvimento que você está empregando.

A Transformação Digital (TD) é um tema cada vez mais relevante, impulsionando organizações a revisarem suas crenças, estratégias de negócio, produtos, práticas de gestão e seu papel na sociedade. Este movimento é notadamente importante no setor público, onde a necessidade de

inovação digital é crescente. **Uma ferramenta chave neste processo é a Lean Inception, que se destaca por sua agilidade e eficácia em definir e construir o Produto Mínimo Viável (MVP).**

A Lean Inception é um workshop intensivo, geralmente realizado em uma semana, focado em atividades de alinhamento e definição de objetivos. O objetivo é desenvolver um plano claro para o projeto, marcado por entregas sequenciais que visam validar o MVP, assegurando que o produto final atenda às necessidades dos usuários. Paulo Caroli narra sua experiência transformadora ao adaptar o processo de Inception para torná-lo mais compacto e focado no MVP.

Esta adaptação foi inspirada após Caroli se tornar pai em 2011, o que o motivou a buscar métodos que exigissem menor tempo de ausência de casa. Caroli vislumbrou a possibilidade de otimizar a duração das Inceptions. **Essa nova abordagem de Inception, mais enxuta e eficiente, integra princípios do Design Thinking e Lean Startup, focando nas necessidades dos usuários através de um processo de design interativo.**

Isso permite a realização de Inceptions mais ágeis, voltadas diretamente para o desenvolvimento progressivo do produto. O ciclo "construir – aprender – construir – aprender", baseado no conceito de MVP, é fundamental. A interação inicial com o usuário proporciona aprendizados valiosos, que são então utilizados para refinar ou expandir o produto. **Assim, cada etapa de desenvolvimento é informada pelas respostas e necessidades dos usuários, permitindo ajustes precisos e eficazes.**

(PETROBRÁS – 2019) Do ponto de vista de design e de usabilidade, o produto mínimo viável dispensa padrões de qualidade comercial, sendo suficiente que a funcionalidade parcial proposta tenha um nível básico de confiabilidade.

Comentários: O MVP (Minimum Viable Product) é uma versão simples de um produto que é dada aos usuários para validar as principais premissas do negócio. Apesar disso, no MVP os padrões de qualidade não são dispensados, eles devem sempre existir (Errado).

(BNB – 2022) MVP (Mínimo Produto Viável) é um protótipo do sistema que é simples o suficiente para testar as características técnicas de um produto bem como é o conceito de uma solução utilizável e valiosa para o negócio.

Comentários: MVP (Minimum Viable Product) não é um protótipo! Trata-se de um produto que possui apenas os principais recursos necessários para torná-lo funcional e utilizável. Geralmente é a primeira versão de um produto e é usado para testar o conceito e obter feedback dos usuários. Um protótipo, por outro lado, é uma versão básica de um produto ou serviço usado para testar ideias, processos e experiência do usuário (Errado).

QUESTÕES COMENTADAS

1. **(CESPE / SERPRO – 2023)** O produto mínimo viável (MVP) é uma ferramenta fundamental no processo de validação de uma ideia de negócio, por meio da coleta de problemas, oportunidades e feedbacks.

Comentários:

O Produto Mínimo Viável (MVP) é uma versão inicial de um produto com funcionalidades básicas suficientes para atender os primeiros usuários e coletar feedback. Essa abordagem permite validar uma ideia de negócio de forma rápida e com menor custo, identificando problemas, oportunidades e áreas de melhoria antes de investir em um desenvolvimento mais completo. O MVP é uma ferramenta fundamental para testar hipóteses de mercado e ajustar o produto de acordo com as necessidades reais dos usuários.

Gabarito: Correto

---

2. **(CESPE / AGER-MT – 2023)** Assinale a opção que indica uma prática ágil adotada para auxílio na validação de um modelo de negócio por meio de uma versão funcional reduzida do produto.

a) protótipo
b) mockup
c) MVP
d) unidade de teste
e) wireframe.

Comentários:

(a) Errado. Um protótipo é uma versão preliminar de um produto ou sistema utilizado para testar conceitos e funcionalidades, mas não necessariamente foca na validação de um modelo de negócio;

(b) Errado. Ele é uma representação visual estática do produto, geralmente usada para demonstrar a interface do usuário, mas não é funcional e não é utilizado para validar um modelo de negócio;

(c) Correto. O MVP (Minimum Viable Product) é uma versão funcional reduzida do produto que permite validar um modelo de negócio com o mínimo de esforço e recursos;

(d) Errado. Uma unidade de teste é um método para validar o comportamento de pequenas partes do código, mas não é usada para validar um modelo de negócio;

(e) Errado. Trata-se de uma representação esquelética da interface do usuário, utilizado para definir a estrutura e o layout do produto, mas não é funcional e não serve para validar um modelo.

Gabarito: Letra C

---

3. **(CESPE / INPI – 2024)** No processo de desenvolvimento utilizando um produto mínimo viável (MVP), entre as etapas do ciclo build, measure e learn (construir, medir e aprender), a etapa build (construir) é a mais aderente ao objetivo central do MVP.

Comentários:

Errado. No processo de desenvolvimento utilizando um Produto Mínimo Viável (MVP), o objetivo central é aprender rapidamente sobre o mercado e as necessidades dos clientes com o mínimo esforço e investimento. Portanto, a etapa mais aderente ao objetivo central do MVP é a etapa "learn" (aprender). A construção (build) é importante, mas serve principalmente como um meio para obter feedback real dos usuários que será medido e analisado na etapa "measure" (medir), culminando no aprendizado que orientará as próximas iterações e melhorias do produto.

Gabarito: Errado

---

4. **(CESGRANRIO / TRANSPETRO – 2023)** Em qualquer negócio, é necessário compreender as necessidades dos usuários de forma a atendê-los adequadamente. Nesse sentido, o conceito de Produto Mínimo Viável (MVP) passou a ser muito utilizado no desenvolvimento de produtos de softwares, sendo que, nessa área, o MVP é:

a) a versão mais simples de um produto, que será criada e disponibilizada aos usuários para validar uma ideia e coletar dados imprescindíveis para validar o direcionamento do negócio.

b) muito utilizado em startups que desejam criar um novo produto, sendo, porém, inadequado para empresas já estabelecidas testarem novas ideias, pois sua plataforma de usuários já fornece as informações necessárias para o lançamento de novos produtos.

c) o primeiro lançamento de um novo produto de software destinado ao grande público, correspondendo à V1.0 desse software lançado no mercado, e os esforços destinados à sua construção devem ser compatíveis para garantir alta manutenibilidade.

d) uma versão necessariamente reduzida de um produto de software, implementado com a utilização das mesmas tecnologias que serão usadas no produto final.

e) a versão de um produto de software lançado no mercado após uma validação de hipóteses, realizada na etapa de pesquisas, durante o processo de ideação desse produto.

Comentários:

(a) Correto. O Produto Mínimo Viável (MVP) é a versão mais simples de um produto, criada e disponibilizada aos usuários para validar uma ideia e coletar dados imprescindíveis para validar o direcionamento do negócio;

(b) Errado. O MVP é utilizado tanto por startups quanto por empresas estabelecidas para testar novas ideias e validar hipóteses, independente da plataforma de usuários existente;

(c) Errado. O MVP não é necessariamente a primeira versão completa (V1.0) destinada ao grande público, mas uma versão básica para validação inicial de hipóteses;

(d) Errado. O MVP é uma versão reduzida do produto, mas não é necessariamente implementado com as mesmas tecnologias do produto final; o foco é na validação da ideia;

(e) Errado. O MVP é lançado no mercado para validar hipóteses, mas é uma etapa anterior à validação completa do produto no mercado, não após essa validação.

Gabarito: Letra A

---

5. **(CESPE / EMPREL – 2023)** Uma equipe de desenvolvimento de software está criando um aplicativo para conceder financiamento de imóveis para residentes no Brasil que não conseguem comprovar renda mensal regular. Nessa situação hipotética, para empregar conceitos de MVP (minimum viable product), essa equipe deve entregar uma versão do produto que:

a) permita que um número limitado de moradores de uma comunidade consiga simular e, eventualmente, contratar o valor de um financiamento imobiliário, receber o valor correspondente (de acordo com algumas opções de garantia de pagamento já aceitas) e iniciar o recebimento do pagamento do financiamento.

b) permita que um número limitado de moradores de uma comunidade consiga simular o valor de um financiamento imobiliário, faltando apenas a concessão dos valores finais do financiamento e a cobrança correspondente.

c) permita que os moradores de uma comunidade aprendam sobre as opções tradicionais de financiamento imobiliário e como aumentar a sua renda familiar, devendo ser selecionado um percentual dos moradores que concluírem o aprendizado para simular o valor de um financiamento imobiliário, receber o valor correspondente e iniciar o recebimento do pagamento do financiamento.

d) ofereça um novo modelo de financiamento para os profissionais do ramo imobiliário que possa contribuir com o novo sistema, de modo que os desenvolvedores possam confirmar quais são os recursos desejados pelo público-alvo, antes de decidir lançar o produto no mercado.

e) permita que um número limitado de moradores de uma comunidade consiga simular e, eventualmente, contratar o valor de um financiamento de um bem de consumo de baixo valor, receber o valor correspondente (de acordo com algumas opções de garantia de pagamento já aceitas) e iniciar o recebimento do pagamento do financiamento.

Comentários:

(a) Correto. Esta alternativa descreve um MVP que permite testar o fluxo completo, desde a simulação até o recebimento do financiamento, o que é útil para validar a viabilidade do processo para o público-alvo;

(b) Errado. A ausência das etapas finais de concessão e cobrança pode não fornecer informações suficientes sobre a viabilidade do processo completo do financiamento imobiliário;

(c) Errado. A inclusão de componentes educacionais desvia o foco do MVP, que deve testar apenas as funcionalidades essenciais relacionadas ao financiamento;

(d) Errado. Focar em profissionais do ramo imobiliário desvia do público-alvo principal do MVP, que são os residentes sem comprovação de renda regular;

(e) Errado. A alternativa menciona financiamento para bens de consumo de baixo valor, que não é o foco do aplicativo de financiamento imobiliário para residentes.

Gabarito: Letra A

Lista de Questões

1. **(CESPE / SERPRO – 2023)** O produto mínimo viável (MVP) é uma ferramenta fundamental no processo de validação de uma ideia de negócio, por meio da coleta de problemas, oportunidades e feedbacks.

2. **(CESPE / AGER-MT – 2023)** Assinale a opção que indica uma prática ágil adotada para auxílio na validação de um modelo de negócio por meio de uma versão funcional reduzida do produto.

   a) protótipo
   b) mockup
   c) MVP
   d) unidade de teste
   e) wireframe.

3. **(CESPE / INPI – 2024)** No processo de desenvolvimento utilizando um produto mínimo viável (MVP), entre as etapas do ciclo build, measure e learn (construir, medir e aprender), a etapa build (construir) é a mais aderente ao objetivo central do MVP.

4. **(CESGRANRIO / TRANSPETRO – 2023)** Em qualquer negócio, é necessário compreender as necessidades dos usuários de forma a atendê-los adequadamente. Nesse sentido, o conceito de Produto Mínimo Viável (MVP) passou a ser muito utilizado no desenvolvimento de produtos de softwares, sendo que, nessa área, o MVP é:

   a) a versão mais simples de um produto, que será criada e disponibilizada aos usuários para validar uma ideia e coletar dados imprescindíveis para validar o direcionamento do negócio.

   b) muito utilizado em startups que desejam criar um novo produto, sendo, porém, inadequado para empresas já estabelecidas testarem novas ideias, pois sua plataforma de usuários já fornece as informações necessárias para o lançamento de novos produtos.

   c) o primeiro lançamento de um novo produto de software destinado ao grande público, correspondendo à V1.0 desse software lançado no mercado, e os esforços destinados à sua construção devem ser compatíveis para garantir alta manutenibilidade.

   d) uma versão necessariamente reduzida de um produto de software, implementado com a utilização das mesmas tecnologias que serão usadas no produto final.

   e) a versão de um produto de software lançado no mercado após uma validação de hipóteses, realizada na etapa de pesquisas, durante o processo de ideação desse produto.

5. **(CESPE / EMPREL – 2023)** Uma equipe de desenvolvimento de software está criando um aplicativo para conceder financiamento de imóveis para residentes no Brasil que não

conseguem comprovar renda mensal regular. Nessa situação hipotética, para empregar conceitos de MVP (minimum viable product), essa equipe deve entregar uma versão do produto que:

a) permita que um número limitado de moradores de uma comunidade consiga simular e, eventualmente, contratar o valor de um financiamento imobiliário, receber o valor correspondente (de acordo com algumas opções de garantia de pagamento já aceitas) e iniciar o recebimento do pagamento do financiamento.

b) permita que um número limitado de moradores de uma comunidade consiga simular o valor de um financiamento imobiliário, faltando apenas a concessão dos valores finais do financiamento e a cobrança correspondente.

c) permita que os moradores de uma comunidade aprendam sobre as opções tradicionais de financiamento imobiliário e como aumentar a sua renda familiar, devendo ser selecionado um percentual dos moradores que concluírem o aprendizado para simular o valor de um financiamento imobiliário, receber o valor correspondente e iniciar o recebimento do pagamento do financiamento.

d) ofereça um novo modelo de financiamento para os profissionais do ramo imobiliário que possa contribuir com o novo sistema, de modo que os desenvolvedores possam confirmar quais são os recursos desejados pelo público-alvo, antes de decidir lançar o produto no mercado.

e) permita que um número limitado de moradores de uma comunidade consiga simular e, eventualmente, contratar o valor de um financiamento de um bem de consumo de baixo valor, receber o valor correspondente (de acordo com algumas opções de garantia de pagamento já aceitas) e iniciar o recebimento do pagamento do financiamento.

GABARITO

1. CORRETO
2. LETRA C
3. ERRADO
4. LETRA A
5. LETRA A

DEPLOYMENT PIPELINE

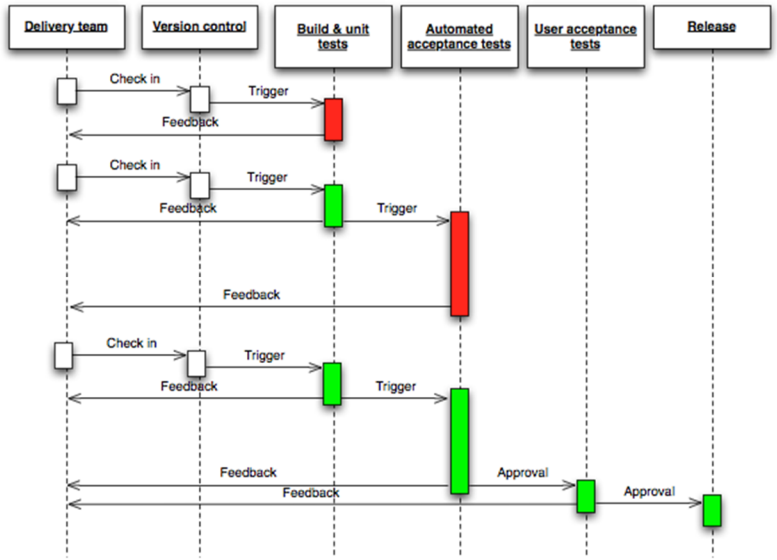
Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Martin Fowler afirma que um dos desafios de um ambiente de *builds* e testes automatizados é que você quer que a compilação seja rápida, de modo que você possa obter um feedback mais veloz, porém você sabe que testes mais abrangentes levam um bom tempo para serem executados. **O** Deployment Pipeline é uma maneira de lidar com esse problema, quebrando o processo em etapas. Cada estágio proporciona um aumento da confiança – muitas vezes, às custas de um tempo extra.

Estágios iniciais podem encontrar a maioria dos problemas produzindo um feedback mais rápido, enquanto as fases posteriores fornecem um feedback mais lento. Deployment Pipelines podem ser considerados uma parte central da prática ágil de Entrega Contínua. Normalmente, a primeira fase fará qualquer compilação e fornecerá binários para fases posteriores, que podem incluir checagens manuais (isto é, testes que não podem ser automatizados).

Fases podem ser automáticas (por meio de triggers) ou sujeitas a ações humanas para serem realizadas. Podem ser paralelizadas sobre várias máquinas para aumentar a velocidade da *build*.



A implantação em produção é geralmente o estágio final! Podemos dizer que o papel do Deployment Pipeline é detectar qualquer mudança que levará a problemas de produção. Elas incluem desempenho, segurança, ou problemas de usabilidade. Um Deployment Pipeline deve habilitar a colaboração entre diversos grupos envolvidos na entrega de software e fornecer visibilidade do fluxo de mudanças.

Bem, essas foram as palavras do ilustre Martin Fowler. Vamos esclarecer alguns pontos agora! **O Deployment Pipeline é uma implementação automatizada do processo de construção,**

implantação, testes e release do seu sistema. Eu gostaria que vocês acompanhassem comigo passo-a-passo a imagem apresentada acima que demonstra o funcionamento do Deployment Pipeline.

Vamos supor que eu faça uma mudança no código. **Dessa forma, a equipe de entrega fará um check-in no meu sistema de controle de versões (Ex: SVN) e imediatamente isso irá disparar uma trigger para a realização de testes de unidade e da build automatizada.** Se houver algum problema (indicado pelo retângulo vermelho na imagem anterior), é retornado um feedback para a equipe de entrega.

Nesse momento, a prioridade será consertar os problemas encontrados no estágio anterior, isto é, só é permitido fazer um novo *check-in* após os erros terem sido corrigidos. *Bacana até aqui?* A galera resolveu o problema encontrado e foi feito *check-in* novamente. **Houve o disparo de uma nova trigger para ativar os testes de unidade e build automatizada.** Observem que agora não houve nenhum problema (vejam o retângulo verde).

Assim, é disparada outra trigger para os testes de aceitação automatizados – eles demoram um pouco mais. Vejam que deu problema novamente (retângulo vermelho)! Volta tudo, a equipe recebe o feedback, corrige o problema e refaz os passos novamente até passar pelos testes de aceitação automatizados. Após isso, ocorrem os testes de aceitação do usuário. Aprovado, segue para a release! A maior diferença para a Entrega Contínua é que cada build é candidata a release.

Dessa forma, não é qualquer *build* que vai para produção. Ela deve passar por vários estágios, aumentando a confiança da equipe de que, uma vez em produção, será suficientemente robusta.

(TCU – 2015) Na gerência de um pipeline de implantação (deployment pipeline), recomenda-se que o código-fonte seja compilado repetidas vezes em contextos diferentes: durante o estágio de commit, nos testes de aceitação, nos testes de capacidade e nos testes exploratórios.

_____

Comentários: a questão afirma que, em cada fase, recomenda-se que o código seja compilado. *Vocês se lembram da figura apresentada em aula?* Pois é, o código deve ser compilado todas as vezes em que falhar em alguma das fases, porque há o feedback e a correção. Caso não haja falha, não faz sentido compilar o código em cada fase. Faz sentido recompilar o código na fase de testes automatizados? Nenhum! (Errado).

AGILE THINK CANVAS

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Mais um assunto tranquilaço! Agile Think Canvas (ATC) é um manual de práticas colaborativas construído com base em um método de trabalho que utiliza conceitos de Design Thinking e Gestão Ágil para o desenvolvimento de novos produtos e serviços. Voltado para aqueles que se utilizam da criatividade nos processos de inovação dos produtos e serviços, a metodologia Agile Think Canvas engloba diversos fatores.

Entre esses fatores, temos: definição de uma visão de produto, estudo de personas, suas respectivas jornadas, entendimento dos riscos, definição do MVP, prototipação, construção do produto por meio de técnicas amplamente difundidas por métodos e frameworks presentes no Agile e aprendizado, etc – essa não é uma lista exaustiva. **O ATC busca ajudar os profissionais das mais diversas áreas que atuam através de processos tradicionais a romperem barreiras.**

Indicado a estudantes, professores e profissionais das áreas de gestão, governança, negócios e produtos que tenham interesse em explorar novas ferramentas visuais de gestão de projetos, adaptadas para suas atividades organizacionais, propiciando maior agilidade, transparência e resultados com alto valor agregado a empresas e clientes. **Galera, isso nunca caiu em prova e é um livro relativamente grande, entre outros.** Bem, é isso...

GESTÃO ÁGIL DE PROJETOS

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Faaaaala, galera... mais um assunto na nossa aula! A abordagem tradicional de gerenciamento de projetos estava sendo apedrejada com diversas críticas a sua estrutura inflexível, complexidade, etc[1]. A partir daí, começou a surgir diversas propostas de abordagens de gerenciamento de projetos, alternativas ao modelo tradicional, para o desenvolvimento de produtos inovadores em ambientes dinâmicos de projeto.

O gerenciamento tradicional de projetos apresenta uma redução em sua eficiência quando aplicados em projetos com elevado nível de incertezas, com equipes pequenas e co-localizadas, e elevado nível de interação entre membros da equipe de projeto e clientes. **Na abordagem tradicional, era praticamente possível antecipar ou prever variáveis relacionadas ao desenvolvimento do projeto**.

Professor, o que as novas abordagens propunham? **Elas propunham novos princípios de gerenciamento em que o envolvimento da equipe e a utilização de técnicas simplificadas** com enfoque no desenvolvimento da autogestão melhor se adequam a ambientes em que as incertezas e mudanças predominam. *Professor, mas qual a definição exatamente de Gerenciamento Ágil de Projetos?*

Highsmith diz que é um conjunto de princípios, valores e práticas que auxiliam a equipe a entregar produtos ou serviços de valor em um ambiente de projetos desafiador. Chin afirma que é um novo elemento base que pode contribuir para o desenvolvimento da teoria tradicional de gestão de projetos de modo que permita que as empresas sejam mais eficientes na gestão de projetos em ambientes incertos.

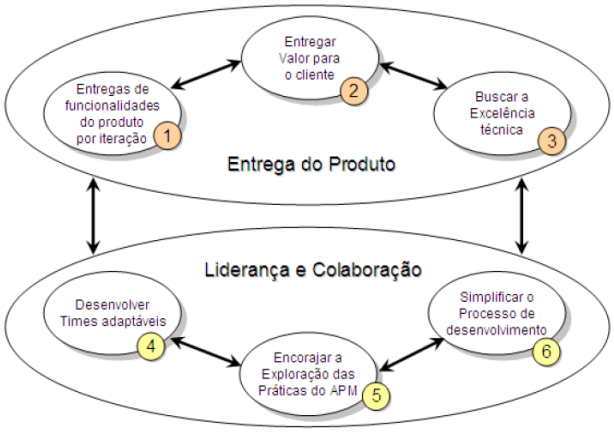
Já o termo "agilidade", no contexto de desenvolvimento ágil e adaptável, significa que o time de projeto desenvolva habilidade necessária para criar e responder às mudanças ocorridas no projeto. **A abordagem do gerenciamento ágil de projetos deve ser encarada como uma habilidade para balancear flexibilidade e estabilidade**. Para tal, buscam-se pessoas com habilidade de improvisação

Augustine afirma que é necessário existir um nível adequado de flexibilidade com estabilidade, caos com ordem, execução com planejamento e exploração com otimização. **Além disso, ele declara que é imprescindível ter habilidade para entregar valor para o cliente, frente à imprevisibilidade e o dinamismo inerente aos projetos de novos produtos**. DeCarlo diz que é a arte e ciência de facilitar e gerenciar o fluxo de pensamentos, emoções e interações. *Para quê?*

[1] Por exemplo: guias de melhores práticas e corpos de conhecimento como PMBOK.

Para produzir resultados de valor em condições adversas e complexas que requerem velocidade, e estão sujeitas a mudanças constantes e elevados níveis de incertezas e estresse. *Galera... são muitas definições, concordam?* No entanto, elas têm algumas coisas em comum! **Todas falam sobre a necessidade de flexibilidade e habilidade para absorver mudanças durante o ciclo de vida do projeto**.

Além disso, há um forte enfoque humanista, valorização do aprendizado contínuo, e a capacidade dos indivíduos como participantes ativos do processo, ao invés de valorizar excessivamente as técnicas e processos de gestão de projetos. O Gerenciamento Ágil de Projetos (GAP) ou Agile Project Management (APM) possui alguns princípios básicos que regem sua aplicação e exploração apresentados na imagem a seguir:

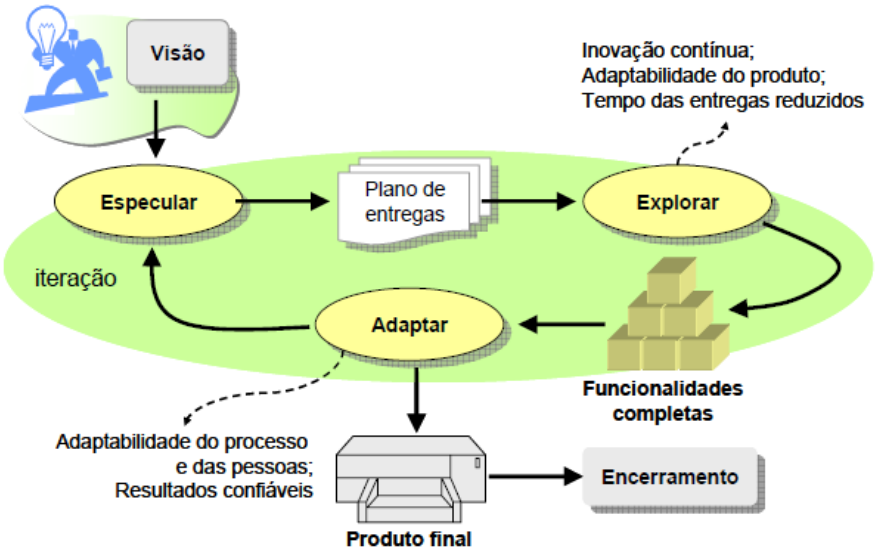


Esses princípios remetem a uma reflexão sobre o enfoque dado na gestão de projetos. Muitos dos autores da teoria tradicional enfatizam o valor do plano de projeto e a antecipação de eventos. **Situações incomuns quando se trata de desenvolvimento de produtos em contextos dinâmicos**. Por exemplo: entregar valor para o cliente também é uma diretriz da abordagem tradicional. *Bacana?*

No entanto, o que se questiona dessa abordagem é como agregar valor para o cliente, como simplificar o processo de desenvolvimento e como criar times adaptáveis em frente às dificuldades em antecipar atividades e seguir um planejamento sem a flexibilidade necessária para absorver eventuais mudanças no projeto. Galera, vamos ver uma tabelinha com as diferenças entre os dois:

ABORDAGEM	TRADICIONAL	ÁGIL
METAS DO PROJETO	Enfoque na finalização do projeto no tempo, custo e requisitos de qualidade.	Enfoque nos resultados do negócio, atingir múltiplos critérios de sucesso.
PLANO DO PROJETO	Uma coleção de atividades que são executadas como planejado para atender a tempo, custo e qualidade.	Organização e o processo para atingir as metas esperadas e os resultados para o negócio.

ABORDAGEM GERENCIAL	Rígida, com foco no plano inicial.	Flexível, variável e adaptativa.
TRABALHO E EXECUÇÃO	Previsível, mensurável, linear e simples.	Imprevisível, não-mensurável, não-linear e complexo.
INFLUÊNCIA DA ORGANIZAÇÃO	Mínimo, imparcial a partir do kick-off do projeto.	Afeta o projeto ao longo de sua execução.
CONTROLE DO PROJETO	Identificar desvios do plano inicial e corrigir o trabalho para seguir o plano.	Identificar mudanças no ambiente, e ajustar o plano adequadamente.
APLICAÇÃO DA METODOLOGIA	Aplicação genérica e igualitária em todos os projetos.	Adaptação do processo dependendo do tipo de projeto.
ESTILO DE GESTÃO	Um modelo atende todos os tipos de projetos.	Abordagem adaptativa, um único modelo não atende todos os tipos de projetos.



FASES	DESCRIÇÃO
VISÃO	O objetivo é determinar a visão do produto e o escopo de projeto, a comunidade do projeto, e definir como a equipe irá trabalhar e interagir. Define-se o que será entregue, os envolvidos e como o time pretende trabalhar.
ESPECULAÇÃO	O objetivo é planejar o projeto com base na visão preliminar construída com o apoio do time do projeto, sobre o que precisa ser entregue, quem são os envolvidos e qual será a estratégia adotada.
EXPLORAÇÃO	O objetivo é executar o que foi planejado, executando as entregas, promovendo a auto-organização e auto-disciplina da equipe de projetos e tratando da gestão das interações da equipe do projeto com o cliente.
ADAPTAÇÃO	O objetivo é rever os resultados da fase anterior, analisar o progresso do projeto e o desempenho da equipe de projetos, para eventuais adaptações no plano de projeto, entregas e plano de iterações, caso seja necessário.

ENCERRAMENTO	Transferem-se os conhecimentos-chave adquiridos no projeto, e celebrado os resultados obtidos. Recomendam-se mini-fechamentos ao final de cada iteração no projeto.

A gestão ágil de projetos é uma abordagem que prioriza a entrega rápida e contínua de valor ao cliente, adaptando-se às mudanças de forma dinâmica e promovendo a colaboração entre as equipes. Vamos explorar as noções básicas dessa abordagem, focando em cooperação, flexibilidade de escopo, interatividade, autonomia e empoderamento de equipes e a programação em pares:

Cooperação

Descrição: a cooperação é um dos pilares da gestão ágil de projetos. Envolve a colaboração contínua entre todos os membros da equipe, bem como entre a equipe e os stakeholders, para garantir que todos estejam alinhados e trabalhando em direção ao mesmo objetivo.

Na prática, isso significa que as equipes ágeis operam em ciclos curtos e interativos, como sprints, onde feedback regular é compartilhado. Reuniões diárias (daily stand-ups) são comuns, permitindo que os membros da equipe atualizem uns aos outros sobre seu progresso e desafios. A cooperação não se limita à equipe interna; envolve também os clientes e stakeholders, que são integrados ao processo de desenvolvimento através de revisões e validações constantes.

Flexibilidade de Escopo

Descrição: diferente da gestão tradicional de projetos, onde o escopo é definido rigidamente no início, a gestão ágil promove a flexibilidade de escopo. Isso significa que o escopo do projeto pode ser ajustado continuamente com base no feedback recebido e nas necessidades emergentes do cliente.

Na prática, essa flexibilidade é gerenciada através de um backlog priorizado, que é constantemente revisado e atualizado conforme o projeto avança. O Product Owner (dono do produto) trabalha junto com a equipe para ajustar o backlog, garantindo que as funcionalidades mais valiosas sejam desenvolvidas primeiro. Isso permite que a equipe responda rapidamente a mudanças no mercado ou nos requisitos do cliente, sem comprometer o cronograma geral do projeto.

Interatividade

Descrição: a gestão ágil enfatiza a interatividade, tanto no processo de desenvolvimento quanto na comunicação entre os membros da equipe e os stakeholders. Isso significa que o trabalho é dividido em pequenos incrementos, cada um dos quais é desenvolvido, testado e revisado antes de avançar.

Na prática, iterações curtas, geralmente chamadas de sprints (em metodologias como Scrum), são usadas para criar versões incrementais e funcionais do produto. Ao final de cada sprint, uma revisão é realizada para demonstrar o progresso, obter feedback e ajustar o curso do projeto se necessário.

Isso cria um ciclo contínuo de melhoria e garante que o produto final evolua de acordo com as necessidades reais do usuário.

Autonomia

Descrição: na gestão ágil, as equipes são incentivadas a serem autônomas. Isso significa que elas têm a liberdade e a responsabilidade de tomar decisões sobre como o trabalho será realizado, sem a necessidade de aprovações ou supervisão constante.

Na prática, a autonomia é promovida através de uma clara definição de papéis e responsabilidades, como Scrum Master, Product Owner, e equipe de desenvolvimento. Cada membro da equipe tem a liberdade de escolher a melhor maneira de realizar suas tarefas, dentro das diretrizes gerais do projeto. Isso não apenas acelera a tomada de decisões, mas também aumenta a motivação e o comprometimento da equipe.

Empoderamento de Equipes

Descrição: o empoderamento das equipes é crucial para o sucesso na gestão ágil. Equipes empoderadas têm o poder de tomar decisões importantes, sugerir melhorias e influenciar diretamente o rumo do projeto.

Na prática, empoderar uma equipe significa fornecer a ela os recursos, informações e autoridade necessárias para cumprir seus objetivos. Isso pode incluir acesso a ferramentas, treinamento, ou apoio da liderança. O empoderamento também envolve a criação de um ambiente onde a equipe se sinta segura para experimentar, falhar e aprender com seus erros. Líderes ágeis atuam como facilitadores, removendo obstáculos e apoiando a equipe, em vez de microgerenciá-la.

Programação em Pares

Descrição: prática de desenvolvimento de software onde duas pessoas trabalham juntas no mesmo código em um único computador. Um dos programadores, chamado de "piloto", escreve o código, enquanto o outro, o "navegador", revisa cada linha de código à medida que é escrita, pensando em melhorias, erros potenciais e considerando o impacto geral da solução.

Na prática, a programação em pares promove a colaboração, a melhoria contínua e a qualidade do código. Embora possa parecer menos eficiente inicialmente, devido ao envolvimento de dois desenvolvedores em uma única tarefa, os benefícios de um código mais limpo, menos erros e maior troca de conhecimento podem compensar essa percepção, resultando em um desenvolvimento de software mais eficaz e sustentável.

Na gestão ágil de projetos, cooperação, flexibilidade de escopo, interatividade, autonomia e empoderamento de equipes são elementos fundamentais que trabalham juntos para promover uma abordagem mais adaptável e orientada para o valor. Essas práticas não apenas aumentam a eficiência e a eficácia do processo de desenvolvimento, mas também criam um ambiente onde as equipes podem prosperar, inovar e entregar produtos de alta qualidade que atendem às necessidades reais dos clientes.

TÉCNICAS P/ PLANEJAMENTO DE ESCOPO EM PROJETOS ÁGEIS

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Técnicas para planejamento e priorização incremental de escopo em projetos ágeis! *Quem aí já tinha ouvido falar nisso?* Bem... o nome é grande, mas é bastante simples. Galera, **quem já trabalhou com projeto de software sabe como é difícil, custoso, complexo, trabalhoso, chato, árduo e ingrato fazer com que o cliente consiga priorizar suas necessidades**. Em geral, clientes acreditam que tudo é importante!

Isso dificulta bastante os trabalhos do projeto – **tem de haver uma priorização e uma estimativa do escopo**. Grosso modo, requisitos no topo da lista ordenada de prioridade são mais importantes e devem ser desenvolvidos primeiro, já quem está na base é menos importante (não quer dizer que não tenham importância). Recomenda-se que cada item do escopo tenha um tamanho mensurável ou estimável! *Como assim, professor? Qual a diferença?*

Bem, mensurável pode ser medido com precisão e estimável é apenas uma estimativa. Respondam-me: *qual desses é mais provável que seja a escolha de uma equipe de desenvolvimento de projetos ágeis?!* A resposta é óbvia: a segunda opção! Um pensamento comum e errôneo entre diversos profissionais é que projetos ágeis sejam incompatíveis com planejamento. Eles imaginam: *se eu planejar, não consigo ser ágil*!

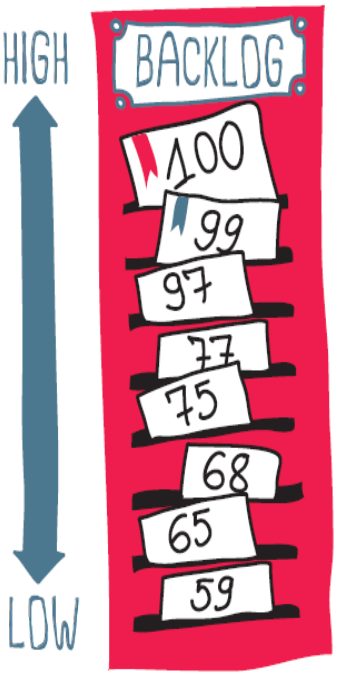
Ora, em projetos ágeis planeja-se, muitas vezes, até mais que em projetos não-ágeis. **Não se desencoraja planejamento, na verdade encoraja-se o contínuo replanejamento**. Em contraste, um projeto tradicional enfatiza um planejamento detalhado feito antes de investimentos em tempo e recursos. Este plano inicial tenta traçar o curso inteiro do projeto, buscando explicar a lista completa dos requisitos previstos para todas as fases de desenvolvimento.

Vamos às técnicas: a primeira delas é a Priorização Binária – essa técnica pode ser usada para ordenar o escopo inteiro de projetos e suas atividades da prioridade mais alta à prioridade mais baixa. *Como?* **Muito simples: escolhe-se dois itens e o cliente deve dizer qual é mais importante. Repete-se iterativamente esse processo até ordenar todos os itens**. Há também a Priorização baseada em Valor de Negócio.

Neste caso, cada requisito carrega um possível valor de negócio que pode gerar à organização. Esse valor é atribuído pelo dono do produto ou pelo chefe da equipe ágil. O requisito com maior valor de negócio é implementado prioritariamente no desenvolvimento. **Outra técnica bastante utilizada é denominada Priorização MoSCoW!** *Por que esse nome com letras maiúsculas e minúsculas?* Esse é o acrônimo para **Must Have**, **Should Have**, **Could Have** e **Won't Have**.

Em outras palavras, o cliente deve decidir o que o software deve ter, deveria ter, poderia ter e não terá (do maior para o menor nível de criticidade). **Há a Priorização Kano, em que se dividem os requisitos em Must-be, One-dimensional, Attractive, Indifferent e Reverse**. Há também técnicas de priorização baseadas nos requisitos de maior risco para a organização e as baseadas nos maiores riscos de implementação, por conta da tecnologia usada no desenvolvimento.

Todo mundo conhece o problema desse método: os clientes marcam Must Have em todos os itens do escopo, portanto se todos têm o mesmo nível de prioridade, não há priorização alguma! Existe outra técnica muito interessante, funciona mais ou menos assim: escolhe-se uma escala numérica arbitrária. Pode ser a Sequência de Fibonacci, Potências de Dois, Progressões Aritméticas, entre outros.

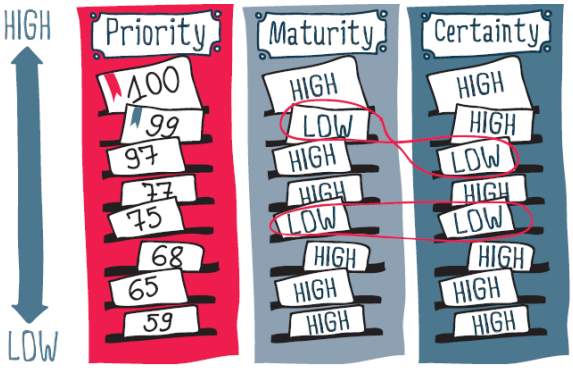


Pode-se inclusive inventar uma sequência – como mostra a imagem ao lado. Então, a equipe escolhe um item do escopo que tenha um tamanho médio e atribui um valor médio da escala escolhida. **Assim, itens do escopo serão comparados a esse item médio para se atribuir uma magnitude maior ou menor**.

Percebam que isso é uma estimativa realizada rapidamente – diferente de uma mensuração, que em geral necessita de um grande esforço. Equipes podem, então, usar essa informação para calcular a velocidade de desenvolvimento da equipe, melhorando a precisão de suas estimativas.

Quando um projeto começa, nem todos os requisitos estão prontos e levantados, portanto deve-se levar em consideração aspectos de incerteza e maturidade. Ora, alguns itens do escopo podem ter alta prioridade, no entanto podem não estar completamente definidos (maduros) ou válidos (incertos) – como mostra a imagem ao lado.

Dessa forma, as equipes buscam rastrear a certeza e maturidade de cada item do escopo para ajudá-los a monitorar quais itens estão prontos para serem desenvolvidos. Percebam que itens com baixa maturidade e certeza não são bons candidatos, mesmo que tenham alta prioridade. **A equipe deve passar algum tempo amadurecendo e validando o requisito antes de desenvolvê-lo**. *Bacana?* Esse é um tema que cai pouquíssima em prova ainda hoje...



(CODENI-RJ – 2010) Trata-se de um modo comum de aplicar a UML, frequentemente com alto retorno no investimento de tempo. Essa definição refere-se a:

a) Case

b) Objetos Syntropy
c) Modelagem Ágil
d) MDA

_____

Comentários: Craig Larman diz: "*Modelagem ágil enfatiza a UML como rascunho; trata-se de um modo comum de aplicar a UML, frequentemente com algo retorno no investimento de tempo (que é tipicamente curto). As ferramentas UML podem ser úteis, mas eu incentivo as pessoas a também considerar uma abordagem ágil de modelagem para aplicar a UML*" (Letra C).

TÉCNICAS DE ESTIMATIVA DE ESCOPO

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

A estimativa de escopo é um processo crítico no gerenciamento de projetos, envolvendo a definição e a documentação detalhada de todos os trabalhos necessários para completar um projeto com sucesso. O escopo do projeto abrange tanto os produtos ou entregáveis finais que devem ser produzidos quanto o trabalho necessário para criar esses entregáveis. *E o que seria exatamente um escopo?*

O escopo de um projeto define e delimita exatamente o que está e o que não está incluído no projeto. Ele abrange:

- **Entregáveis do Projeto:** os produtos, serviços ou resultados que serão entregues ao final do projeto.
- **Trabalho do Projeto:** todas as tarefas, atividades e processos necessários para criar esses entregáveis, incluindo tempo, recursos e processos envolvidos.

IMPORTÂNCIA	DESCRIÇÃO
CLAREZA E DIREÇÃO	Uma definição clara de escopo fornece uma compreensão comum do que o projeto visa alcançar, oferecendo direção para todos os envolvidos.
BASE PARA PLANEJAMENTO	O escopo é a base para o planejamento detalhado do projeto, incluindo cronograma, orçamento e alocação de recursos. Sem uma definição precisa do escopo, é impossível planejar eficazmente.
GERENCIAMENTO DE EXPECTATIVAS	Definir o escopo ajuda a gerenciar as expectativas das partes interessadas, garantindo que todos tenham a mesma compreensão dos objetivos do projeto e dos resultados esperados.
CONTROLE DE MUDANÇAS	Uma vez que o escopo está definido, qualquer alteração nos entregáveis ou no trabalho do projeto precisa passar por um processo formal de controle de mudanças. Isso ajuda a evitar o "Scope Creep" (Expansão Não Controlada do Escopo), que pode levar a atrasos, aumento de custos e outros problemas.
AVALIAÇÃO DE DESEMPENHO	O escopo estabelece critérios claros contra os quais o desempenho do projeto pode ser medido, permitindo avaliar se os objetivos do projeto estão sendo atendidos.

O gerenciamento eficaz do escopo é vital para o sucesso do projeto. Ele requer comunicação clara, documentação detalhada e uma abordagem colaborativa para garantir que o projeto seja entregue conforme planejado, dentro do prazo e do orçamento estabelecidos. **A estimativa de escopo**

obedece a um processo que contém as seguintes fases: Coleta de Requisitos; Definição de Escopo; Criação da Estrutura Analítica do Projeto (EAP) e Validação do Escopo.

PROCESSO	DESCRIÇÃO
COLETA DE REQUISITOS	Reunir informações das partes interessadas para entender as necessidades e expectativas.
DEFINIÇÃO DE ESCOPO	Documentar detalhadamente os entregáveis e o trabalho do projeto.
CRIAÇÃO DA EAP	Decompor o trabalho do projeto em componentes menores e mais gerenciáveis.
VALIDAÇÃO DO ESCOPO	Garantir que o escopo definido atenda às necessidades e expectativas das partes interessadas.

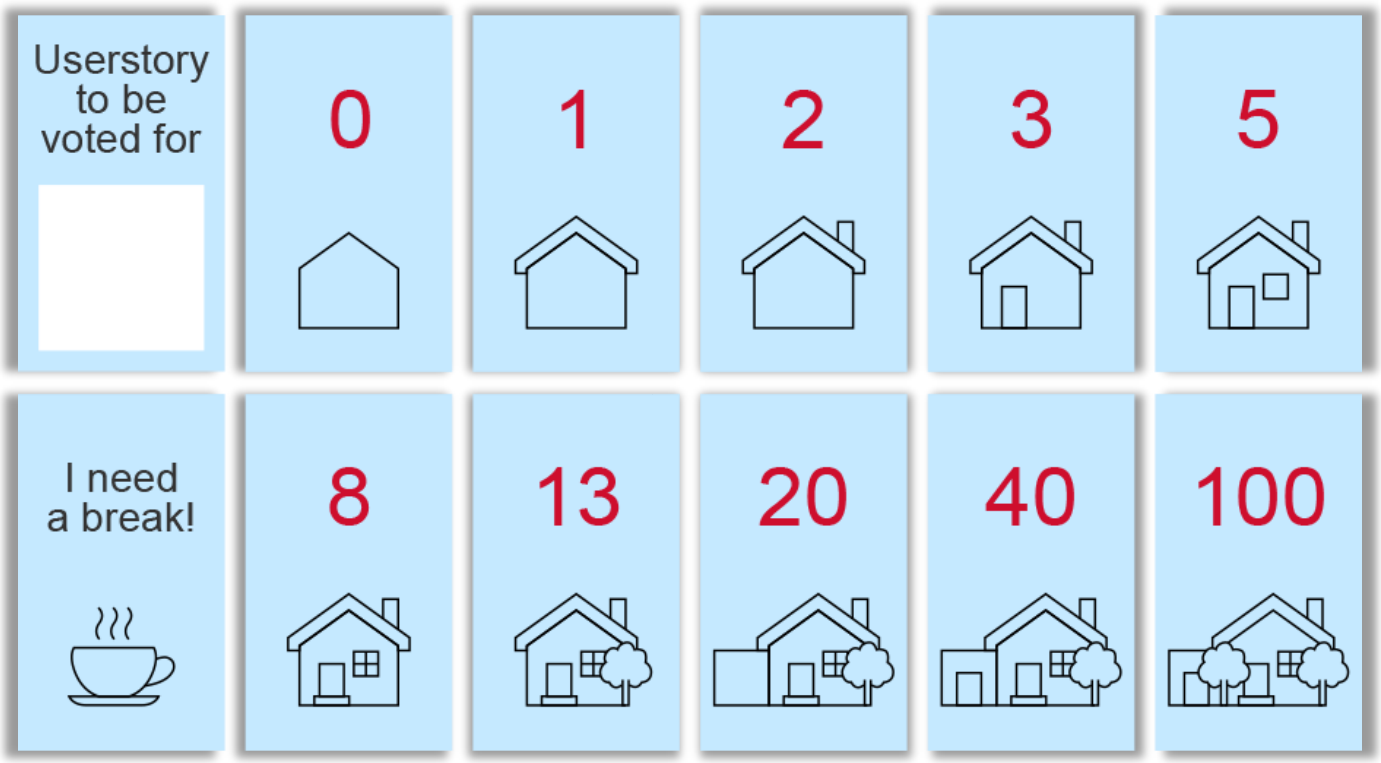
A estimativa de escopo é a estimativa de escopo é essencial para garantir que um projeto seja entregue com sucesso, dentro do prazo e do orçamento previstos. **Ela não apenas define o que precisa ser feito para alcançar os objetivos do projeto, mas também estabelece a base para o planejamento de recursos, tempo e custos.** Vejamos na tabela a seguir os principais objetivos da estimativa de escopo e, no próximo tópico, veremos as principais técnicas:

OBJETIVOS	DESCRIÇÃO
CLAREZA NOS RESULTADOS ESPERADOS	A estimativa de escopo ajuda a definir com clareza os entregáveis do projeto, garantindo que todas as partes interessadas tenham uma compreensão unificada do que o projeto visa alcançar.
ALINHAMENTO DE EXPECTATIVAS	Facilita o alinhamento das expectativas entre a equipe do projeto, as partes interessadas e os patrocinadores, minimizando mal-entendidos e conflitos potenciais.
BASE PARA CRONOGRAMA E ORÇAMENTO	Ao compreender o escopo do projeto, os gerentes podem desenvolver cronogramas de trabalho mais precisos e estimativas de custo, fundamentais para a alocação eficaz de tempo e recursos.
ALOCAÇÃO DE RECURSOS	Permite a identificação e alocação adequada dos recursos necessários – humanos, financeiros e materiais – para a execução do projeto.
IDENTIFICAÇÃO DE RISCOS	A análise detalhada do escopo ajuda a identificar potenciais riscos e desafios associados ao projeto, permitindo o desenvolvimento de estratégias de mitigação.
CONTROLE DE MUDANÇAS	Estabelece uma base para o gerenciamento de mudanças de escopo, ajudando a controlar o impacto de quaisquer alterações nos objetivos originais do projeto.
PREVENÇÃO DE SOBRECARGA DE TRABALHO (SCOPE CREEP)	Evita a expansão não planejada do escopo, que pode levar a atrasos, aumento de custos e desgaste da equipe.
OTIMIZAÇÃO DO PROCESSO DE TRABALHO	Facilita a identificação de redundâncias ou ineficiências no plano de trabalho, permitindo ajustes que otimizem a execução do projeto.
CRITÉRIOS PARA MEDIÇÃO DE SUCESSO	Fornece critérios claros contra os quais o sucesso do projeto pode ser medido, facilitando a avaliação de seu desempenho.

FERRAMENTA PARA COMUNICAÇÃO	Serve como uma ferramenta de comunicação eficaz, fornecendo uma descrição detalhada do trabalho a ser realizado para todas as equipes e partes interessadas envolvidas.

Planning Poker

INCIDÊNCIA EM PROVA: BAIXA



O Planning Poker, também conhecido como Scrum Poker, é uma técnica lúdica e colaborativa de estimativa de escopo usada principalmente em projetos de desenvolvimento de software, especialmente naqueles que adotam metodologias ágeis como o Scrum. Essa técnica incentiva a participação de toda a equipe no processo de estimativa, promovendo o entendimento comum do trabalho a ser realizado e ajudando a garantir que as estimativas sejam precisas e consensuais.

Nós vamos ver como é seu **funcionamento** e depois veremos um **exemplo concreto** de utilização dessa técnica. Vamos lá...

ETAPAS	DESCRIÇÃO
PREPARAÇÃO	Cada participante recebe um conjunto de cartas, que podem ser cartas reais ou digitais, com números representando diferentes níveis de esforço, geralmente sequências que seguem a série de Fibonacci (1, 2, 3, 5, 8, 13, 21, etc.), pois esta série reflete a incerteza inerente à estimativa de tarefas complexas.
APRESENTAÇÃO DOS ITENS	Um item de trabalho (por exemplo, uma história de usuário, requisito ou funcionalidade) é apresentado à equipe. O moderador, que pode ser o Scrum Master, explica os detalhes e objetivos do item.
DISCUSSÃO INICIAL	A equipe discute o item para esclarecer dúvidas. Esta etapa é crucial para garantir que todos tenham uma compreensão clara do que está sendo estimado.

ESTIMATIVA INDIVIDUAL	Cada membro da equipe seleciona uma carta de seu baralho que representa sua estimativa do esforço necessário para completar o item. As escolhas são feitas em segredo para evitar que as estimativas sejam influenciadas pelas escolhas dos outros.
RELEVAÇÃO	Todos os participantes revelam suas cartas simultaneamente. Se houver um consenso, essa estimativa é aceita. Se as estimativas variarem significativamente, os membros com as estimativas mais altas e mais baixas explicam seu raciocínio.
DISCUSSÃO E REESTIMATIVA	Após a discussão, a equipe realiza outra rodada de estimativas, repetindo o processo até alcançar um consenso ou uma aproximação aceitável.

EXEMPLO CONCRETO

Vamos ver um exemplo? **Imagine uma equipe de desenvolvimento de software que está prestes a começar um novo sprint em seu projeto.** O projeto envolve a criação de um aplicativo móvel para uma rede de restaurantes, permitindo que os clientes façam pedidos online. A equipe adota metodologias ágeis e está pronta para estimar o esforço necessário para implementar várias histórias de usuário planejadas para o próximo sprint. Agora vejam a seguinte história de usuário:

- **História de Usuário:** *"Como cliente, quero poder filtrar o menu por categorias de alimentos (como veganos, sem glúten, etc.), para que eu possa encontrar facilmente os pratos que atendem às minhas preferências alimentares".*

ETAPAS	DESCRIÇÃO
PREPARAÇÃO	A equipe se reúne (virtualmente ou fisicamente) com seus conjuntos de cartas de Planning Poker. O Scrum Master apresenta a história de usuário e esclarece os detalhes, objetivos e critérios de aceitação.
DISCUSSÃO INICIAL	A equipe discute a história. O desenvolvedor front-end menciona a necessidade de implementar um novo componente de interface para os filtros, enquanto o especialista em back-end fala sobre ajustar as consultas ao banco de dados para suportar a filtragem. O designer de UX/UI também contribui, destacando a importância de uma interface intuitiva para a seleção de filtros.
ESTIMATIVA INDIVIDUAL	Após a discussão, cada membro da equipe escolhe uma carta que representa sua estimativa de esforço. O desenvolvedor front-end escolhe 8, considerando o trabalho na interface. O back-end escolhe 5, pois já existe alguma infraestrutura que pode ser reaproveitada. O designer escolhe 3, acreditando que o design pode ser adaptado de componentes existentes.
RELEVAÇÃO	Todos na equipe revelam suas cartas simultaneamente. Com a discrepância nas estimativas, é claro que há uma necessidade de mais discussão.

DISCUSSÃO E REESTIMATIVA	O desenvolvedor front-end explica seu ponto de vista sobre os desafios técnicos, enquanto o designer oferece soluções que podem simplificar o trabalho. Após entenderem melhor os desafios uns dos outros, a equipe realiza uma segunda rodada de estimativas.
CONSENSO	Na segunda rodada, as estimativas são mais alinhadas. A equipe concorda com uma estimativa de 5 pontos de esforço para a história, considerando as soluções propostas e a colaboração entre as áreas.

Este exemplo mostra como o Planning Poker facilita a colaboração e o entendimento mútuo dentro da equipe, permitindo que diferentes perspectivas sejam compartilhadas e consideradas na estimativa do esforço necessário. Ao final, a equipe chega a um consenso informado, garantindo uma compreensão compartilhada do trabalho e promovendo um comprometimento coletivo com a execução da tarefa.

Story Points

INCIDÊNCIA EM PROVA: BAIXA

Story Points são uma unidade de medida utilizada em metodologias ágeis de gerenciamento de projetos para estimar o esforço necessário para completar uma história de usuário ou outro item de trabalho. Diferentemente de estimativas baseadas em tempo (Ex: horas ou dias), eles consideram fatores que podem afetar o esforço de trabalho, incluindo a complexidade da tarefa, volume de trabalho a ser feito, riscos potenciais e incertezas. Vejamos o seu funcionamento:

ETAPAS	DESCRIÇÃO
SESSÕES DE PLANEJAMENTO	Durante as reuniões de planejamento, a equipe discute cada história de usuário e atribui Story Points por meio de um consenso ou técnicas como Planning Poker.
AJUSTES COM BASE NA EXPERIÊNCIA	Conforme a equipe avança no projeto, ela se torna mais eficiente em estimar Story Points, ajustando suas estimativas com base na experiência adquirida em sprints anteriores.
MONITORAMENTO DA VELOCIDADE	A equipe acompanha a quantidade de Story Points completados em cada sprint, o que ajuda a prever a capacidade de trabalho para sprints futuros.

EXEMPLO CONCRETO

Vamos imaginar uma equipe de desenvolvimento de software que está utilizando a metodologia ágil Scrum para gerenciar o desenvolvimento de um novo aplicativo de finanças pessoais. A equipe está prestes a realizar sua sessão de planejamento para o próximo sprint e precisa estimar o esforço necessário para completar as histórias de usuário selecionadas. Vejamos três histórias de usuário para estimativa:

- **História de Usuário 1:** *"Como usuário, quero poder categorizar minhas despesas, para que eu possa visualizar meu orçamento por categorias".*
- **História de Usuário 2:** *"Como usuário, quero receber notificações sobre transações suspeitas, para que eu possa garantir a segurança da minha conta".*
- **História de Usuário 3:** *"Como usuário, quero poder exportar relatórios mensais das minhas despesas para PDF, para que eu possa arquivá-las para referência futura".*

Inicialmente, a equipe discute cada história de usuário para esclarecer os requisitos e avaliar a complexidade, o volume de trabalho e os possíveis riscos associados. Vejamos:

- Para a **História de Usuário 1**, a equipe considera que a implementação envolverá tanto o desenvolvimento de uma nova interface de usuário quanto ajustes no back-end para suportar a categorização. Eles avaliam essa história como tendo uma complexidade média.

- A **História de Usuário 2** requer a implementação de um sistema de detecção de anomalias e notificações push. A equipe considera isso mais desafiador devido à necessidade de algoritmos para detectar transações suspeitas.

- Para a **História de Usuário 3**, a equipe identifica a necessidade de implementar uma funcionalidade de exportação para PDF, que eles já têm experiência anterior, considerando-a relativamente simples em comparação com as outras histórias.

Em seguida, a equipe realiza uma rodada de Planning Poker para estimar o esforço de cada história em Story Points. Vejamos:

- **História de Usuário 1** é avaliada em 8 Story Points, refletindo sua complexidade média e o trabalho envolvido.

- **História de Usuário 2** recebe 13 Story Points devido à sua complexidade técnica e ao esforço necessário para implementar a funcionalidade de detecção de transações suspeitas.

- **História de Usuário 3** é estimada em 5 Story Points, visto que a equipe já possui experiência na implementação de funcionalidades semelhantes.

Com base nas estimativas de **Story Points** e na capacidade de trabalho conhecida da equipe (sua velocidade), o Scrum Master e a equipe decidem quais histórias serão incluídas no próximo sprint.

Este exemplo ilustra como a técnica de Story Points permite à equipe de desenvolvimento fazer estimativas informadas sobre o esforço necessário para completar diferentes histórias de usuário. Ao considerar a complexidade, o volume de trabalho e os riscos, em vez de simplesmente o tempo, a equipe pode priorizar melhor o trabalho e planejar seus sprints de forma mais eficaz, garantindo uma abordagem balanceada para alcançar seus objetivos.

Enquetes

INCIDÊNCIA EM PROVA: BAIXA

Essa técnica é utilizada para coletar opiniões e alcançar um consenso entre especialistas sobre questões específicas, incluindo a estimativa do escopo de um projeto. Embora o termo "Enquetes" possa não ser um termo técnico padrão no gerenciamento de projetos, o conceito se alinha com a abordagem de consultar um grupo diversificado de partes interessadas ou especialistas para informar e refinar estimativas de projeto.

Nós vamos explorar como esse conceito é aplicado na prática, particularmente por meio do Método Delphi. Esse método é uma técnica de pesquisa e estimativa estruturada que visa obter um consenso entre especialistas através de rodadas de questionários anônimos. **É utilizado em várias áreas, incluindo previsão de tecnologia, determinação de prioridades de políticas públicas e, claro, estimativa de escopo em gerenciamento de projetos.** *E como funciona?*

ETAPAS	DESCRIÇÃO
SELEÇÃO DOS ESPECIALISTAS	Um grupo diversificado de especialistas é selecionado. Esses especialistas devem ter conhecimento relevante sobre o tópico ou projeto em questão.
RODADAS DE QUESTIONÁRIOS	São enviados questionários anônimos aos especialistas. Após a primeira rodada de respostas, um resumo das estimativas é preparado e compartilhado com o grupo.
FEEDBACK E ITERAÇÃO	Os especialistas revisam as respostas dos colegas (ainda de forma anônima) e têm a oportunidade de revisar suas próprias estimativas na rodada seguinte de questionários, considerando as perspectivas dos demais.
CONVERGÊNCIA E CONSENSO	O processo é repetido até que um consenso seja alcançado ou as opiniões comecem a convergir para uma faixa comum de respostas.

Nesse sentido, o anonimato incentiva a expressão honesta de opiniões, reduzindo o impacto da influência ou da autoridade de indivíduos específicos no grupo. Além disso, a estrutura iterativa e a exposição a diversas opiniões ajudam a reduzir o viés individual e a influência de outliers nas estimativas. Ao final, a equipe de projeto ganha uma estimativa de escopo informada pelo consenso de especialistas, refletindo a compreensão ampla e considerada do trabalho necessário.

O Método Delphi pode ser demorado e requer um planejamento cuidadoso para gerenciar as várias rodadas de questionários e análise. A eficácia da técnica depende da seleção apropriada de especialistas com conhecimento e experiência relevantes. Embora tradicionalmente utilizado para previsões e estimativas, o Método Delphi pode ser adaptado para uma ampla gama de decisões de planejamento e avaliação em gerenciamento de projetos.

EXEMPLO CONCRETO

Vamos considerar um exemplo prático do uso da técnica de Enquetes, inspirada no Método Delphi, em um projeto de desenvolvimento de um novo sistema de informação hospitalar. Este sistema tem como objetivo integrar diferentes departamentos do hospital, desde a recepção até a alta do paciente, incluindo prontuários eletrônicos, gestão de estoque de medicamentos e agendamento de consultas. *E qual é o contexto?*

O hospital deseja estimar o escopo e os recursos para o desenvolvimento desse sistema, levando em conta a complexidade técnica, as necessidades dos usuários finais e a conformidade regulatória.

ETAPAS	DESCRIÇÃO
SELEÇÃO DOS ESPECIALISTAS	A equipe de projeto seleciona um grupo diversificado de especialistas, incluindo desenvolvedores de software com experiência em sistemas hospitalares, gerentes de TI do hospital, médicos, enfermeiros, e um especialista em conformidade regulatória.
PRIMEIRA RODADA DE QUESTIONÁRIOS	A equipe de projeto envia um questionário anônimo aos especialistas, pedindo-lhes para estimar o esforço necessário para desenvolver o sistema, considerando as diferentes funcionalidades e requisitos regulatórios. Os especialistas são encorajados a fornecer justificativas para suas estimativas.
COMPILAÇÃO E FEEDBACK	As respostas são compiladas e um resumo é enviado de volta aos especialistas. Este resumo inclui a faixa de estimativas fornecidas, as justificativas e qualquer comentário relevante, sem identificar os autores.
SEGUNDA RODADA DE QUESTIONÁRIOS	Os especialistas revisam o resumo e são solicitados a refinar suas estimativas iniciais, levando em consideração as informações compartilhadas. Eles têm a opção de ajustar suas estimativas ou mantê-las, com justificativas adicionais, se necessário.
CONVERGÊNCIA E CONSENSO	O processo é repetido por várias rodadas até que as estimativas comecem a convergir para uma faixa comum ou um consenso claro seja alcançado. No nosso exemplo, um consenso é alcançado após três rodadas, com a equipe concordando com uma estimativa detalhada do escopo e dos recursos necessários.

O resultado é uma estimativa de escopo bem fundamentada que reflete o consenso entre especialistas de diferentes áreas. Isso oferece à equipe de projeto uma base sólida para o planejamento detalhado do projeto, incluindo alocação de recursos, cronograma e orçamento. Além disso, o processo ajuda a identificar áreas de potencial risco e incerteza, bem como oportunidades para simplificar ou modularizar o sistema para facilitar o desenvolvimento.

Este exemplo ilustra como a técnica pode ser aplicada para estimar o escopo de projetos complexos, aproveitando a experiência e o conhecimento coletivo de um grupo de especialistas.

T-Shirt Sizing

INCIDÊNCIA EM PROVA: BAIXA

Essa técnica de estimativa de escopo é um método simples e intuitivo frequentemente utilizado em ambientes de desenvolvimento ágil para estimar rapidamente o tamanho relativo de tarefas ou histórias de usuário. Essa técnica compara tarefas a tamanhos de camisetas: XS (Extra Small), S (Small), M (Medium), L (Large), XL (Extra Large), e às vezes até maiores como XXL, para representar a complexidade, o esforço ou o tempo necessário para completá-las.

O objetivo é facilitar discussões iniciais sobre escopo e complexidade sem se prender a detalhes exatos ou tentar quantificar o trabalho em horas ou dias. *E como funciona?*

ETAPAS	DESCRIÇÃO
DISCUSSÃO INICIAL	A equipe reúne-se para discutir as características de cada tarefa ou história de usuário. Essas discussões focam na compreensão do trabalho necessário, possíveis desafios e complexidades envolvidas.
CLASSIFICAÇÃO RELATIVA	Baseando-se na compreensão adquirida durante a discussão, a equipe classifica cada tarefa atribuindo-lhe um "tamanho de camiseta". Este processo estimula a equipe a pensar sobre o trabalho de forma relativa às outras tarefas, ao invés de tentar atribuir valores absolutos de início.
REVISÃO E AJUSTE	As classificações podem ser revistas e ajustadas conforme necessário, especialmente se novas informações vierem à tona ou se a equipe perceber que comparou erroneamente as tarefas entre si.

Após a classificação inicial, a equipe pode optar por converter os tamanhos de camisetas em estimativas mais quantificáveis, como Story Points ou horas, se necessário para o planejamento detalhado. Esse passo adicional pode se basear na experiência anterior da equipe ou em discussões adicionais para definir a equivalência entre os tamanhos de camisetas e os valores numéricos.

Enquanto o T-Shirt Sizing é útil para estimativas rápidas e discussões iniciais, ele pode precisar ser complementado por técnicas de estimativa mais detalhadas à medida que o projeto avança

e mais informações se tornam disponíveis. O método é particularmente eficaz em ambientes que valorizam a flexibilidade e a capacidade de adaptar planos conforme o projeto evolui. Vejamos – portanto – um exemplo prático de utilização dessa técnica para o projeto de um aplicativo móvel.

EXEMPLO CONCRETO

O objetivo do projeto é desenvolver um aplicativo de entrega de comida que permita aos usuários navegar por restaurantes locais, fazer pedidos e acompanhar as entregas em tempo real. A equipe se reúne para uma sessão de planejamento. O Product Owner (PO) apresenta uma lista de histórias de usuário preliminares necessárias para o lançamento mínimo viável (MVP) do aplicativo. As histórias incluem:

- **Pedido do Usuário:** permitir que o usuário selecione itens do menu de um restaurante e faça um pedido.
- **Rastreamento de Entrega em Tempo Real:** mostrar ao usuário um mapa em tempo real com a localização do entregador.
- **Cadastro de Usuário:** deve-se permitir que os usuários possam criar uma conta no aplicativo móvel.
- **Avaliações e Comentários:** permitir que os usuários avaliem os restaurantes e deixem comentários sobre suas refeições.

Em seguida, a equipe discute cada história de usuário para entender suas necessidades e complexidades. Em seguida, eles estimam o tamanho de cada história usando **T-Shirt Sizing**:

- **Pedido do Usuário - L (Large):** considerada de grande porte devido à necessidade de integrar um sistema de carrinho de compras, interação com o sistema de pagamento e confirmação de pedido.
- **Rastreamento de Entrega em Tempo Real - XL (Extra Large):** avaliada como a mais complexa devido à necessidade de integração com serviços de localização em tempo real e atualizações constantes de posição.
- **Cadastro de Usuário - M (Medium):** trata-se de uma funcionalidade relativamente padrão, porém que requer integração com sistemas de autenticação e armazenamento de dados seguros.
- **Avaliações e Comentários - S (Small):** trata-se de uma funcionalidade considerada a menos complexa, envolvendo a criação de uma interface para submissão de avaliações e o armazenamento desses dados.

Com base nas estimativas de tamanho, a equipe e o Product Owner (PO) priorizam as histórias para o MVP (Minimum Viable Product). Eles decidem começar com as histórias classificadas como S (Small) e M (Medium) para garantir um progresso rápido e deixam o rastreamento de entrega em tempo real (XL) para uma fase posterior do desenvolvimento, após uma avaliação mais detalhada da complexidade e dos recursos necessários.

Este exemplo demonstra como a técnica de T-Shirt Sizing pode ser usada para facilitar a discussão sobre complexidade relativa e esforço necessário para diferentes histórias de usuário em um projeto ágil. Ajuda a equipe a priorizar o trabalho com base em uma compreensão comum da escala de esforço, sem se prender a estimativas numéricas detalhadas desde o início, permitindo um planejamento flexível e iterativo.

QUESTÕES COMENTADAS

1. **(CESPE / INPI – 2024)** Na gestão do backlog de um produto usando-se uma metodologia ágil, o controle do versionamento dos artefatos é adequado para procurar manter o backlog com a característica de emergente.

Comentários:

Para responder à questão, é importante entender que "emergente" foi colocado no sentido de "em constante desenvolvimento". Nesse sentido, o controle de versionamento de artefatos é realmente adequado para manter o backlog em constante evolução e desenvolvimento.

Gabarito: Correto

---

2. **(PROF. DIEGO / INÉDITA – 2024)** O escopo de um projeto inclui apenas os entregáveis finais, não abrangendo o trabalho necessário para criá-los.

Comentários:

O escopo de um projeto abrange tanto os entregáveis finais quanto o trabalho necessário para criá-los. Ele define não apenas o que será entregue ao final do projeto (os produtos, serviços ou resultados esperados), mas também inclui todas as tarefas, atividades e processos necessários para produzir esses entregáveis. O escopo é fundamental para um planejamento de projeto eficaz, pois ajuda a determinar e comunicar o que está incluído no projeto e o que está fora dele, guiando o trabalho da equipe do projeto e as expectativas das partes interessadas.

Gabarito: Errado

---

3. **(PROF. DIEGO / INÉDITA – 2024)** Uma definição clara de escopo fornece direção e facilita a compreensão comum dos objetivos do projeto entre todas as partes interessadas.

Comentários:

Definir o escopo do projeto ajuda a delinear o que será realizado, estabelecendo os limites do que está incluído e o que está excluído do projeto. Isso é crucial para gerenciar expectativas, evitar mal-entendidos e garantir que todos os envolvidos tenham uma visão alinhada dos objetivos e entregáveis. Uma comunicação clara e uma documentação detalhada do escopo contribuem para a coordenação eficaz entre as equipes, a alocação apropriada de recursos e a minimização de alterações no escopo, conhecidas como "Scope Creep", que podem afetar o cronograma e o orçamento do projeto.

Gabarito: Correto

---

4. **(PROF. DIEGO / INÉDITA – 2024)** A estimativa de escopo é uma etapa dispensável no processo de gerenciamento de projetos, uma vez que os detalhes podem ser ajustados conforme o desenvolvimento do projeto.

Comentários:

A estimativa de escopo é essencial no gerenciamento de projetos, pois define a extensão do trabalho, orienta o planejamento de recursos, o orçamento e o cronograma. Sem ela, o projeto pode enfrentar desvios de escopo, atrasos e custos adicionais, comprometendo o sucesso do projeto.

Gabarito: Errado

---

5. **(PROF. DIEGO / INÉDITA – 2024)** A criação da Estrutura Analítica do Projeto (EAP) é um processo que visa decompor o trabalho do projeto em componentes menores e mais gerenciáveis.

Comentários:

A criação da Estrutura Analítica do Projeto (EAP) é um processo fundamental no gerenciamento de projetos que visa decompor o trabalho do projeto em componentes menores e mais gerenciáveis. Isso facilita o planejamento, a execução, o monitoramento e o controle do projeto, contribuindo para uma melhor organização e compreensão do escopo total do trabalho a ser realizado.

Gabarito: Correto

---

6. **(PROF. DIEGO / INÉDITA – 2024)** O controle de mudanças não é necessário uma vez que o escopo do projeto tenha sido definido e validado pelas partes interessadas.

Comentários:

Mesmo após a definição e validação do escopo do projeto pelas partes interessadas, o controle de mudanças é necessário. Ele permite gerenciar de forma eficaz quaisquer alterações no escopo, garantindo que todas as modificações sejam avaliadas, aprovadas e documentadas adequadamente. Isso ajuda a evitar o "Scope Creep" e a manter o projeto alinhado com seus objetivos originais, prazos e orçamento.

Gabarito: Errado

---

7. **(PROF. DIEGO / INÉDITA – 2024)** A validação do escopo é uma fase em que se verifica se os entregáveis do projeto atendem às necessidades e expectativas das partes interessadas.

Comentários:

A validação do escopo é o processo de verificação e confirmação de que os entregáveis do projeto atendem às necessidades e expectativas das partes interessadas. Essa fase é crucial para garantir a satisfação do cliente e o sucesso do projeto, envolvendo a revisão dos entregáveis antes da aceitação final.

Gabarito: Correto

---

8. **(PROF. DIEGO / INÉDITA – 2024)** A estimativa de escopo não contribui para a identificação e alocação de recursos necessários para a execução do projeto.

Comentários:

A estimativa de escopo desempenha um papel crucial na identificação e alocação de recursos necessários para a execução do projeto. Ao definir a extensão do trabalho e as atividades específicas necessárias, a estimativa de escopo permite que os gerentes de projeto planejem adequadamente quais recursos (humanos, materiais, financeiros) serão necessários, bem como quando e onde serão utilizados, contribuindo diretamente para a eficiência e sucesso do projeto.

Gabarito: Errado

---

9. **(PROF. DIEGO / INÉDITA – 2024)** A estimativa de escopo ajuda a prevenir a expansão não planejada do escopo, conhecida como "Scope Creep", que pode levar a atrasos e aumento de custos.

Comentários:

A estimativa de escopo é fundamental para prevenir a expansão não planejada do escopo, conhecida como "Scope Creep". Ao definir claramente o escopo do projeto desde o início e estabelecer um processo formal para o controle de mudanças, as equipes podem gerenciar eficazmente as alterações no escopo, evitando atrasos e aumento de custos que podem ocorrer quando o trabalho adicional não é devidamente planejado ou controlado.

Gabarito: Correto

---

10. **(PROF. DIEGO / INÉDITA – 2024)** O que é "Scope Creep"?

a) Uma técnica de gerenciamento de riscos.
b) Uma metodologia de definição de escopo.
c) Um processo formal de controle de mudanças.
d) A expansão não controlada do escopo do projeto.
e) Uma ferramenta para comunicação eficaz.

Comentários:

(a) Errado. "Scope Creep" não é uma técnica de gerenciamento de riscos, mas sim um fenômeno que pode ser um risco para o gerenciamento do projeto;

(b) Errado. Embora esteja relacionado ao escopo, "Scope Creep" não é uma metodologia de definição de escopo; trata-se da alteração não planejada deste;

(c) Errado. Não é um processo formal de controle de mudanças. Pelo contrário, ocorre justamente quando o controle de mudanças é ineficaz ou inexistente;

(d) Correto. "Scope Creep" refere-se à expansão não controlada do escopo de um projeto, frequentemente ocorrendo sem a devida consideração das implicações em termos de tempo, custos ou recursos;

(e) Errado. "Scope Creep" não é uma ferramenta para comunicação eficaz, mas um problema que pode surgir devido à falta de comunicação clara sobre o escopo do projeto.

Gabarito: Letra D

---

11. (PROF. DIEGO / INÉDITA – 2024) O que a criação da Estrutura Analítica do Projeto (EAP) facilita?

a) A comunicação entre as equipes.
b) O controle de mudanças.
c) A identificação de redundâncias no trabalho.
d) A decomposição do trabalho do projeto.
e) A validação do escopo com as partes interessadas.

Comentários:

(a) Errado. Embora a EAP possa indiretamente melhorar a comunicação ao clarificar os elementos do projeto, esta não é sua principal função;

(b) Errado. O controle de mudanças é facilitado pela EAP ao definir claramente os entregáveis, mas essa não é a principal função da EAP;

(c) Errado. A identificação de redundâncias pode ser um benefício indireto da criação da EAP, porém não é o principal objetivo da sua criação;

(d) Correto. A principal função da EAP é facilitar a decomposição do trabalho do projeto em partes menores e mais gerenciáveis, o que ajuda na organização e planejamento do projeto;

(e) Errado. Embora a validação do escopo com as partes interessadas seja importante, este processo é mais amplo e envolve outras ferramentas e técnicas além da EAP.

Gabarito: Letra D

---

12. (PROF. DIEGO / INÉDITA – 2024) Qual dos seguintes NÃO é um objetivo da estimativa de escopo?

a) Prevenção de "Scope creep".
b) Facilitação da comunicação entre as partes interessadas.
c) Redução do tempo necessário para completar o projeto.
d) Identificação e alocação de recursos necessários.
e) Alinhamento de expectativas entre a equipe e os patrocinadores.

Comentários:

(a) Errado. A prevenção de "Scope Creep" é, de fato, um objetivo da estimativa de escopo, pois ao estimar corretamente, pode-se evitar a expansão não controlada do escopo;

(b) Errado. A facilitação da comunicação entre as partes interessadas é um objetivo da estimativa de escopo, pois proporciona uma base comum para discussões;

(c) Correto. A redução do tempo necessário para completar o projeto não é um objetivo direto da estimativa de escopo. A estimativa de escopo visa entender a magnitude e os requisitos do projeto, mas não necessariamente reduz o tempo de conclusão do projeto; este aspecto depende de outros fatores, como a eficiência da execução e gestão do projeto;

(d) Errado. A identificação e alocação de recursos necessários é um dos objetivos da estimativa de escopo, garantindo que os recursos adequados estejam disponíveis para o projeto;

(e) Errado. O alinhamento de expectativas entre a equipe e os patrocinadores é um objetivo da estimativa de escopo, pois esclarece o que será entregue, contribuindo para a gestão de expectativas.

Gabarito: Letra C

---

13. (PROF. DIEGO / INÉDITA – 2024) Planning Poker é uma técnica de estimativa que se baseia unicamente na intuição individual de cada membro da equipe, sem necessidade de discussões em grupo.

Comentários:

O Planning Poker é uma técnica de estimativa que envolve discussões em grupo e não se baseia unicamente na intuição individual. Durante o processo, cada membro da equipe usa cartas para votar na estimativa de esforço para uma tarefa ou história, mas antes da votação, há uma discussão coletiva para esclarecer dúvidas e compartilhar entendimentos. As rodadas de votação e discussão continuam até que a equipe chegue a um consenso, fazendo do Planning Poker uma abordagem colaborativa para estimativa.

Gabarito: Errado

---

14. (PROF. DIEGO / INÉDITA – 2024) Story Points consideram a complexidade, o volume de trabalho, os riscos potenciais e incertezas, ao contrário das estimativas baseadas apenas em tempo.

Comentários:

Story Points são uma técnica de estimativa que considera múltiplos fatores, como a complexidade da tarefa, o volume de trabalho, os riscos potenciais e as incertezas, ao invés de se basear apenas em tempo. Isso permite uma compreensão mais holística e relativa do esforço necessário para completar as tarefas ou histórias de usuário, facilitando o planejamento e a gestão de projetos, especialmente em ambientes ágeis.

Gabarito: Correto

---

15. (PROF. DIEGO / INÉDITA – 2024) O Método Delphi é caracterizado pela falta de anonimato nas respostas, permitindo que os participantes se influenciem mutuamente.

Comentários:

O Método Delphi é caracterizado justamente pelo anonimato nas respostas dos participantes. Esse método envolve rodadas de questionários enviados a especialistas, onde as respostas são coletadas e sumarizadas anonimamente pelo facilitador. O objetivo é evitar a influência direta entre os participantes, permitindo que cada um forneça sua opinião ou estimativa de forma independente. Isso contribui para a obtenção de um consenso ou entendimento coletivo mais imparcial sobre determinado assunto.

Gabarito: Errado

---

16. (PROF. DIEGO / INÉDITA – 2024) T-Shirt Sizing é uma técnica de estimativa que atribui valores numéricos exatos às tarefas.

Comentários:

T-Shirt Sizing é uma técnica de estimativa que não atribui valores numéricos exatos às tarefas. Em vez disso, utiliza categorias representadas por tamanhos de camisetas (como XS, S, M, L, XL) para indicar o tamanho relativo ou a complexidade de uma tarefa ou história de usuário. Essa abordagem ajuda a facilitar discussões rápidas e a chegar a um consenso sobre a estimativa de escopo sem se aprofundar em cálculos detalhados de tempo ou esforço.

Gabarito: Errado

---

17. (PROF. DIEGO / INÉDITA – 2024) No Planning Poker, se as estimativas iniciais variarem significativamente, o consenso deve ser atingido sem discussão adicional.

Comentários:

No Planning Poker, se as estimativas iniciais variarem significativamente, o processo encoraja justamente a discussão adicional entre os membros da equipe. Aqueles com as estimativas mais altas e mais baixas são convidados a explicar seus raciocínios, promovendo um entendimento mais profundo das tarefas em questão. Esse diálogo visa facilitar a convergência para um consenso informado sobre a estimativa de esforço, em vez de buscar um acordo sem a devida discussão e compreensão mútua.

Gabarito: Errado

---

18. (PROF. DIEGO / INÉDITA – 2024) A técnica de T-Shirt Sizing pode ser utilizada em qualquer fase do projeto para ajudar a priorizar tarefas.

Comentários:

A técnica de T-Shirt Sizing pode ser utilizada em qualquer fase do projeto para ajudar a priorizar tarefas. Por ser uma forma rápida e intuitiva de estimar o tamanho relativo ou a complexidade das tarefas, ela facilita a tomada de decisões sobre a priorização sem a necessidade de entrar em detalhes exatos de tempo ou esforço. Isso a torna útil não apenas nas fases iniciais de planejamento, mas também ao longo do projeto, à medida que novas tarefas surgem ou quando é necessário reavaliar prioridades.

Gabarito: Correto

---

19. (PROF. DIEGO / INÉDITA – 2024) A estimativa de Story Points se torna menos precisa à medida que a equipe ganha mais experiência no projeto.

Comentários:

Na verdade, Story Points tende a se tornar mais precisa à medida que a equipe ganha mais experiência no projeto. À medida que os membros da equipe se familiarizam com o trabalho, o

domínio do projeto e a dinâmica de colaboração, eles desenvolvem uma melhor compreensão da complexidade relativa das tarefas e como elas se comparam entre si. Isso permite que as estimativas de Story Points reflitam mais de perto o esforço necessário para completar as tarefas, resultando em uma capacidade de planejamento e execução do projeto mais eficaz.

Gabarito: Errado

---

20. (PROF. DIEGO / INÉDITA – 2024) No Método Delphi, a convergência para um consenso ocorre após uma única rodada de questionários.

Comentários:

O Método Delphi geralmente envolve várias rodadas de questionários para alcançar a convergência para um consenso. Após cada rodada, as respostas são compiladas e sumarizadas, e as informações agregadas são compartilhadas com os participantes. Isso permite que eles revisem suas estimativas iniciais com base no feedback coletivo, promovendo uma compreensão mais aprofundada e ajustes nas estimativas. O processo é repetido até que as opiniões comecem a convergir e um consenso ou um entendimento comum mais claro seja alcançado, o que raramente ocorre após apenas uma rodada.

Gabarito: Errado

---

21. (PROF. DIEGO / INÉDITA – 2024) Qual das seguintes opções melhor descreve a finalidade do Planning Poker?

a) Determinar a duração exata do projeto.
b) Estimular a competição entre os membros da equipe.
c) Facilitar estimativas de esforço consensuais e baseadas em discussão.
d) Alocar tarefas específicas a membros da equipe.
e) Priorizar as histórias de usuário com base em sua importância.

Comentários:

(a) Errado. Planning Poker não é usado para determinar a duração exata do projeto, mas sim para estimar o esforço ou complexidade das tarefas

(b) Errado. Planning Poker é uma técnica colaborativa; seu objetivo não é estimular a competição, mas promover a discussão e o consenso;

(c) Correto. A finalidade do Planning Poker é facilitar estimativas de esforço consensuais e baseadas em discussão entre os membros da equipe, utilizando cartas para votar na estimativa de esforço de tarefas ou histórias de usuário;

(d) Errado. Planning Poker não é utilizado para alocar tarefas específicas a membros da equipe, mas para estimar o esforço necessário para completar as tarefas;

(e) Errado. Embora as estimativas possam ajudar a priorizar histórias de usuário ao revelar sua complexidade, a principal função do Planning Poker não é priorizar, mas estimar o esforço.

Gabarito: Letra C

---

22. (PROF. DIEGO / INÉDITA – 2024) No contexto de metodologias ágeis, o que Story Points ajudam a estimar?

a) O custo total do projeto.
b) O número de membros necessários na equipe.
c) O esforço necessário para completar uma história de usuário.
d) A duração total de cada sprint.
e) A ordem de prioridade das tarefas.

Comentários:

(a) Errado. Story Points são utilizados para estimar o esforço ou complexidade de uma história de usuário, não o custo total do projeto.

(b) Errado. Story Points não são usados para determinar o número de membros necessários na equipe, mas para estimar a complexidade das histórias de usuário.

(c) Correto. Story Points ajudam a estimar o esforço necessário para completar uma história de usuário, considerando fatores como complexidade, incertezas e esforço requerido.

(d) Errado. A duração total de cada sprint é geralmente fixada em metodologias ágeis, e os Story Points ajudam a determinar quantas histórias podem ser completadas em um sprint, não sua duração total.

(e) Errado. Embora os Story Points possam influenciar a priorização ao revelar a complexidade das tarefas, seu propósito principal é ajudar na estimativa do esforço necessário, não na ordem de prioridade diretamente.

Gabarito: Letra C

---

23. (PROF. DIEGO / INÉDITA – 2024) Qual característica é específica do Método Delphi de estimativa?

a) Discussões em grupo abertas.
b) Estimativas feitas com base no menor tempo possível.

c) Foco exclusivo em estimativas quantitativas.
d) Uso de uma única rodada de questionários para decisão.
e) Anonimato nas rodadas de questionários.

Comentários:

(a) Errado. O Método Delphi é caracterizado pela ausência de discussões em grupo abertas; ao invés disso, utiliza um processo anônimo;

(b) Errado. As estimativas feitas no Método Delphi não se baseiam exclusivamente no menor tempo possível, mas na opinião consensual dos especialistas;

(c) Errado. Embora o Método Delphi possa focar em estimativas quantitativas, ele não se limita exclusivamente a elas; também pode ser usado para qualitativas;

(d) Errado. O Método Delphi normalmente envolve múltiplas rodadas de questionários para refinar as estimativas até alcançar um consenso;

(e) Correto. Uma característica específica é o anonimato nas rodadas de questionários, o que ajuda a evitar o viés de conformidade e a influência direta entre os participantes.

Gabarito: Letra E

---

24. (PROF. DIEGO / INÉDITA – 2024) Quais dos seguintes fatores NÃO são considerados ao usar T-Shirt Sizing para estimativas de escopo?

a) Complexidade técnica.
b) Tempo exato de entrega.
c) Volume de trabalho.
d) Possíveis desafios.
e) Riscos potenciais.

Comentários:

(a) Errado. A complexidade técnica é considerada em T-Shirt Sizing, pois influencia a estimativa do tamanho do esforço necessário para realizar uma tarefa;

(b) Correto. O tempo exato de entrega não é considerado em T-Shirt Sizing. Esta técnica é mais focada na comparação relativa do tamanho do esforço ou complexidade entre tarefas, e não determina um cronograma específico;

(c) Errado. O volume de trabalho é considerado, pois afeta a estimativa de quanto esforço é necessário para completar uma tarefa ou projeto;

(d) Errado. Possíveis desafios são levados em conta em T-Shirt Sizing, uma vez que podem afetar a complexidade ou o esforço necessário para completar uma tarefa;

(e) Errado. Riscos potenciais também são considerados, pois podem influenciar a avaliação da complexidade ou do esforço necessário para a realização das tarefas.

Gabarito: Letra B

---

25. (PROF. DIEGO / INÉDITA – 2024) Utilizar o Planning Poker em projetos que seguem metodologias ágeis pode reduzir o risco de subestimação ou superestimação das tarefas.

Comentários:

Utilizar o Planning Poker em projetos que seguem metodologias ágeis pode, de fato, reduzir o risco de subestimação ou superestimação das tarefas. Essa técnica de estimativa colaborativa envolve toda a equipe no processo de avaliação do esforço necessário para completar as tarefas, promovendo discussões que ajudam a esclarecer dúvidas e considerar diferentes perspectivas.

Ao incentivar a participação de todos os membros da equipe e exigir que justifiquem suas estimativas, o Planning Poker ajuda a alcançar um entendimento mais profundo e equilibrado das tarefas, levando a estimativas mais precisas e reduzindo a probabilidade de erros significativos de estimativa.

Gabarito: Correto

---

26. (PROF. DIEGO / INÉDITA – 2024) No Método Delphi, a identificação dos participantes nas rodadas de questionários é essencial para o processo de convergência de opiniões.

Comentários:

No Método Delphi, a identificação dos participantes nas rodadas de questionários não é essencial; na verdade, o anonimato é um componente chave do processo. O anonimato ajuda a evitar que as opiniões sejam influenciadas por fatores externos, como a autoridade percebida ou a popularidade de certos participantes, permitindo que as respostas sejam mais imparciais e focadas no conteúdo. Isso facilita uma convergência de opiniões baseada na reflexão cuidadosa e na reconsideração das próprias estimativas à luz das respostas agregadas, sem o viés de saber quem forneceu cada resposta.

Gabarito: Errado

---

27. (PROF. DIEGO / INÉDITA – 2024) T-Shirt Sizing é mais eficaz quando aplicado a projetos pequenos com tarefas bem definidas e pouco complexas.

Comentários:

Enquanto o T-Shirt Sizing pode ser eficaz em projetos pequenos com tarefas bem definidas e pouco complexas, a verdadeira força dessa técnica reside na sua aplicabilidade a uma ampla gama de projetos, incluindo aqueles de maior escala e com tarefas de complexidade variada. O T-Shirt Sizing ajuda a facilitar discussões iniciais sobre o tamanho relativo e a complexidade das tarefas, independentemente do tamanho do projeto.

Ele é particularmente útil em fases iniciais de planejamento, onde o objetivo é obter uma visão geral rápida do escopo e priorizar tarefas sem se aprofundar em estimativas numéricas detalhadas. Portanto, a técnica é valiosa tanto para projetos pequenos quanto grandes, ajudando equipes a estabelecer uma compreensão comum do esforço necessário de maneira rápida e intuitiva.

Gabarito: Errado

---

28. (PROF. DIEGO / INÉDITA – 2024) A técnica de Story Points é incompatível com a abordagem do Scrum, pois Scrum requer estimativas de tempo precisas para cada tarefa.

Comentários:

A técnica de Story Points é, na verdade, muito compatível e frequentemente usada em conjunto com a abordagem do Scrum. O Scrum não requer estimativas de tempo precisas para cada tarefa; em vez disso, valoriza estimativas relativas que ajudam a equipe a compreender o esforço necessário para completar as tarefas dentro do contexto do projeto. Story Points refletem a complexidade, o esforço e os riscos associados a uma tarefa, permitindo que a equipe faça planejamentos mais flexíveis e gerencie melhor o fluxo de trabalho durante os sprints.

Esta abordagem apoia a natureza iterativa e incremental do Scrum, focando mais na priorização e no progresso relativo do que em estimativas de tempo absolutas.

Gabarito: Errado

---

29. (PROF. DIEGO / INÉDITA – 2024) A utilização do Planning Poker pode aumentar o engajamento da equipe ao tornar o processo de estimativa mais interativo e inclusivo.

Comentários:

A utilização do Planning Poker de fato pode aumentar o engajamento da equipe ao tornar o processo de estimativa mais interativo e inclusivo. Essa técnica colaborativa envolve todos os membros da equipe no processo de estimativa, permitindo que cada um contribua com sua perspectiva única. Ao discutir as estimativas e justificar as escolhas, os membros da equipe participam ativamente, o que pode melhorar a comunicação, aumentar o entendimento mútuo das

tarefas e fortalecer o senso de propriedade e compromisso com o projeto. O aspecto lúdico do Planning Poker também adiciona um elemento de diversão ao processo de estimativa, tornando-o mais envolvente e menos tedioso.

Gabarito: Correto

---

30. (PROF. DIEGO / INÉDITA – 2024) No Método Delphi, a convergência para um consenso geralmente acontece rapidamente, frequentemente após a primeira rodada de questionários.

Comentários:

No Método Delphi, a convergência para um consenso não acontece geralmente de forma rápida, e raramente se alcança um consenso após apenas a primeira rodada de questionários. O método é projetado para facilitar um processo iterativo de questionários anônimos, com várias rodadas permitindo que os participantes refinem suas respostas com base nas informações agregadas das rodadas anteriores.

Esse processo de iteração ajuda a explorar e resolver diferenças de opinião, promovendo uma compreensão mais profunda das questões em discussão e, eventualmente, levando a um consenso ou a uma convergência de opiniões. Dependendo da complexidade do tema e da diversidade de perspectivas entre os especialistas, o processo pode exigir várias rodadas até que se chegue a um entendimento comum.

Gabarito: Errado

---

31. (PROF. DIEGO / INÉDITA – 2024) Estimativas baseadas em T-Shirt Sizing podem ser convertidas em Story Points para facilitar o planejamento detalhado e a alocação de recursos.

Comentários:

Estimativas baseadas em T-Shirt Sizing podem, de fato, ser convertidas em Story Points para facilitar o planejamento detalhado e a alocação de recursos. O T-Shirt Sizing é uma técnica de estimativa rápida e intuitiva que classifica as tarefas em categorias relativas, como tamanhos de camisetas (XS, S, M, L, XL).

Embora seja útil para obter uma visão geral rápida da complexidade das tarefas, para fins de planejamento mais detalhado, como a organização de sprints em metodologias ágeis, pode ser útil converter essas categorias relativas em Story Points. Essa conversão permite uma quantificação mais precisa do esforço necessário, ajudando na priorização das tarefas, no cálculo da velocidade da equipe e na previsão de capacidade para sprints futuros.

Gabarito: Correto

---

32. (FUNDATEC / PROCERGS – 2023) A garantia de segurança e confiabilidade de um blockchain é feita por meio de uma terceira parte mediadora, cuja confiabilidade é publicamente aceitNo Scrum, é muito comum estimar o peso das demandas através de pontos, também conhecido no inglês como Story Points, utilizando Fibonacci. Nesse contexto, quais das seguintes possibilidades de pontos NÃO é válida?

a) 3, 5, 7.
b) 1, 8, 13.
c) 2, 3, 5.
d) 1, 2, 3.
e) 1, 5, 8.

Comentários:

(a) Correto. A sequência 3, 5, 7 não segue a sequência de Fibonacci, que é um padrão utilizado em estimativas de Story Points no Scrum. A sequência de Fibonacci é caracterizada pela soma dos dois números anteriores para obter o próximo número, e 7 não é resultado da soma de 3 e 5;

(b) Errado. 1, 8, 13 seguem a lógica da sequência de Fibonacci, onde cada número é a soma dos dois números anteriores (1+2=3, 3+5=8, 5+8=13);

(c) Errado. 2, 3, 5 são números consecutivos na sequência de Fibonacci, onde cada número é a soma dos dois números anteriores;

(d) Errado. 1, 2, 3 também podem ser considerados parte da sequência de Fibonacci inicial, especialmente no contexto de estimativas ágeis, onde os números iniciais pequenos são comuns;

(e) Errado. 1, 5, 8 são válidos dentro do contexto de usar a sequência de Fibonacci para estimar Story Points, mesmo que não sejam consecutivos na sequência, pois o uso de Fibonacci em estimativas ágeis não exige que todos os números consecutivos sejam utilizados.

Gabarito: Letra A

---

33. (CESPE / CRO-SC – 2023) A engenharia de requisitos é uma etapa crítica no desenvolvimento de software, pois ajuda a garantir que o produto final atenda às necessidades do cliente e aos objetivos do projeto. Acerca da engenharia de requisitos, julgue o item subsequente.

A análise de ponto de função tem como base as funcionalidades que um sistema deve realizar, enquanto a story points é baseada em uma estimativa relativa, que compara a complexidade e o esforço de uma tarefa com outras já realizadas.

Comentários:

A análise de ponto de função é uma técnica para estimar o tamanho de um desenvolvimento de software baseando-se em suas funcionalidades. Story points, por outro lado, são usados em métodos ágeis para estimar o esforço de desenvolvimento de uma tarefa em relação a outras, levando em consideração a complexidade, o esforço necessário, e até mesmo os riscos envolvidos. Essas abordagens diferem essencialmente em sua base de cálculo, uma sendo mais objetiva (ponto de função) e a outra subjetiva (story points).

Gabarito: Correto

---

34. **(FEPESE / Companhia Águas de Joinville – 2023)** Dentro do Scrum, a técnica utilizada para medir esforço em uma tarefa, que geralmente utiliza a sequência de fibonacci, é:

a) ponto de função.
b) ponto de caso de uso.
c) kanban.
d) Planning Poker.
e) scrum-buts.

Comentários:

(a) Errado. Pontos de função são uma técnica de medição que avalia a funcionalidade que o software proporciona ao usuário, mas não está diretamente relacionada ao Scrum ou ao uso da sequência de Fibonacci para estimativas de esforço;

(b) Errado. Ponto de caso de uso é outra técnica de medição focada na análise de requisitos e não está associada ao Scrum ou ao uso da sequência de Fibonacci para estimativas;

(c) Errado. Kanban é uma metodologia ágil focada na gestão visual do trabalho, mas não é uma técnica específica para medir esforço em tarefas com base na sequência de Fibonacci;

(d) Correto. Planning Poker é uma técnica de estimativa utilizada em Scrum que envolve a equipe no processo de estimativa do esforço necessário para realizar tarefas ou histórias de usuário, frequentemente usando a sequência de Fibonacci para representar o esforço estimado;

(e) Errado. Scrum-buts refere-se a uma situação onde as equipes dizem que estão fazendo Scrum, mas introduzem exceções que contradizem os princípios fundamentais do Scrum. Não é uma técnica de medição de esforço.

Gabarito: Letra D

---

35. **(CONSULPLAN / MPE-PA – 2022)** No Scrum, algumas unidades são usadas para estimar o tempo para a realização de itens do Product Backlog de um projeto. A unidade relativa de tempo

criada pelo Time de Desenvolvimento, a qual é a unidade mais utilizada por equipes ágeis é conhecida como:

a) Tempo real.
b) Story Points.
c) Tempo ideal.
d) Planning Poker.

Comentários:

(a) Errado. "Tempo real" refere-se a uma estimativa baseada no tempo real esperado para completar uma tarefa, o que não é a unidade relativa de tempo mais usada em métodos ágeis como Scrum;

(b) Correto. "Story Points" são uma unidade relativa de estimativa usada em metodologias ágeis para representar a complexidade de um item do backlog, sem associá-lo a um tempo específico. Esta abordagem ajuda a considerar vários fatores como esforço, complexidade e incerteza;

(c) Errado. "Tempo ideal" é uma estimativa de quanto tempo algo levaria para ser concluído em condições perfeitas, o que é diferente da abordagem relativa utilizada pelos "Story Points";

(d) Errado. "Planning Poker" é uma técnica de estimativa usada por equipes ágeis para estimar a quantidade de trabalho usando "Story Points" ou outras unidades. Não é uma unidade de medida em si, mas um jogo de estimativa.

Gabarito: Letra B

---

36. (FGV / SEFAZ-AM – 2022) A técnica Planning Poker é tipicamente utilizada na metodologia Scrum para:

a) inspecionar o incremento e adaptar o backlog do produto.
b) sincronizar o trabalho previsto para as próximas 24 horas.
c) estimar o esforço necessário para terminar atividades.
d) documentar o trabalho do desenvolvedor.
e) identificar requisitos para tornar o produto mais útil.

Comentários:

(a) Errado. Inspecionar o incremento e adaptar o backlog do produto são atividades realizadas durante as revisões de sprint e o refinamento do backlog, e não através do Planning Poker;

(b) Errado. A sincronização do trabalho previsto para as próximas 24 horas é o objetivo da reunião diária de Scrum (Daily Scrum), não do Planning Poker;

(c) Correto. O Planning Poker é uma técnica utilizada para estimar o esforço necessário para completar as atividades ou histórias de usuário. Ela envolve a equipe inteira em uma estimativa por consenso, usando cartas que representam a quantidade de esforço que acreditam ser necessária para cada tarefa;

(d) Errado. Documentar o trabalho do desenvolvedor não é o foco do Planning Poker. O objetivo principal é estimar o esforço necessário para as atividades de desenvolvimento;

(e) Errado. Identificar requisitos para tornar o produto mais útil é uma parte do processo de planejamento e refinamento do backlog de produto, mas não é o propósito do Planning Poker. O foco do Planning Poker está na estimativa de esforço das histórias de usuário já identificadas.

Gabarito: Letra C

---

37. (IADES / BRB – 2021) No contexto das metodologias ágeis, o Planning Poker é uma

a) técnica para realizar estimativas de esforço de tarefas por meio do consenso entre o time.
b) cerimônia para discutir e estipular as metas de entrega da próxima sprint.
c) atividade lúdica realizada no decorrer das pausas das cerimônias para integrar o time.
d) técnica para escrever casos de uso em cartões, estruturando seus requisitos do ponto de vista do usuário.
e) ferramenta para registrar as lições aprendidas ao longo da sprint.

Comentários:

(a) Correto. O Planning Poker é uma técnica de estimativa no desenvolvimento ágil de software que utiliza consenso para estimar, por exemplo, a complexidade de tarefas ou o esforço necessário. Esta técnica envolve discussões que levam a um entendimento comum sobre a carga de trabalho, promovendo a colaboração da equipe;

(b) Errado. As metas de entrega da próxima sprint são discutidas durante a cerimônia de planejamento da sprint, não através do Planning Poker;

(c) Errado. Embora possa ser considerada uma atividade lúdica devido ao uso de cartas, o propósito principal do Planning Poker não é a integração do time, mas a estimativa de esforço das tarefas;

(d) Errado. Escrever casos de uso em cartões é uma técnica relacionada ao levantamento de requisitos e não está associada ao Planning Poker, que é uma técnica de estimativa;

(e) Errado. O registro de lições aprendidas é uma prática comum em metodologias ágeis, geralmente realizada durante a retrospectiva da sprint, e não está relacionado ao Planning Poker.

Gabarito: Letra A

---

38. (COMPERVE / UFRN – 2018) Uma ferramenta que pode ser usada na gestão de projetos é a Planning Poker. Sobre essa ferramenta, analise as afirmativas abaixo.

I É uma técnica que privilegia a opinião do "jogador" ganhador em detrimento da opinião dos demais.
II O "jogo" é composto por cartas com números que representam esforço estimado.
III O "jogo" possui 356 cartas.
IV Há uma forte interação entre os "jogadores" e product owners, que discutem questões do projeto antes de realizarem suas jogadas.

Estão corretas as afirmativas

a) II e IV.
b) I e III.
c) I e II.
d) III e IV.

Comentários:

I Errado. O Planning Poker não privilegia a opinião do jogador com a estimativa mais alta ou mais baixa; em vez disso, procura o consenso ou uma média das estimativas para chegar a um acordo sobre o esforço necessário;

II Correto. As cartas usadas no Planning Poker têm números que representam o esforço estimado necessário para realizar uma tarefa ou história de usuário. Esses números geralmente seguem uma sequência de Fibonacci ou uma distribuição semelhante para representar a complexidade de forma não linear;

III Errado. O jogo não possui 356 cartas. Planning Poker geralmente utiliza um conjunto limitado de cartas, com séries como 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, por exemplo, não alcançando um número tão alto quanto 356;

IV Correto. Durante as sessões de Planning Poker, há interação entre os desenvolvedores, o scrum master, e o product owner. O product owner explica os itens do backlog e pode esclarecer dúvidas, permitindo que os desenvolvedores façam estimativas informadas;

Gabarito: Letra A

---

39. (CESPE / SE-DF – 2017) Pontos de estórias (story points) são o meio mais adequado de se determinar o tempo de desenvolvimento de uma tarefa de uma sprint, pois, nesse caso, os desenvolvedores atribuem pontos de dificuldade para o desenvolvimento de uma tarefa específica e a pontuação de menor valor é a que determina o tempo de tarefa da sprint.

Comentários:

A questão mistura conceitos sobre a utilização de pontos de estória em metodologias ágeis. Pontos de estória são usados para estimar o esforço necessário para realizar uma tarefa, mas não determinam diretamente o tempo. Além disso, não é a pontuação de menor valor que determina o tempo de uma tarefa da sprint; a pontuação serve mais como um guia para a equipe entender a complexidade relativa das tarefas.

Gabarito: Errado

---

40. (FCC / TRT – 4ª Região – 2011) Para utilizar o processo de estimativa por Story Points em Scrum, inicialmente

a) o Product Owner deve atribuir valores de negócio para cada um dos itens do Product Backlog.
b) o Product Backlog deve considerar todos os fatores de Sprint contidos no Backlog Owner.
c) os Stakeholders devem atribuir os riscos do Product Owner para cada Sprint Planning.
d) os Stakeholders devem atribuir valores de negócio do Product Owner para cada Sprint.
e) o Product Planning deve avaliar cada Sprint contida no Backlog transacional e decidir pela prioridade de atividades.

Comentários:

(a) Correto. O Product Owner realmente é responsável por definir os valores de negócio dos itens do Product Backlog;

(b) Errado. Não existe um termo conhecido como "Backlog Owner" em Scrum, e a descrição não se alinha ao processo de estimativa por Story Points, que foca na complexidade e esforço dos itens do Product Backlog, independentemente dos "fatores de Sprint".

(c) Errado. A atribuição de riscos pelos stakeholders ao Product Owner não é uma etapa inicial para o uso de Story Points em Scrum. A estimativa por Story Points é realizada pela equipe de desenvolvimento para avaliar a complexidade das tarefas.

(d) Errado. A atribuição de valores de negócio pelos stakeholders para cada Sprint não é uma etapa no processo de estimativa por Story Points, que é uma técnica de estimativa de complexidade e esforço, não de valor de negócio.

(e) Errado. Não existe um processo denominado "Product Planning" em Scrum que avalia cada Sprint no "Backlog transacional". As prioridades são definidas pelo Product Owner no Product Backlog, e as estimativas de Story Points são geralmente feitas pela equipe de desenvolvimento, focando na complexidade e no esforço necessário para realizar as tarefas.

Gabarito: Letra A

---

41. (CESGRANRIO / Transpetro – 2018) No uso de alguns métodos ágeis, como o SCRUM, é comum que o esforço de desenvolvimento seja avaliado por meio de Pontos de História (Story Points). Essa metodologia usa cartas, semelhantes a cartas de baralho, onde cada uma apresenta um valor de uma escala de valores numéricos, que, normalmente, segue a seguinte sequência:

a) 0,1,2,3,4,5,6,7,8,9,10
b) 0,1,2,3,5,8,13,20,40 e 100
c) 0,1,2,4,8,16,32,64,128
d) 1,2,3,6,12,24,48,96
e) 1,5,10,50,100,500,1000

Comentários:

(a) Errado. Esta sequência é linear e não reflete a natureza exponencial da incerteza em estimativas de projetos ágeis;

(b) Correto. Esta sequência, conhecida como sequência de Fibonacci modificada, é comumente usada no planejamento ágil para estimar esforço, refletindo a incerteza e a complexidade crescentes à medida que as histórias de usuário se tornam maiores;

(c) Errado. Apesar de exponencial, esta sequência não é comumente usada no SCRUM para estimar Pontos de História;

(d) Errado. Esta sequência não segue o padrão típico usado para estimar Pontos de História em métodos ágeis;

(e) Errado. Esta sequência reflete grandes saltos entre valores, não sendo prática para a estimativa de esforço em projetos ágeis, onde se prefere uma progressão mais suave para refletir incerteza.

Gabarito: Letra B

Lista de Questões

1. **(CESPE / INPI – 2024)** Na gestão do backlog de um produto usando-se uma metodologia ágil, o controle do versionamento dos artefatos é adequado para procurar manter o backlog com a característica de emergente.

2. **(PROF. DIEGO / INÉDITA – 2024)** O escopo de um projeto inclui apenas os entregáveis finais, não abrangendo o trabalho necessário para criá-los.

3. **(PROF. DIEGO / INÉDITA – 2024)** Uma definição clara de escopo fornece direção e facilita a compreensão comum dos objetivos do projeto entre todas as partes interessadas.

4. **(PROF. DIEGO / INÉDITA – 2024)** A estimativa de escopo é uma etapa dispensável no processo de gerenciamento de projetos, uma vez que os detalhes podem ser ajustados conforme o desenvolvimento do projeto.

5. **(PROF. DIEGO / INÉDITA – 2024)** A criação da Estrutura Analítica do Projeto (EAP) é um processo que visa decompor o trabalho do projeto em componentes menores e mais gerenciáveis.

6. **(PROF. DIEGO / INÉDITA – 2024)** O controle de mudanças não é necessário uma vez que o escopo do projeto tenha sido definido e validado pelas partes interessadas.

7. **(PROF. DIEGO / INÉDITA – 2024)** A validação do escopo é uma fase em que se verifica se os entregáveis do projeto atendem às necessidades e expectativas das partes interessadas.

8. **(PROF. DIEGO / INÉDITA – 2024)** A estimativa de escopo não contribui para a identificação e alocação de recursos necessários para a execução do projeto.

9. **(PROF. DIEGO / INÉDITA – 2024)** A estimativa de escopo ajuda a prevenir a expansão não planejada do escopo, conhecida como "Scope Creep", que pode levar a atrasos e aumento de custos.

10. **(PROF. DIEGO / INÉDITA – 2024)** O que é "Scope Creep"?

a) Uma técnica de gerenciamento de riscos.
b) Uma metodologia de definição de escopo.
c) Um processo formal de controle de mudanças.
d) A expansão não controlada do escopo do projeto.
e) Uma ferramenta para comunicação eficaz.

11. **(PROF. DIEGO / INÉDITA – 2024)** O que a criação da Estrutura Analítica do Projeto (EAP) facilita?

a) A comunicação entre as equipes.
b) O controle de mudanças.
c) A identificação de redundâncias no trabalho.
d) A decomposição do trabalho do projeto.
e) A validação do escopo com as partes interessadas.

12. **(PROF. DIEGO / INÉDITA – 2024)** Qual dos seguintes NÃO é um objetivo da estimativa de escopo?

a) Prevenção de "Scope creep".
b) Facilitação da comunicação entre as partes interessadas.
c) Redução do tempo necessário para completar o projeto.
d) Identificação e alocação de recursos necessários.
e) Alinhamento de expectativas entre a equipe e os patrocinadores.

13. **(PROF. DIEGO / INÉDITA – 2024)** Planning Poker é uma técnica de estimativa que se baseia unicamente na intuição individual de cada membro da equipe, sem necessidade de discussões em grupo.

14. **(PROF. DIEGO / INÉDITA – 2024)** Story Points consideram a complexidade, o volume de trabalho, os riscos potenciais e incertezas, ao contrário das estimativas baseadas apenas em tempo.

15. **(PROF. DIEGO / INÉDITA – 2024)** O Método Delphi é caracterizado pela falta de anonimato nas respostas, permitindo que os participantes se influenciem mutuamente.

16. **(PROF. DIEGO / INÉDITA – 2024)** T-Shirt Sizing é uma técnica de estimativa que atribui valores numéricos exatos às tarefas.

17. **(PROF. DIEGO / INÉDITA – 2024)** No Planning Poker, se as estimativas iniciais variarem significativamente, o consenso deve ser atingido sem discussão adicional.

18. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de T-Shirt Sizing pode ser utilizada em qualquer fase do projeto para ajudar a priorizar tarefas.

19. **(PROF. DIEGO / INÉDITA – 2024)** A estimativa de Story Points se torna menos precisa à medida que a equipe ganha mais experiência no projeto.

20. **(PROF. DIEGO / INÉDITA – 2024)** No Método Delphi, a convergência para um consenso ocorre após uma única rodada de questionários.

21. **(PROF. DIEGO / INÉDITA – 2024)** Qual das seguintes opções melhor descreve a finalidade do Planning Poker?

a) Determinar a duração exata do projeto.
b) Estimular a competição entre os membros da equipe.
c) Facilitar estimativas de esforço consensuais e baseadas em discussão.
d) Alocar tarefas específicas a membros da equipe.
e) Priorizar as histórias de usuário com base em sua importância.

22. **(PROF. DIEGO / INÉDITA – 2024)** No contexto de metodologias ágeis, o que Story Points ajudam a estimar?

a) O custo total do projeto.
b) O número de membros necessários na equipe.
c) O esforço necessário para completar uma história de usuário.
d) A duração total de cada sprint.
e) A ordem de prioridade das tarefas.

23. **(PROF. DIEGO / INÉDITA – 2024)** Qual característica é específica do Método Delphi de estimativa?

a) Discussões em grupo abertas.
b) Estimativas feitas com base no menor tempo possível.
c) Foco exclusivo em estimativas quantitativas.
d) Uso de uma única rodada de questionários para decisão.
e) Anonimato nas rodadas de questionários.

24. **(PROF. DIEGO / INÉDITA – 2024)** Quais dos seguintes fatores NÃO são considerados ao usar T-Shirt Sizing para estimativas de escopo?

a) Complexidade técnica.
b) Tempo exato de entrega.
c) Volume de trabalho.
d) Possíveis desafios.
e) Riscos potenciais.

25. **(PROF. DIEGO / INÉDITA – 2024)** Utilizar o Planning Poker em projetos que seguem metodologias ágeis pode reduzir o risco de subestimação ou superestimação das tarefas.

26. **(PROF. DIEGO / INÉDITA – 2024)** No Método Delphi, a identificação dos participantes nas rodadas de questionários é essencial para o processo de convergência de opiniões.

27. **(PROF. DIEGO / INÉDITA – 2024)** T-Shirt Sizing é mais eficaz quando aplicado a projetos pequenos com tarefas bem definidas e pouco complexas.

28. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de Story Points é incompatível com a abordagem do Scrum, pois Scrum requer estimativas de tempo precisas para cada tarefa.

29. **(PROF. DIEGO / INÉDITA – 2024)** A utilização do Planning Poker pode aumentar o engajamento da equipe ao tornar o processo de estimativa mais interativo e inclusivo.

30. **(PROF. DIEGO / INÉDITA – 2024)** No Método Delphi, a convergência para um consenso geralmente acontece rapidamente, frequentemente após a primeira rodada de questionários.

31. **(PROF. DIEGO / INÉDITA – 2024)** Estimativas baseadas em T-Shirt Sizing podem ser convertidas em Story Points para facilitar o planejamento detalhado e a alocação de recursos.

32. **(FUNDATEC / PROCERGS – 2023)** A garantia de segurança e confiabilidade de um blockchain é feita por meio de uma terceira parte mediadora, cuja confiabilidade é publicamente aceitNo Scrum, é muito comum estimar o peso das demandas através de pontos, também conhecido no inglês como Story Points, utilizando Fibonacci. Nesse contexto, quais das seguintes possibilidades de pontos NÃO é válida?

a) 3, 5, 7.
b) 1, 8, 13.
c) 2, 3, 5.
d) 1, 2, 3.
e) 1, 5, 8.

33. **(CESPE / CRO-SC – 2023)** A engenharia de requisitos é uma etapa crítica no desenvolvimento de software, pois ajuda a garantir que o produto final atenda às necessidades do cliente e aos objetivos do projeto. Acerca da engenharia de requisitos, julgue o item subsequente.

A análise de ponto de função tem como base as funcionalidades que um sistema deve realizar, enquanto a story points é baseada em uma estimativa relativa, que compara a complexidade e o esforço de uma tarefa com outras já realizadas.

34. **(FEPESE / Companhia Águas de Joinville – 2023)** Dentro do Scrum, a técnica utilizada para medir esforço em uma tarefa, que geralmente utiliza a sequência de fibonacci, é:

a) ponto de função.
b) ponto de caso de uso.
c) kanban.
d) Planning Poker.
e) scrum-buts.

35. **(CONSULPLAN / MPE-PA – 2022)** No Scrum, algumas unidades são usadas para estimar o tempo para a realização de itens do Product Backlog de um projeto. A unidade relativa de tempo

criada pelo Time de Desenvolvimento, a qual é a unidade mais utilizada por equipes ágeis é conhecida como:

a) Tempo real.
b) Story Points.
c) Tempo ideal.
d) Planning Poker.

36. **(FGV / SEFAZ-AM – 2022)** A técnica Planning Poker é tipicamente utilizada na metodologia Scrum para:

a) inspecionar o incremento e adaptar o backlog do produto.
b) sincronizar o trabalho previsto para as próximas 24 horas.
c) estimar o esforço necessário para terminar atividades.
d) documentar o trabalho do desenvolvedor.
e) identificar requisitos para tornar o produto mais útil.

37. **(IADES / BRB – 2021)** No contexto das metodologias ágeis, o Planning Poker é uma

a) técnica para realizar estimativas de esforço de tarefas por meio do consenso entre o time.
b) cerimônia para discutir e estipular as metas de entrega da próxima sprint.
c) atividade lúdica realizada no decorrer das pausas das cerimônias para integrar o time.
d) técnica para escrever casos de uso em cartões, estruturando seus requisitos do ponto de vista do usuário.
e) ferramenta para registrar as lições aprendidas ao longo da sprint.

38. **(COMPERVE / UFRN – 2018)** Uma ferramenta que pode ser usada na gestão de projetos é a Planning Poker. Sobre essa ferramenta, analise as afirmativas abaixo.

I É uma técnica que privilegia a opinião do "jogador" ganhador em detrimento da opinião dos demais.
II O "jogo" é composto por cartas com números que representam esforço estimado.
III O "jogo" possui 356 cartas.
IV Há uma forte interação entre os "jogadores" e product owners, que discutem questões do projeto antes de realizarem suas jogadas.

Estão corretas as afirmativas

a) II e IV.
b) I e III.
c) I e II.
d) III e IV.

39. **(CESPE / SE-DF – 2017)** Pontos de estórias (story points) são o meio mais adequado de se determinar o tempo de desenvolvimento de uma tarefa de uma sprint, pois, nesse caso, os desenvolvedores atribuem pontos de dificuldade para o desenvolvimento de uma tarefa específica e a pontuação de menor valor é a que determina o tempo de tarefa da sprint.

40. **(FCC / TRT – 4ª Região – 2011)** Para utilizar o processo de estimativa por Story Points em Scrum, inicialmente

a) o Product Owner deve atribuir valores de negócio para cada um dos itens do Product Backlog.
b) o Product Backlog deve considerar todos os fatores de Sprint contidos no Backlog Owner.
c) os Stakeholders devem atribuir os riscos do Product Owner para cada Sprint Planning.
d) os Stakeholders devem atribuir valores de negócio do Product Owner para cada Sprint.
e) o Product Planning deve avaliar cada Sprint contida no Backlog transacional e decidir pela prioridade de atividades.

41. **(CESGRANRIO / Transpetro – 2018)** No uso de alguns métodos ágeis, como o SCRUM, é comum que o esforço de desenvolvimento seja avaliado por meio de Pontos de História (Story Points). Essa metodologia usa cartas, semelhantes a cartas de baralho, onde cada uma apresenta um valor de uma escala de valores numéricos, que, normalmente, segue a seguinte sequência:

a) 0,1,2,3,4,5,6,7,8,9,10
b) 0,1,2,3,5,8,13,20,40 e 100
c) 0,1,2,4,8,16,32,64,128
d) 1,2,3,6,12,24,48,96
e) 1,5,10,50,100,500,1000

GABARITO

1. CORRETO
2. ERRADO
3. CORRETO
4. ERRADO
5. CORRETO
6. ERRADO
7. CORRETO
8. ERRADO
9. CORRETO
10. LETRA D
11. LETRA D
12. LETRA C
13. ERRADO
14. CORRETO
15. ERRADO
16. ERRADO
17. ERRADO
18. CORRETO
19. ERRADO
20. ERRADO
21. LETRA C
22. LETRA C
23. LETRA E
24. LETRA B
25. CORRETO
26. ERRADO
27. ERRADO
28. ERRADO
29. CORRETO
30. ERRADO
31. CORRETO
32. LETRA A
33. CORRETO
34. LETRA D
35. LETRA B
36. LETRA C
37. LETRA A
38. LETRA A
39. ERRADO
40. LETRA A
41. LETRA B

ESPECIFICAÇÃO POR EXEMPLO

Conceitos Básicos

O que é uma especificação? **Especificação é a definição detalhada das características de algo!** Uma especificação deve ser precisa e testável; uma verdadeira especificação e, não, um script; e deve ser sobre funcionalidade de negócios e, não, sobre design de software. Uma vez implementada a funcionalidade, a especificação que a descreve servirá a um propósito diferente. Ele documentará o que o sistema faz e nos alertará sobre a regressão funcional.

Já uma Especificação por Exemplo é um **conjunto de padrões de processos que ajudam equipes na construção do produto correto, focando na comunicação entre todas as partes envolvidas, fazendo com que todos tenham um entendimento claro sobre o que está sendo produzido e com isso possam colaborar da forma mais eficaz possível**. Galera, a Especificação por Exemplo não é a forma como a maioria de nós foi criada para pensar em especificações.

Nós aprendemos que, em regra, as especificações devem ser genéricas, para cobrir todos os casos. Quando utilizamos exemplos, eles destacam apenas alguns pontos e você mesmo deve inferir as generalizações. *O que eu quero dizer com isso?* **Isso significa que a Especificação por Exemplo não pode ser a única técnica de requisitos que você usa, mas não significa que ela não possa assumir um papel de liderança.**

Até agora, a ideia dominante com especificações rigorosas, ou seja, aquelas que podem ser claramente julgadas como aprovadas ou reprovadas, é usar pré-condições e pós-condições. Essas técnicas dominam os métodos formais e também sustentam o *Design por Contrato*[1]. **Elas têm seu lugar, mas não são ideais. As pré-condições e pós-condições podem ser muito fáceis de escrever em algumas situações, mas em outras podem ser muito complicadas.**

Já houve várias tentativas em aplicações corporativas e, em muitas situações, é tão difícil escrever as pré/pós-condições quanto escrever a solução em si. **Na Especificação por Exemplo, os exemplos geralmente são muito mais fáceis de criar, especialmente para os não especialistas em TI.** Uma das coisas mais perigosas sobre uma especificação de requisitos tradicional é quando as pessoas pensam que depois de escrevê-la, a comunicação será encerrada.

A Especificação por Exemplo só funciona em contextos de relação de trabalho em que ambos os lados estão colaborando. Os exemplos acionam abstrações na equipe de design enquanto mantêm as abstrações fundamentadas. A Especificação por Exemplo é uma ferramenta poderosa, mas não deve ser a única ferramenta. Vamos resumir rapidamente tudo que precisamos saber sobre a Especificação por Exemplo...

---

[1] A ideia principal por trás do **Design por Contrato** é que cada componente deve ter um contrato que descreva seu comportamento e como ele deve interagir com outros componentes do sistema. Este contrato é escrito em linguagem formal e pode incluir condições, invariantes, pré-condições, pós-condições e outros elementos.

A Especificação por Exemplo é um conjunto de padrões de processos que facilitam a mudança em produtos de software para garantir que o produto correto seja entregue com eficiência. **Trata-se de uma abordagem colaborativa para definir os requisitos e testes funcionais orientados ao negócio para produtos de software com base na captura e ilustração de requisitos usando exemplos realistas em vez de declarações abstratas.**

Busca-se focar no desenvolvimento e na entrega de requisitos de negócios prioritários e verificáveis. Embora o conceito de Especificação por Exemplo em si seja relativamente novo, é simplesmente uma reformulação das práticas existentes. Ela suporta um vocabulário muito específico/conciso conhecido como linguagem ubíqua que: habilita requisitos executáveis; é usado por todos na equipe; é criado por uma equipe multifuncional; e captura a compreensão de todos.

A Especificação por Exemplo pode ser usada como uma entrada direta na construção de testes automatizados que refletem o domínio de negócios. Assim, seu foco é construir o produto correto e construir o produto corretamente. Dentre suas vantagens, temos: maior qualidade; desperdício reduzido; risco reduzido de defeitos de produção; esforço focado; alterações podem ser feitas com mais segurança; e envolvimento comercial aprimorado. Pontos importantes...

- Construir o produto correto e construir corretamente o produto são duas coisas diferentes – você precisa fazer as duas coisas para ter sucesso.

- A Especificação por Exemplo fornece documentação suficiente no momento certo, ajudando a criar o produto certo com iterações curtas ou processos de desenvolvimento baseados em fluxo.

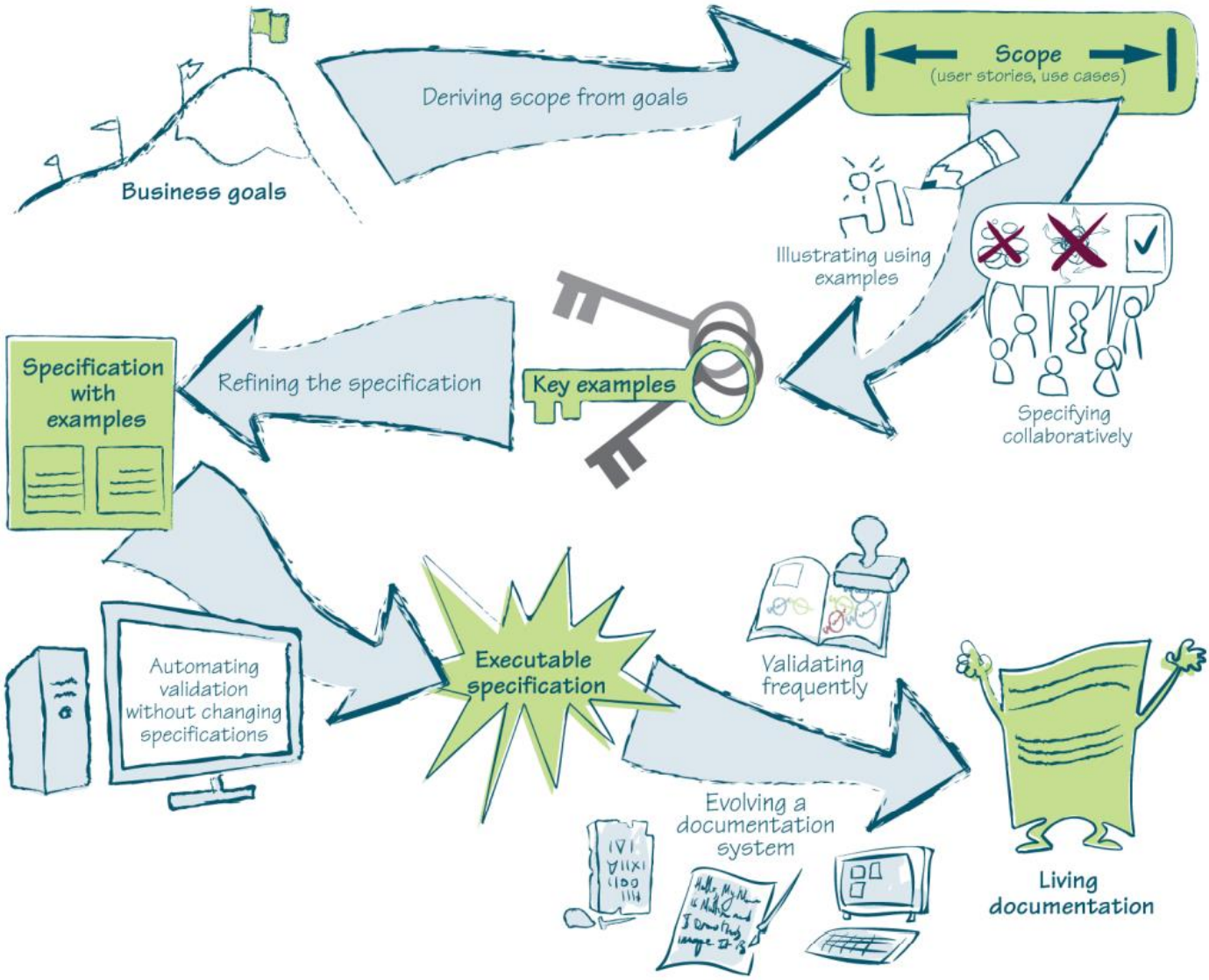
- A Especificação por Exemplo ajuda a melhorar a qualidade dos produtos de software, reduz significativamente o retrabalho e permite que as equipes alinhem melhor as atividades de análise, desenvolvimento e teste.

- A longo prazo, a Especificação por Exemplo ajuda as equipes a criar um sistema de documentação vivo, uma descrição relevante e confiável da funcionalidade que é atualizada automaticamente com o código da linguagem de programação.

- As práticas funcionam melhor com métodos de desenvolvimento iterativos curtos (Scrum, XP) ou baseados em fluxo (Kanban). Algumas ideias também são aplicáveis ao RUP ou Cascata.

Principais Padrões de Processo

Os principais padrões de processo de Especificação por Exemplo são:



Derivando o escopo das metas

O escopo de implementação oferece uma solução para um problema de negócio ou um meio para atingir uma meta de negócios. Muitas equipes esperam que um cliente, um proprietário do produto ou um usuário comercial decida sobre o escopo do trabalho antes do início da implementação (tudo o que ocorre antes da implementação é geralmente ignorado pela equipe de desenvolvimento de software).

Depois que os usuários de negócios especificam exatamente o que desejam, a equipe de entrega de software o implementa. Isso é supostamente o que deixará os clientes satisfeitos. Na verdade, é

aí que começam os problemas com a construção do produto certo. Ao contar com os clientes para fornecer uma lista de histórias de usuários, casos de uso ou outras informações relevantes, **as equipes de entrega de software estão pedindo a seus clientes que projetem uma solução.**

Mas os usuários de negócios não são designers de software. Se os clientes definem o escopo, o projeto não se beneficia do conhecimento das pessoas da equipe de entrega. Isso resulta em um software que faz o que o cliente pediu, mas não o que ele realmente queria. Em vez de aceitar cegamente os requisitos de software como uma solução para um problema desconhecido, as equipes bem-sucedidas obtêm o escopo dos objetivos.

Eles começam com a meta de negócios de um cliente. **Em seguida, eles colaboram para definir o escopo que atingirá esse objetivo.** A equipe trabalha com os usuários de negócios para determinar a solução. Os usuários de negócios se concentram em comunicar a intenção do recurso desejado e o valor que esperam obter dele. Isso ajuda todos a entender o que é necessário. A equipe pode então sugerir uma solução que seja mais barata, rápida e fácil de fornecer ou manter do que os usuários de negócios criariam por conta própria.

Especificando Colaborativamente

Se os desenvolvedores e testadores não estiverem envolvidos na criação de especificações, essas especificações devem ser comunicadas separadamente à equipe. Na prática, isso deixa muitas oportunidades para mal-entendidos; detalhes podem se perder na tradução. Como consequência, os usuários de negócios precisam validar o software após a entrega e as equipes precisam voltar e fazer alterações se ele falhar na validação. Isso tudo é retrabalho desnecessário.

Em vez de depender de uma única pessoa para obter as especificações corretas isoladamente, as equipes de entrega bem-sucedidas colaboram com os usuários de negócios para especificar a solução. Pessoas de origens diferentes têm ideias diferentes e usam suas próprias técnicas baseadas na experiência para resolver problemas. Especialistas técnicos sabem como fazer melhor uso da infraestrutura subjacente ou como as tecnologias emergentes podem ser aplicadas.

Os testadores sabem onde procurar possíveis problemas e a equipe deve trabalhar para evitar esses problemas. Todas essas informações precisam ser capturadas ao projetar especificações. Especificar de forma colaborativa nos permite aproveitar o conhecimento e a experiência de toda a equipe. Também cria uma propriedade coletiva das especificações, tornando todos mais engajados no processo de entrega.

Ilustrando Usando Exemplos

A linguagem natural é ambígua e depende do contexto. Requisitos escritos em tal linguagem sozinhos não podem fornecer um contexto completo e inequívoco para desenvolvimento ou teste. Os desenvolvedores e testadores precisam interpretar os requisitos para produzir software e scripts de teste, e pessoas diferentes podem interpretar conceitos complicados de maneira diferente. Isso

é especialmente problemático quando algo que parece óbvio, na verdade, requer experiência no domínio ou conhecimento do jargão para ser totalmente compreendido.

Pequenas diferenças de entendimento têm um efeito cumulativo, muitas vezes levando a problemas que exigem retrabalho após a entrega. Isso causa atrasos desnecessários. Em vez de esperar que as especificações sejam expressas precisamente pela primeira vez em uma linguagem de programação durante a implementação, as equipes bem-sucedidas ilustram as especificações usando exemplos.

A equipe trabalha com os usuários de negócios para identificar os principais exemplos que descrevem a funcionalidade esperada. Durante esse processo, desenvolvedores e testadores geralmente sugerem exemplos adicionais que ilustram casos extremos ou abordam áreas do sistema que são particularmente problemáticas. **Isso elimina lacunas e inconsistências funcionais e garante que todos os envolvidos tenham uma compreensão compartilhada do que precisa ser entregue, evitando retrabalho resultante de má interpretação e tradução.**

Se o sistema funcionar corretamente para todos os exemplos-chave, ele atenderá à especificação com a qual todos concordaram. Os principais exemplos definem efetivamente o que o software precisa fazer. Eles são o alvo do desenvolvimento e um critério de avaliação objetivo para verificar se o desenvolvimento foi feito. **Se os principais exemplos forem fáceis de entender e comunicar, eles podem ser efetivamente usados como requisitos inequívocos e detalhados.**

Refinando as Especificações

Uma discussão aberta durante a colaboração cria um entendimento compartilhado do domínio, mas os exemplos resultantes geralmente apresentam mais detalhes do que o necessário. Por exemplo, os usuários corporativos pensam sobre a perspectiva da interface do usuário, então eles oferecem exemplos de como algo deve funcionar ao clicar em links e preencher campos. **Essas descrições detalhadas restringem o sistema; detalhar como algo deve ser feito em vez do que é necessário é um desperdício.**

O excesso de detalhes torna os exemplos mais difíceis de comunicar e entender. Os principais exemplos devem ser concisos para serem úteis. Ao refinar a especificação, as equipes bem-sucedidas removem informações estranhas e criam um contexto concreto e preciso para desenvolvimento e teste. Eles definem o alvo com a quantidade certa de detalhes para implementá-lo e verificá-lo. Eles identificam o que o software deve fazer, não como ele faz.

Exemplos refinados podem ser usados como critérios de aceitação para entrega; o desenvolvimento não é feito até que o sistema funcione corretamente para todos os exemplos. Depois de fornecer informações adicionais para facilitar a compreensão dos principais exemplos, as equipes criam especificações com exemplos, que são uma especificação de trabalho, um teste de aceitação e um teste de regressão funcional futuro.

Automatizando Validação Sem Alterar Especificações

Depois que uma equipe concorda com as especificações com exemplos e as refina, a equipe pode usá-las como um alvo para implementação e um meio para validar o produto. O sistema será validado várias vezes com esses testes durante o desenvolvimento para garantir que atenda ao objetivo. Executar essas verificações manualmente introduziria atrasos desnecessários e o feedback seria lento.

O feedback rápido é uma parte essencial do desenvolvimento de software em iterações curtas ou em modo de fluxo, portanto, precisamos tornar o processo de validação do sistema barato e eficiente. Uma solução óbvia é a automação. Mas essa automação é conceitualmente diferente da automação usual do desenvolvedor ou do testador.

Se automatizarmos a validação dos principais exemplos usando ferramentas tradicionais de automação de programação (unidade) ou ferramentas tradicionais de automação de teste funcional, corremos o risco de apresentar problemas se os detalhes se perderem entre a especificação de negócios e a automação técnica. **As especificações tecnicamente automatizadas se tornarão inacessíveis aos usuários corporativos.**

Quando os requisitos mudarem (e é quando, não se) ou quando os desenvolvedores ou testadores precisarem de mais esclarecimentos, não poderemos usar a especificação que automatizamos anteriormente. Poderíamos manter os principais exemplos como testes e em uma forma mais legível, como documentos do Word ou páginas da web, mas assim que houver mais de uma versão da verdade, teremos problemas de sincronização. **É por isso que a documentação em papel nunca é ideal.**

Para obter o máximo dos exemplos-chave, as equipes bem-sucedidas automatizam a validação sem alterar as informações. **Eles literalmente mantêm tudo na especificação igual durante a automação - não há risco de tradução incorreta.** Como eles automatizam a validação sem alterar as especificações, os principais exemplos parecem quase iguais aos de um quadro branco: compreensíveis e acessíveis a todos os membros da equipe.

Uma especificação automatizada com exemplos compreensíveis e acessíveis a todos os membros da equipe torna-se uma especificação executável. Podemos usá-lo como alvo de desenvolvimento e verificar facilmente se o sistema cumpre o que foi acordado, e podemos usar esse mesmo documento para obter esclarecimentos dos usuários de negócios. Se precisarmos alterar a especificação, teremos que fazê-lo em apenas um lugar.

Validando Frequentemente

Para oferecer suporte eficiente a um sistema de software, precisamos saber o que ele faz e por quê. **Em muitos casos, a única maneira de fazer isso é detalhar o código de programação ou encontrar alguém que possa fazer isso por nós.** O código geralmente é a única coisa em que

realmente podemos confiar; a maioria da documentação escrita está desatualizada antes da entrega do projeto. Programadores são oráculos de conhecimento e gargalos de informação.

As especificações executáveis podem ser facilmente validadas no sistema. **Se essa validação for frequente, podemos ter tanta confiança nas especificações do executável quanto no código.** Ao verificar com frequência todas as especificações executáveis, as equipes com quem conversei descobrem rapidamente quaisquer diferenças entre o sistema e as especificações.

Como as especificações executáveis são fáceis de entender, as equipes podem discutir as mudanças com seus usuários de negócios e decidir como lidar com os problemas. Eles sincronizam constantemente seus sistemas e especificações executáveis.

Evoluindo um Sistema de Documentação

As equipes mais bem-sucedidas não ficam satisfeitas com um conjunto de especificações executáveis frequentemente validadas. **Eles garantem que as especificações sejam bem organizadas, fáceis de encontrar e acessar e consistentes.** À medida que seus projetos evoluem, a compreensão das equipes sobre o domínio muda. As oportunidades de mercado também causam mudanças nos modelos de domínio de negócios.

As equipes que tiram o máximo proveito da Especificação por Exemplo atualizam suas especificações para refletir essas mudanças, desenvolvendo um sistema de documentação vivo. A documentação viva é uma fonte confiável e autorizada de informações sobre a funcionalidade do sistema que qualquer pessoa pode acessar. **É tão confiável quanto o código, mas muito mais fácil de ler e entender.**

A equipe de suporte pode usá-lo para descobrir o que o sistema faz e por quê. Os desenvolvedores podem usá-lo como um alvo para o desenvolvimento. Os testadores podem usá-lo para testes. Os analistas de negócios podem usá-lo como ponto de partida ao analisar o impacto de uma alteração de funcionalidade solicitada. Ele também fornece testes de regressão gratuitos.

DÉBITO TÉCNICO

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Imagine que você seja um cara bastante procrastinador (muito comum entre concurseiros) e tenha uma tarefa para fazer, mas decide deixá-la para depois porque está ocupado com outras coisas. **Esse adiamento pode gerar uma dívida para você, que terá que lidar com a tarefa mais tarde e possivelmente em um momento menos conveniente.** De maneira semelhante, o débito técnico no desenvolvimento de software é como uma dívida do desenvolvedor.

O desenvolvedor a assume ao deixar de realizar tarefas importantes, como refatorar o código, realizar testes adequados ou atualizar bibliotecas. Essas omissões podem levar a problemas futuros, como bugs, mau desempenho ou problemas de segurança. Assim como uma dívida financeira, o débito técnico precisa ser pago mais tarde, muitas vezes com juros, na forma de correções de emergência, retrabalho ou atrasos no cronograma.

É importante que os desenvolvedores e equipes de software gerenciem e controlem o débito técnico para garantir que o software seja robusto, de alta qualidade e fácil de manter. Isso envolve identificar as áreas de dívida técnica, priorizar o pagamento de débitos e investir em práticas de desenvolvimento saudáveis para minimizar a criação de novas dívidas. No meu trabalho, nós mantemos um backlog dos principais débitos técnicos para ir "pagando" sempre que possível.

Enfim, o débito técnico é basicamente o custo a longo prazo de desenvolver um software com baixa qualidade. Quando você escreve um código de má qualidade ou deixa de seguir boas práticas de programação, você acumula um débito técnico. Mais cedo ou mais tarde, você precisará corrigir esses problemas para manter o software funcionando corretamente ou adicionar novas funcionalidades.

As consequências do débito técnico são muitas e variadas. **Quando você acumula débito técnico, o software se torna mais difícil de entender, de modificar e de manter.** Isso pode levar a problemas de desempenho, segurança e escalabilidade, além de dificultar a adição de novas funcionalidades no futuro. Logo, é importante destacar a relação intrínseca entre débito técnico no desenvolvimento de software e refatoração de software.

Além disso, acumular débito técnico pode ser muito caro a longo prazo. A correção de problemas de qualidade geralmente é demorada e cara, o que pode afetar negativamente o orçamento e o cronograma do projeto. Por isso, é importante gerenciar o débito técnico de forma eficaz. Identificar, priorizar e corrigir o débito técnico acumulado pode ajudar a melhorar a qualidade do software, a reduzir custos e a garantir que o software esteja preparado para as demandas futuras.

Identificar o débito técnico pode ser um pouco complicado, mas existem algumas formas de se fazer isso. Geralmente, o débito técnico pode ser identificado a partir de sintomas comuns, como

atrasos no projeto, problemas de qualidade, bugs frequentes, entre outros. **Além disso, uma das melhores maneiras de identificar o débito técnico é realizar revisões regulares do código, seja por meio de revisões de código (code review) ou testes de software.**

Durante essas revisões, é possível encontrar problemas comuns de qualidade de software, tais como código duplicado, falta de documentação, uso inadequado de variáveis e funções, entre outros. Quanto às categorias de débito técnico, existem várias maneiras de categorizar o débito técnico, mas uma das mais comuns é a divisão entre débito técnico "intencional" e "não intencional".

O débito técnico intencional ocorre quando você toma uma decisão consciente de priorizar a entrega rápida do software em detrimento da qualidade. Por outro lado, o débito técnico não intencional ocorre quando você não percebe que está acumulando débito técnico, geralmente por falta de conhecimento ou tempo limitado. **Também é importante medir o débito técnico para ter uma ideia clara da quantidade de trabalho necessário para corrigi-lo.**

Existem várias ferramentas que podem ajudar a medir o débito técnico, como o SonarQube e o CodeClimate, que analisam o código-fonte em busca de problemas comuns e geram relatórios sobre o débito técnico acumulado. **A métrica mais comum para medir o débito técnico é o tempo necessário para corrigir os problemas encontrados, geralmente medido em dias ou semanas.** Gerenciar o débito técnico também é uma parte essencial do desenvolvimento de software.

Para priorizar o tratamento do débito técnico, você deve primeiro avaliar os riscos associados a cada problema identificado. **Problemas mais críticos, como problemas de segurança ou problemas que afetam o desempenho do software, devem ser tratados com mais urgência do que problemas menos críticos.** Outra forma de priorizar o tratamento do débito técnico é avaliar o impacto que cada problema tem no projeto como um todo.

Problemas que afetam mais áreas do software ou que afetam a capacidade de adicionar novas funcionalidades devem ter uma prioridade maior do que problemas que afetam apenas uma parte específica do software. Para planejar e executar a correção do débito técnico, é importante ter um processo claro e bem definido. **Isso envolve identificar os problemas, avaliar a prioridade de cada problema e planejar as etapas necessárias para corrigi-los.**

Uma das melhores práticas para gerenciar o débito técnico é estabelecer um processo contínuo de revisão e correção. Isso significa que você deve realizar revisões regulares do código, identificar os problemas e corrigi-los o mais rápido possível. Além disso, é importante envolver toda a equipe no processo de gerenciamento do débito técnico, para garantir que todos estejam cientes dos problemas e trabalhem juntos para resolvê-los.

Outra prática importante é manter um registro de todos os problemas de débito técnico identificados e corrigidos. Isso pode ajudar a identificar padrões e tendências ao longo do tempo e a avaliar a eficácia do processo de gerenciamento do débito técnico. **Por fim, é importante ter em**

mente que o gerenciamento do débito técnico é um processo contínuo e nunca termina: sempre haverá novos problemas e desafios a serem enfrentados.

No entanto, com um processo claro e uma equipe comprometida, é possível gerenciar o débito técnico de forma eficaz e garantir a qualidade e a escalabilidade do software ao longo do tempo. **Prevenir a acumulação de débito técnico é a melhor forma de garantir que seu software seja eficiente, escalável e fácil de manter a longo prazo.** Aqui estão algumas estratégias para evitar o acúmulo de débito técnico:

ESTRATÉGIAS	DESCRIÇÃO
ESTABELEÇA PADRÕES DE CÓDIGO E BOAS PRÁTICAS DE DESENVOLVIMENTO	Ter padrões de código claros e boas práticas de desenvolvimento estabelecidos desde o início é uma ótima maneira de garantir que o código seja claro, fácil de entender e fácil de manter.
FAÇA REVISÕES DE CÓDIGO REGULARES	Revisões de código regulares ajudam a identificar problemas de qualidade do código antes que eles se tornem problemas maiores. Além disso, elas promovem a colaboração da equipe e ajudam a garantir que todos estejam trabalhando dentro dos mesmos padrões e práticas.
INVISTA EM TESTES DE QUALIDADE	Testes automatizados e manuais são uma ótima maneira de garantir que o software esteja funcionando corretamente e atendendo às necessidades dos usuários.
PLANEJE PARA ESCALABILIDADE	Ao desenvolver um software, é importante pensar em como ele pode ser escalável no futuro. Isso significa pensar em como o software pode lidar com grandes quantidades de dados e usuários, bem como garantir que ele possa ser facilmente mantido e atualizado.
USE FERRAMENTAS DE AUTOMAÇÃO	Ferramentas de automação podem ajudar a reduzir o tempo necessário para realizar tarefas repetitivas e propensas a erros. Isso pode incluir ferramentas de integração contínua, ferramentas de análise de código e ferramentas de teste automatizado.
MANTENHA UM REGISTRO DE DÍVIDA TÉCNICA	Manter um registro de dívida técnica pode ajudar a garantir que os problemas sejam identificados e tratados o mais rápido possível.

(Câmara de Goiânia/GO – 2018) Leia o texto a seguir extraído da Internet.

Se o débito técnico não é pago, ele pode acumular, tornando mais difícil implementar mudanças posteriores.

No contexto desta informação, o débito técnico:

a) pode ser eliminado durante a construção de software sem dependência do projeto (design) do software.
b) pode ser eliminado por alteração no projeto (design) sem repercussão na implementação.

c) é uma questão de projeto (design) com repercussão na funcionalidade do software.
e) tem impacto na evolução do software.

_____

Comentários: no contexto desta informação, o débito técnico tem impacto na evolução do software. Quando o débito técnico não é tratado, ele se acumula e pode tornar mais difícil a implementação de mudanças posteriores. Isso pode afetar negativamente a evolução do software, levando a atrasos no lançamento, custos adicionais e, possivelmente, um produto final de menor qualidade. Logo, é importante gerenciar o débito técnico de maneira eficaz para garantir que o software possa evoluir de maneira consistente e sem problemas (Letra D).

(TJ/PE – 2012) No contexto de programação ágil XP, um débito técnico é descrito como o:

a) número de pontos funcionais não entregues no último período.
b) custo homem/hora da equipe técnica para um determinado projeto.
c) método de modificação do código fonte, com alteração do seu comportamento, porém sem alteração de seu significado.
d) dispêndio relacionado ao desenvolvimento, teste ou entrega da parte funcional do sistema.
e) total de desenvolvimento feito de maneira rápida e simples sem, às vezes, levar em consideração testes e arquitetura do sistema.

_____

Comentários: o débito técnico em XP é entendido como uma dívida técnica que é adquirida quando o desenvolvimento é realizado de forma rápida e simplificada, muitas vezes sem levar em consideração os testes e a arquitetura adequados do sistema. Essa dívida técnica pode se acumular e, se não for tratada, pode levar a problemas futuros no projeto. Logo, é importante gerenciar o débito técnico para garantir que ele não se acumule a ponto de afetar negativamente o projeto (Letra E).

(CEBRASPE / SERPRO – 2023) A dívida técnica pode ser vista como um empréstimo que o time de desenvolvimento faz para si mesmo com o objetivo de acelerar o processo de desenvolvimento, porém, por ser considerada uma estratégia ruim, deve ser evitada, devido a suas possíveis consequências negativas.

_____

Comentários: a dívida técnica é, de fato, comparada a um empréstimo que o time de desenvolvimento faz para acelerar a entrega, consciente de que terá que "pagar" esse débito mais tarde com esforços adicionais de refatoração ou correções. No entanto, ela não é necessariamente uma estratégia ruim. Pode ser uma escolha deliberada e útil em determinadas situações, como quando há uma necessidade urgente de entrega. Por conta da péssima redação, o item foi anulado sob a seguinte justificativa: "A ausência de informações relevantes na redação do item prejudicou seu julgamento objetivo" (Anulado).

ÁGIL EM ESCALA

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Ágil Escalado refere-se à aplicação de princípios ágeis em um ambiente de alta escala, particularmente em grandes projetos, programas e portfólios, ou em organizações inteiras que envolvem múltiplas equipes trabalhando simultaneamente em diferentes componentes de um produto/serviço. O objetivo é manter a entrega rápida de valor, resposta às mudanças e adaptabilidade mesmo à medida que o escopo do trabalho e o tamanho da organização aumentam.

Por que o ágil precisa eventualmente ser escalado? **Bem, à medida que as organizações crescem e seus projetos se tornam mais complexos, surgem novos desafios que não podem ser abordados efetivamente apenas pelos métodos ágeis em nível de equipe, como Scrum ou Kanban, que são mais adequados para pequenas equipes.** Nesse sentido, a agilidade precisa ser de certa forma escalada para:

- **Coordenar Múltiplas Equipes:** garantir que todas as equipes estejam alinhadas com a visão geral do produto e os objetivos da organização, evitando esforços duplicados e garantindo que o trabalho de uma equipe complemente o das outras;

- **Gerenciar Complexidade:** projetos grandes e complexos podem envolver múltiplos componentes interdependentes, requerendo uma coordenação detalhada entre as equipes para garantir a entrega coesa do produto.

- **Sustentar Agilidade em Escala:** preservar os princípios ágeis, como a capacidade de responder rapidamente a mudanças, mesmo em uma grande organização com processos potencialmente mais rígidos.

Escalar práticas ágeis apresenta vários desafios que precisam ser superados para **manter a eficácia da metodologia em grandes organizações**:

DESAFIOS	DESCRIÇÃO
COMUNICAÇÃO E COORDENAÇÃO	Assegurar comunicação eficaz e coordenação entre múltiplas equipes ágeis, especialmente quando estão distribuídas geograficamente, é um desafio significativo. Isso inclui manter todos na mesma página em relação a objetivos, progresso e mudanças.
INTEGRAÇÃO E ENTREGA CONTÍNUA	Integrar e entregar trabalho de forma contínua de múltiplas equipes em um único produto coeso pode ser complexo, exigindo práticas robustas de CI/CD (Integração Contínua/Entrega Contínua).
CULTURA E MENTALIDADE	Mudar a cultura organizacional para adotar completamente os princípios ágeis em todos os níveis da organização, especialmente em níveis de gestão que podem estar acostumados a abordagens mais tradicionais de gerenciamento de projetos.

GESTÃO DE DEPENDÊNCIAS	Identificar e gerenciar dependências entre as equipes e seus entregáveis é crucial para evitar atrasos e garantir uma entrega suave.
PADRONIZAÇÃO X FLEXIBILIDADE	Encontrar o equilíbrio certo entre padronizar práticas ágeis para garantir alinhamento e consistência, enquanto se mantém flexível o suficiente para permitir que as equipes adaptem práticas às suas necessidades específicas.

Principais Frameworks

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Os frameworks de Ágil Escalado são projetados para ajudar organizações a aplicarem práticas ágeis além do nível da equipe, permitindo a agilidade em projetos, programas e portfolios maiores. Cada framework tem seus próprios princípios, práticas e ferramentas projetadas para facilitar a coordenação entre equipes, garantir alinhamento estratégico, e promover uma cultura de melhoria contínua em toda a organização. Vejamos...

SAFe (Scaled Agile Framework)

SAFe é um framework abrangente que oferece uma estrutura detalhada para escalar ágil em todos os níveis da organização, incluindo equipe, programa, valor de solução e portfólio. Tende a ser mais prescritivo, com várias camadas e roles bem definidos, processos e artefatos. Isso o torna adequado para organizações muito grandes que necessitam de uma estrutura formal para coordenar muitas equipes e projetos.

Nexus

Nexus é um framework desenvolvido pela Scrum.org, projetado para escalar o Scrum para projetos que envolvem 3 a 9 equipes Scrum trabalhando no mesmo produto. É menos prescritivo e complexo que o SAFe, focando principalmente em como coordenar o trabalho entre múltiplas equipes Scrum. Ele adiciona o mínimo de roles, eventos e artefatos necessários para facilitar a escalabilidade.

LeSS (Large Scale Scrum)

LeSS é um framework que estende o Scrum para múltiplas equipes trabalhando no mesmo produto. Ela foca em simplicidade, tentando aplicar os princípios do Scrum em larga escala com o mínimo de adição de novos elementos. LeSS enfatiza a importância de ter menos roles, menos artefatos e menos processos. Existem duas variantes: LeSS para até 8 equipes e LeSS Huge para mais de 8 equipes.

DaD (Disciplined Agile Delivery)

DaD é um framework de processo decisório que fornece uma abordagem pragmática para a entrega ágil de projetos. Ela é mais abrangente que Scrum, cobrindo todas as fases do ciclo de vida do projeto, desde a concepção até a entrega. DaD enfatiza a flexibilidade e a capacidade de escolha, oferecendo várias opções para práticas de gerenciamento de projetos, desenvolvimento de software, e outras áreas, permitindo que as organizações customizem sua abordagem ágil.

Scrum@Scale

Desenvolvido pelo co-criador do Scrum, Jeff Sutherland, Scrum@Scale é uma framework para escalar o Scrum através de uma organização inteira. Ela é projetada para adaptar-se a qualquer tamanho de organização e a qualquer indústria. Scrum@Scale consiste em dois ciclos: o Ciclo de Scrum Master, focado em como as equipes trabalham juntas, e o Ciclo de Product Owner, focado em como os backlogs são priorizados entre todas as equipes.

NEXUS

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Nexus é um framework para desenvolvimento e sustentação de iniciativas de entrega de produtos e de softwares em escala. Ele usa o Scrum como alicerce para sua construção e possui um guia que contêm a sua definição. Esta definição consiste nos papéis, eventos, artefatos e regras do Nexus que colocam esses elementos juntos. Ken Schwaber e a Scrum.org desenvolveram o Nexus, sendo o Guia do Nexus escrito e fornecido por eles.

Em primeiro lugar, é importante mencionar que a palavra "nexus" significa um relacionamento ou conexão entre pessoas ou coisas. **Nesse sentido, Nexus é um framework constituído de papéis, eventos, artefatos e regras que os unem e entrelaçam junto o trabalho de aproximadamente três a nove Times Scrum em um único Backlog do Produto para construir um incremento integrado que alcance uma meta.**

A entrega de software é complexa e a integração do trabalho em um software funcional tem muitos artefatos e atividades que precisam ser coordenados para se criar um resultado "Pronto". **O trabalho precisa ser organizado, sequenciado, ter as dependências resolvidas e os resultados faseados.** Muitos desenvolvedores de software têm usado o framework Scrum para trabalhar coletivamente para desenvolver e entregar um incremento de software funcional.

No entanto, se mais de um Time Scrum está trabalhando no mesmo Backlog do Produto e no mesmo repositório de código para um produto, dificuldades frequentemente emergem. *Se os desenvolvedores não estão fisicamente lado a lado no mesmo time, como eles poderão se comunicar quando estiverem fazendo um trabalho que afete um ao outro? Se eles trabalharem em times diferentes, como integrarão seu trabalho e testarão o incremento integrado?*

Estes desafios aparecem quando dois times estão integrando seus trabalhos em um mesmo incremento, e tornam significativamente mais difíceis quando três ou mais times Scrum integram seu trabalho em um único incremento. **Existem muitas dependências que emergem entre o trabalho de múltiplos times que colaboram para criar um incremento completo e "Pronto" pelo menos a cada Sprint.** Essas dependências são relacionadas a:

DEPENDÊNCIA	DESCRIÇÃO
REQUERIMENTOS	O escopo dos requerimentos pode se sobrepor, e a maneira no qual eles são implementados podem também afetar um ao outro. Este conhecimento pode ser considerado na ordenação do Backlog do Produto e na seleção de itens do Backlog do Produto.
DOMÍNIO DE CONHECIMENTO	As pessoas nos times têm conhecimento de vários negócios e sistemas de computadores. Este conhecimento pode ser distribuído entre os Times Scrum para garantir que os times tenham o

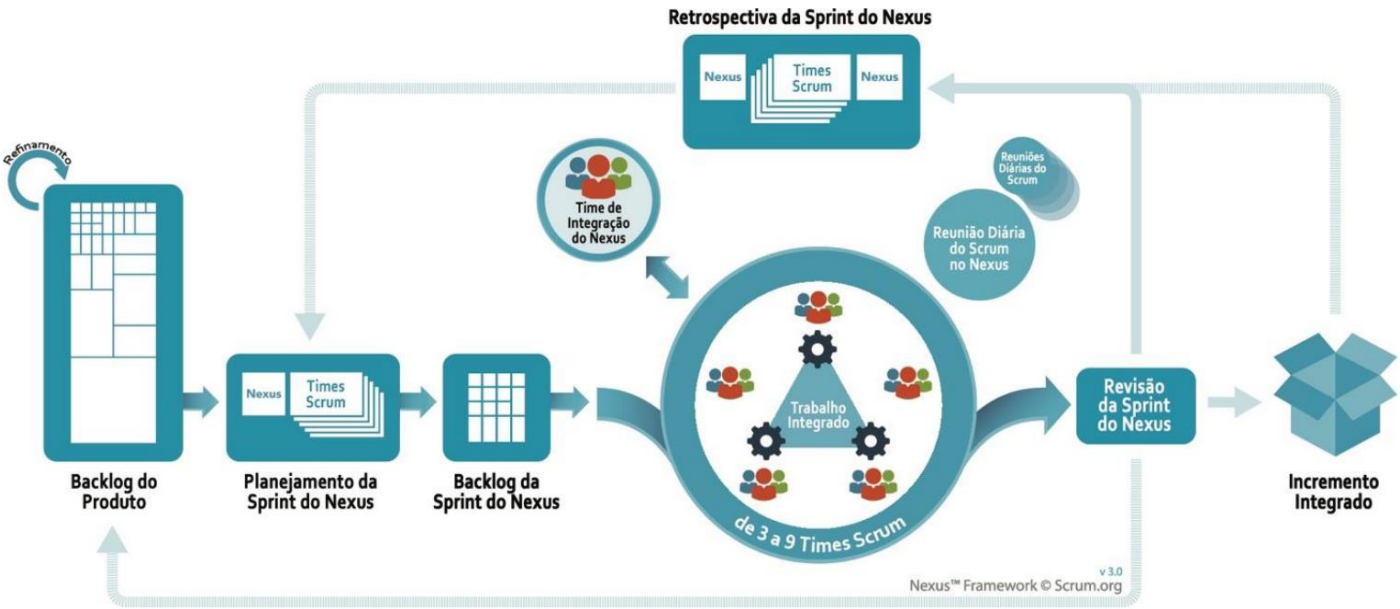
	conhecimento que eles precisam para fazer seu trabalho, e minimizar interrupções entre os times durante uma Sprint.
SOFTWARE E ARTEFATOS DE TESTE	Os requerimentos são ou serão instanciados no software.

À medida que os requerimentos, o conhecimento dos membros do time, e os artefatos de software são mapeados para os mesmos Times Scrum, os times **podem reduzir o número de dependências entre eles**. Quando o desenvolvimento utilizando Scrum é escalado, essas dependências de requerimentos, domínio de conhecimento, e artefatos de software podem direcionar a organização dos Times de Desenvolvimento. À medida que isso ocorre, a produtividade vai sendo otimizada.

Framework Nexus

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Nexus é um framework de processo para múltiplos Times Scrum trabalharem juntos para criarem um Incremento Integrado. Ele é consistente com o Scrum e suas partes serão familiares para aqueles que tem usado Scrum. A diferença é que mais atenção é dada para as dependências e interoperação entre os Times Scrum, entregando pelo menos um Incremento Integrado e "Pronto" a cada Sprint.



Como mostrado no gráfico, o Nexus consiste em:

COMPONENTES	DESCRIÇÃO
PAPEIS	Um novo papel, o Time de Integração do Nexus, este existe para coordenar, treinar e supervisionar a aplicação do Nexus e a operação do Scrum para que os melhores resultados sejam obtidos. O Time de Integração do Nexus consiste em um Product Owner, um Scrum Master e os Membros do Time de Integração do Nexus.
ARTEFATOS	Todos os Times Scrum usam o mesmo e único Backlog do Produto. Assim que os Itens do Backlog do Produto são refinados e estão "Preparados", indicadores de qual time irá fazer o trabalho na Sprint se tornam transparentes. Um novo artefato, o Backlog da Sprint do Nexus, existe para auxiliar com a transparência durante a Sprint. Todos os Times Scrum mantem seu Backlog da Sprint individual.
EVENTOS	Eventos são anexados, colocados em volta, ou substituem (no caso da Revisão da Sprint) os eventos normais do Scrum para ampliá-los. Uma vez modificados, eles atendem tanto o esforço geral de todos os Times Scrum no Nexus, quanto de cada time individualmente.

Fluxo de Processo

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Conforme já vimos, o Nexus consiste em múltiplos Times Scrum multifuncionais trabalhando juntos para entregar um potencial incremento integrado e possível de ser entregue pelo menos até o final de cada Sprint. **Baseado nas dependências, os times podem selecionar os membros mais apropriados para fazer um trabalho específico.** Dito isso, vejamos a seguir o fluxo de processo do Nexus:

FLUXO DE PROCESSO	DESCRIÇÃO
REFINAR O PRODUCT BACKLOG	O Product Backlog precisa ser decomposto para que as dependências sejam identificadas, removidas ou minimizadas. Os Itens do Backlog do Produto são refinados em pequenos pedaços de funcionalidades e o provável time para fazer o trabalho deve ser identificado.
PLANEJAMENTO DE SPRINT DO NEXUS	Representantes apropriados de cada Time Scrum se reúnem para discutir e revisar o refinamento do Backlog do Produto. Eles selecionam os Itens do Backlog do Produto para cada time. Cada Time Scrum então planeja sua própria Sprint, interagindo com os outros times quando apropriado. O resultado é um conjunto de Metas de Sprint alinhadas à Meta global da Sprint do Nexus, o Backlog da Sprint de cada Time Scrum e um único Backlog da Sprint do Nexus. O Backlog da Sprint do Nexus torna transparente o trabalho de todos os itens selecionados do Backlog do Produto dos Times Scrum, assim como qualquer dependência.
TRABALHO DE DESENVOLVIMENTO	Todos os times frequentemente integram seu trabalho em um ambiente comum que pode ser testado para garantir que a integração está feita.
REUNIÃO DIÁRIA	Representantes apropriados de cada Time de Desenvolvimento se encontram diariamente para identificar se existe alguma questão de integração. Se identificado, essa informação é transferida de volta para cada Reunião Diária dos Times Scrum. Os Times Scrum então usam sua Reunião Diária para criar um plano para o dia, estando certos de trabalharem as questões de integração que emergiram durante a Reunião Diária do Nexus.
REVISÃO DA SPRINT	A Revisão de Sprint do Nexus é feita no final da Sprint para promover comentários e opiniões sobre o incremento integrado que um Nexus construiu através da Sprint. Todos os Times Scrum individualmente se encontram com as partes interessadas para revisarem o Incremento Integrado. Ajustes podem ser feitos no Backlog do Produto.
RETROSPECTIVA DA SPRINT	Representantes apropriados de cada Time Scrum se encontram para identificar os desafios compartilhados. Então, cada Time Scrum realiza sua Reunião de Retrospectiva do Scrum individualmente. Representantes apropriados de cada time se encontram novamente para discutir quaisquer ações necessárias baseadas nos desafios compartilhados a fim de fornecer inteligência de baixo para cima.

Papeis do Nexus

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Os papéis do Nexus, eventos, e artefatos herdam os atributos de propósito e intenção dos seus correspondentes papéis, eventos, e artefatos do Scrum conforme documentado no Guia do Scrum.

Time de Integração

O Time de Integração do Nexus é responsável por garantir que um Incremento Integrado (o trabalho combinado e completado pelo Nexus) seja produzido pelo menos a cada Sprint. Os Times Scrum são responsáveis por entregar incrementos "Prontos" de produto potencialmente possíveis de serem entregues, como indicado no Scrum. Todos os papeis para os membros dos Times Scrum são prescritos no Guia do Scrum. O Time de Integração do Nexus é composto de:

- O Product Owner
- Um Scrum Master
- Um ou mais Membros do Time de Integração do Nexus.

Membros do Time de Integração Nexus são frequentemente membros de um Time Scrum Individual daquele Nexus. Se este for o caso, eles devem dar prioridade para seu trabalho no Time de Integração do Nexus; a participação no Time de Integração do Nexus tem precedência em relação à participação individual no Time Scrum. Essa preferência ajuda a garantir que o trabalho que resolva questões que afetem muitos times tenha prioridade.

A composição do Time de Integração Nexus pode mudar ao longo do tempo para refletir as necessidades do Nexus. **Atividades comuns que o Time de Integração do Nexus pode realizar incluem mentoria, consultoria, e destacar o conhecimento de dependências e problemas entre os times.** Este time pode também realizar trabalho do Backlog do Produto. O Time Scrum endereça questões de integração dentro do Nexus.

O Time de Integração do Nexus provê um ponto focal de integração para o Nexus. **Esta Integração inclui resolver qualquer restrição técnica ou não técnica entre os times que possa impedir a habilidade do Nexus de entregar constantemente um Incremento Integrado.** Eles podem usar a inteligência de baixo para cima do Nexus para alcançar a solução. Falemos agora sobre o Product Owner.

Product Owner (PO)

Um Nexus trabalha um único Backlog do Produto, e assim como descrito no framework Scrum, um Backlog do Produto possui um único Product Owner que possui a última palavra sobre o conteúdo do Backlog. **O Product Owner é responsável por maximizar o valor do produto e do trabalho desempenhado e integrado pelos Times Scrum no Nexus.** O Product Owner é um membro do Time de Integração do Nexus.

O Product Owner é responsável por gerenciar o Backlog do Produto para que o máximo de valor seja derivado através do Incremento Integrado criado pelo Nexus. Como isso é feito pode variar amplamente entre organizações, Nexus, Times Scrum e indivíduos.

Scrum Master

O Scrum Master no Time de Integração do Nexus possui a responsabilidade geral de garantir que o framework Nexus seja entendido e publicado oficialmente. **Este Scrum Master pode também ser o Scrum Master de um ou mais times Scrum naquele Nexus.**

Membros do Time de Integração

O Time de Integração do Nexus consiste em profissionais que são capacitados no uso de ferramentas, várias práticas e nas áreas gerais de engenharia de sistemas. O Time de Integração do Nexus garante que os Times Scrum dentro do Nexus entendam e implementem as práticas e ferramentas necessárias para detectar dependências e frequentemente integrar todos os artefatos à definição de "Pronto".

Os Membros do Time de Integração do Nexus são responsáveis por treinar e orientar os Times Scrum no Nexus para que adquiram, implementem e aprendam essas práticas e ferramentas. Adicionalmente, o Time de Integração do Nexus treina os Times Scrum individuais no desenvolvimento, infraestrutura e padrões de arquitetura necessários e requeridos pela organização para garantir o desenvolvimento de Incrementos Integrados de qualidade.

Se suas responsabilidades primárias estão satisfeitas, os Membros do Time de Integração podem também trabalhar como membros do Time de Desenvolvimento em um ou mais Times Scrum.

Eventos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

A duração dos eventos do Nexus é guiada pelos tamanhos dos seus eventos correspondentes no Guia do Scrum. **Eles são time-boxes também de seus eventos correspondentes no Scrum.**

Refinamento

Refinamento do Backlog do Produto em escala serve para dois propósitos. Ajudar os Times Scrum a preverem qual time irá entregar cada item do Backlog do Produto, e identificar dependências entre esses times. Esta transparência permite que os times monitorem e minimizem dependências. **O refinamento dos Itens do Backlog do Produto continua até que eles estejam suficientemente independentes para serem trabalhados por um único Time Scrum sem conflitos excessivos.**

O número, frequência, duração e participantes do Refinamento são baseados nas dependências e incertezas herdadas do Backlog do Produto. Os Itens do Backlog do Produto passam por diferentes níveis de decomposição desde requisições muito grandes e vagas até o trabalho acionável que um único Time Scrum possa entregar dentro de uma Sprint. **O Refinamento é contínuo ao longo da Sprint quando necessário e apropriado.**

Ele irá continuar dentro de cada Time Scrum a fim de que os Itens do Backlog do Produto estejam preparados para seleção no evento de Planejamento do Nexus.

Planejamento da Sprint

O propósito da Reunião de Planejamento Nexus é coordenar as atividades de todos os Times Scrum no Nexus para uma única Sprint. O Product Owner provê domínio de conhecimento e guia a seleção e a priorização das decisões. O Backlog do Produto deve ser adequadamente refinado com as dependências identificadas e removidas ou minimizadas antes do Planejamento da Sprint do Nexus.

Durante o Planejamento da Sprint do Nexus, representantes apropriados de cada Time Scrum validam e fazem os ajustes na ordenação do trabalho criado durante os eventos de Refinamento. Todos os membros dos Times Scrum devem participar para minimizar os problemas de comunicação. **O Product Owner discute a Meta da Sprint do Nexus durante o Planejamento da Sprint do Nexus.**

A Meta da Sprint do Nexus descreve o propósito que será alcançado pelos Times Scrum durante a Sprint. **Uma vez que o trabalho geral para o Nexus é entendido, o Planejamento da Sprint do Nexus continua com cada Time Scrum realizando seu próprio Planejamento da Sprint separadamente.** Os Times Scrum devem continuar compartilhando novas dependências encontradas com outros Times Scrum no Nexus.

O Planejamento da Sprint do Nexus está completa quando cada time tiver terminado seus eventos de Planejamento da Sprint individuais. Novas dependências podem emergir durante o Planejamento da Sprint. Estas devem ser transparentes e minimizadas. A sequência do trabalho entre os times pode também ser ajustada. Um refinamento adequado do Backlog do Produto minimizará o surgimento de novas dependências durante o Planejamento da Sprint.

Todos os itens do Backlog do Produto selecionados para a Sprint e suas dependências devem ser transparentes no Backlog da Sprint do Nexus.

Meta da Sprint

A Meta da Sprint do Nexus é um objetivo definido para a Sprint. **Ela é a soma de todo o trabalho e Metas das Sprints dos Times Scrum do Nexus.** O Nexus deve demonstrar a funcionalidade que ele tem desenvolvida "Pronta" para alcançar a Meta da Sprint do Nexus na Revisão da Sprint do Nexus com a finalidade de receber comentários e observações do stakeholder.

Reunião Diária

A Reunião Diária do Nexus é um evento para os representantes adequados dos Times de Desenvolvimento individuais **inspecionarem a atual situação do Incremento Integrado e para identificarem questões de integração ou recentes descobertas sobre as dependências ou impactos entre os times**. Durante a Reunião Diária do Nexus, os participantes devem focar no impacto de cada time no Incremento Integrado e discutir:

- *O trabalho do dia anterior foi integrado com sucesso? Se não, por que não?*
- *Que novas dependências ou impactos foram identificadas?*
- *Que informações precisam ser compartilhadas entre as equipes no Nexus?*

Os Times de Desenvolvimento usam a Reunião Diária do Nexus para inspecionar o progresso em direção à Meta da Sprint. Pelo menos a cada Reunião Diária, o Backlog da Sprint deve ser ajustado para refletir o atual entendimento do trabalho dos Times Scrum dentro do Nexus. Os times Scrum individuais então pegam de volta as questões e trabalhos identificados durante a Reunião Diária para seus Times Scrum individuais planejarem dentro das suas Reuniões Diárias individuais.

Revisão da Sprint

A Revisão da Sprint do Nexus é mantida no final da Sprint e proporciona retorno do incremento integrado que o Nexus construiu ao longo da Sprint e para adaptar o Backlog do Produto se necessário. A Revisão da Sprint do Nexus substitui as Reuniões individuais de Revisão do Scrum, porque todo o incremento integrado é o foco para capturar o retorno das partes interessadas. Pode não ser possível mostrar todo o trabalho completado em detalhe.

Técnicas podem ser necessárias para maximizar o retorno das partes interessadas. O resultado da Revisão da Sprint do Nexus é o Backlog do Produto revisado.

Retrospectiva da Sprint

A Retrospectiva da Sprint do Nexus é uma oportunidade formal para um Nexus inspecionar e adaptar a si mesmo e criar um plano para que melhorias entrem em vigor na próxima Sprint para garantir melhoria contínua. A Retrospectiva da Sprint do Nexus ocorre depois da Revisão da Sprint do Nexus e antes do próximo Planejamento da Sprint do Nexus. Consiste basicamente em três partes:

1. A primeira parte é a oportunidade para os representantes adequados de todo o Nexus conhecerem e identificarem questões que tem impacto além de um único time. O propósito é fazer com que todas as questões compartilhadas sejam transparentes para todos os times Scrum.

2. A segunda parte consiste em cada time Scrum manter sua própria Retrospectiva da Sprint como descrito no Framework Scrum. Eles podem usar as questões levantadas na primeira parte da Retrospectiva do Nexus como entrada para as discussões do time. Os Times Scrum individuais devem formar ações para endereçar estas questões durante suas Retrospectivas da Sprint dos Times Scrum individuais.

3. No final, a terceira parte é a oportunidade para os representantes adequados dos Times Scrum se reunirem novamente e chegarem a um acordo sobre como monitorar e controlar as ações identificadas.

Porque são disfunções comuns de escala, toda Retrospectiva deve pautar os seguintes assuntos:

- *Foi deixado de realizar algum trabalho? O Nexus gerou débito técnico?*
- *Todos os artefatos, em particular os códigos, foram com frequência (quantas vezes por dia) integrados com sucesso?*
- *O software foi construído com sucesso, testado e implantado com a frequência suficiente para impedir o acúmulo esmagador de dependências não resolvidas?*

Para as questões acima, aponte se necessário:

- *Por que isso aconteceu?*
- *Como pode ser resolvido o débito técnico?*
- *Como a recorrência pode ser evitada*

Artefatos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Artefatos representam o trabalho ou valor que fornece transparência e oportunidades para inspeção e adaptação, assim como descrito no Guia do Scrum.

Backlog do Produto

Há um único Backlog do Produto para todo o Nexus e todos os Times Scrum. O Product Owner é o responsável pelo Backlog do Produto, incluindo seu conteúdo, disponibilidade e ordenação. Em escala, o Backlog do Produto deve ser entendido em um nível onde as dependências possam ser detectadas e minimizadas. Para suportar a resolução de questões, os itens do Backlog do Produto são decompostos em uma granularidade chamada funcionalidades "finamente fatiadas".

Os itens do Backlog do Produto são considerados "preparados" para a Reunião de Planejamento da Sprint quando os Times Scrum podem selecionar itens para serem feitos pelos sem ou com uma dependência mínima de outros Times Scrum.

Backlog da Sprint

O Backlog da Sprint do Nexus é composto pelos itens de Backlog do Produto dos Backlogs das Sprints dos Times Scrum individuais. Este é usado para destacar as dependências e o fluxo de trabalho durante a Sprint. Este é atualizado pelo menos uma vez ao dia, frequentemente como parte da Reunião Diária do Nexus.

Incremento Integrado

Um incremento integrado representa a soma atual de todos os trabalhos integrados completados para o Nexus. Um Incremento Integrado deve ser utilizável e potencialmente possível de ser entregue que significa que este deve atender a definição de "Pronto". Um Incremento Integrado é inspecionado na Revisão da Sprint do Nexus.

Transparência do Artefato

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Assim como seu elemento de base, Scrum, o Nexus é baseado na transparência**. O Time de Integração do Nexus trabalha com os Times Scrum dentro de um Nexus e da organização para garantir que a transparência é aparente em todos os artefatos e que o estado integrado dos incrementos é amplamente compreendido.** As decisões tomadas com base no estado dos artefatos do Nexus só são eficazes com o nível de transparência do artefato.

Informações incompletas ou parciais levam a decisões incorretas ou defeituosas. O impacto destas decisões pode ser ampliado na escala do Nexus. Software deve ser desenvolvido de modo que as dependências sejam detectadas e resolvidas antes que o débito técnico se torne inaceitável para o Nexus. A falta de uma completa transparência tornará impossível guiar o Nexus eficazmente a fim de minimizar o risco e maximizar o valor.

Definição de Pronto

O Time de Integração é responsável pela definição de "Pronto" que pode ser aplicado para o incremento integrado desenvolvido em cada sprint. Todos os Times Scrum aderem a esta definição de "Pronto". O incremento é "Pronto" somente quando é integrado, utilizável e potencialmente possível de ser entregue pelo Product Owner. Times Scrum individuais podem escolher por aplicar uma definição de "Pronto" mais rigorosa dentro de seus próprios times, mas não podem aplicar critérios menos rigorosos do que os acordados para o incremento.

Obs: o Nexus é gratuito e, assim como acontece com o Scrum, papéis, artefatos, eventos e regras do Nexus são imutáveis. É possível implementar partes do Nexus, mas o resultado **não é Nexus**.

O Nexus é frequentemente utilizado em ambientes onde múltiplos times Scrum precisam trabalhar juntos para entregar um produto complexo. Um exemplo real de utilização do Nexus poderia ser no desenvolvimento de um grande sistema de software para uma instituição financeira, como um novo sistema de banking online que requer integração com sistemas de back-end existentes, cumprimento de normas de segurança rígidas, e uma interface de usuário amigável e acessível.

Neste cenário, diversos times Scrum poderiam ser responsáveis por diferentes componentes do sistema, como desenvolvimento de frontend, integração de back-end, segurança da informação, e compliance. **O Nexus seria empregado para coordenar o trabalho desses times, garantindo que todas as partes do sistema se integrem harmoniosamente e que o produto final seja entregue de forma coesa e dentro do prazo.**

O Time de Integração do Nexus, incluindo um PO, um SM e Membros do Time de Integração, coordenaria as atividades de integração, ajudaria a resolver dependências entre os times, e garantiria que um Incremento Integrado "Pronto" seja produzido a cada sprint. Isso pode incluir a

coordenação de atividades de teste de integração, a priorização de itens no Backlog do Produto que afetam múltiplos times, e o suporte à resolução de impedimentos que possam afetar a entrega.

QUESTÕES COMENTADAS

1. **(PROF. DIEGO / INÉDITA – 2024)** O Nexus é um framework que permite a colaboração de até nove Times Scrum em um único Backlog de Produto para construir um incremento integrado.

Comentários:

O Nexus é um framework projetado para entrelaçar o trabalho de aproximadamente três a nove Times Scrum em um único Backlog do Produto, com o objetivo de construir um incremento integrado que atenda a uma meta. Isso facilita a colaboração e integração entre múltiplos times trabalhando no mesmo projeto.

Gabarito: Correto

---

2. **(PROF. DIEGO / INÉDITA – 2024)** A palavra "nexus" não tem relação com a natureza do framework, sendo apenas um nome escolhido arbitrariamente.

Comentários:

A palavra "nexus" significa um relacionamento ou conexão entre pessoas ou coisas, o que reflete diretamente a essência do framework Nexus. Ele é projetado para conectar e coordenar o trabalho de múltiplos Times Scrum em um projeto compartilhado, o que está em alinhamento direto com o significado da palavra.

Gabarito: Errado

---

3. **(PROF. DIEGO / INÉDITA – 2024)** O Nexus elimina completamente as dependências entre os times trabalhando em um incremento integrado.

Comentários:

Enquanto o Nexus é projetado para ajudar a gerenciar e coordenar o trabalho de múltiplos Times Scrum, ele não elimina completamente as dependências entre os times. O framework ajuda a identificar, organizar e resolver essas dependências, especialmente aquelas relacionadas a requerimentos, domínio de conhecimento e artefatos de software, mas a existência dessas dependências é inerente à natureza do trabalho colaborativo em escala.

Gabarito: Errado

---

4. **(PROF. DIEGO / INÉDITA – 2024)** No Nexus, não há necessidade de organizar e sequenciar o trabalho, uma vez que múltiplos times Scrum trabalham de forma independente.

Comentários:

Existe a complexidade da entrega de software, especialmente quando múltiplos times Scrum trabalham juntos em um mesmo Backlog do Produto. Há uma necessidade explícita de organizar, sequenciar o trabalho, resolver dependências e fasear os resultados para criar um incremento "Pronto". O Nexus provê uma estrutura para essa coordenação, sublinhando a importância de tais práticas para o sucesso da integração de trabalhos em software funcional.

Gabarito: Errado

---

5. **(PROF. DIEGO / INÉDITA – 2024)** A distribuição do conhecimento do domínio entre os Times Scrum é uma estratégia do Nexus para minimizar interrupções durante uma Sprint.

Comentários:

Uma das dependências identificadas no Nexus é o domínio de conhecimento. Distribuir esse conhecimento entre os Times Scrum é uma forma de garantir que os times tenham as informações necessárias para realizar seu trabalho eficientemente, minimizando as interrupções entre os times durante uma Sprint. Isso facilita a colaboração e aumenta a produtividade ao otimizar a utilização dos recursos de conhecimento disponíveis.

Gabarito: Correto

---

6. **(PROF. DIEGO / INÉDITA – 2024)** O Time de Integração do Nexus é responsável apenas por supervisionar a aplicação do Nexus.

Comentários:

O Time de Integração do Nexus tem responsabilidades que vão além de apenas supervisionar a aplicação do Nexus. Este time também é encarregado de coordenar e treinar os Times Scrum na aplicação do Nexus e na operação do Scrum, visando garantir que os melhores resultados sejam alcançados.

Gabarito: Errado

---

7. **(PROF. DIEGO / INÉDITA – 2024)** No Nexus, cada Time Scrum trabalha com seu próprio e exclusivo Backlog do Produto.

Comentários:

No Nexus, todos os Times Scrum trabalham a partir do mesmo e único Backlog do Produto. Isso é feito para assegurar que, à medida que os Itens do Backlog são refinados e preparados para a Sprint,

fica claro qual time será responsável por trabalhar em determinados itens, promovendo transparência e colaboração entre os times.

Gabarito: Errado

---

8. **(PROF. DIEGO / INÉDITA – 2024)** A Revisão da Sprint no Nexus é realizada de forma idêntica ao Scrum tradicional, sem modificações.

Comentários:

No Nexus, eventos como a Revisão da Sprint são modificados ou adaptados para atender tanto ao esforço geral de todos os Times Scrum dentro do Nexus quanto de cada time individualmente. Isso implica que a Revisão da Sprint no Nexus pode ser alterada para abranger as necessidades de integração e colaboração entre múltiplos Times Scrum.

Gabarito: Errado

---

9. **(PROF. DIEGO / INÉDITA – 2024)** O Backlog da Sprint do Nexus é um artefato que substitui os Backlogs da Sprint individuais de cada Time Scrum.

Comentários:

O Backlog da Sprint do Nexus não substitui os Backlogs da Sprint individuais de cada Time Scrum. Em vez disso, ele existe para complementar esses Backlogs, auxiliando na transparência e coordenação durante a Sprint. Cada Time Scrum mantém seu próprio Backlog da Sprint, além de contribuir para e interagir com o Backlog da Sprint do Nexus.

Gabarito: Errado

---

10. **(PROF. DIEGO / INÉDITA – 2024)** O Time de Integração do Nexus consiste exclusivamente de Membros do Time de Integração.

Comentários:

O Time de Integração do Nexus inclui não apenas os Membros do Time de Integração, mas também um Product Owner e um Scrum Master. Essa composição reflete a necessidade de um esforço coordenado e especializado para gerenciar as complexidades e desafios de integrar o trabalho de múltiplos Times Scrum, garantindo que os princípios e práticas do Scrum sejam adequadamente aplicados dentro do contexto do Nexus.

Gabarito: Errado

---

11. **(PROF. DIEGO / INÉDITA – 2024)** O Time de Integração do Nexus tem a função exclusiva de realizar trabalho técnico, sem envolver-se na coordenação ou supervisão dos Times Scrum.

Comentários:

O Time de Integração do Nexus possui um escopo de responsabilidades muito mais amplo do que apenas realizar trabalho técnico. Além de potencialmente realizar trabalho do Backlog do Produto, este time é crucial para coordenar, supervisionar e garantir a integração eficaz entre os Times Scrum, resolvendo restrições técnicas e não técnicas e facilitando a colaboração e comunicação entre os times para a entrega de um Incremento Integrado a cada Sprint.

Gabarito: Errado

---

12. **(PROF. DIEGO / INÉDITA – 2024)** O Product Owner dentro do Nexus não tem influência sobre o Backlog do Produto, visto que as decisões são tomadas coletivamente por todos os membros do Nexus.

Comentários:

O Product Owner, dentro do contexto do Nexus, mantém a responsabilidade singular de gerenciar o Backlog do Produto, assegurando que o máximo valor seja extraído do trabalho realizado pelos Times Scrum. Essa posição confere ao Product Owner a autoridade final sobre o conteúdo do Backlog do Produto, alinhando-se com a intenção de maximizar o valor do produto e do trabalho integrado entregue pelo Nexus.

Gabarito: Errado

---

13. **(PROF. DIEGO / INÉDITA – 2024)** Os membros do Time de Integração do Nexus podem simultaneamente fazer parte de um Time Scrum individual e priorizar suas responsabilidades no Time Scrum sobre o trabalho no Time de Integração do Nexus.

Comentários:

Quando membros do Time de Integração do Nexus também fazem parte de um Time Scrum individual, eles devem priorizar seu trabalho no Time de Integração do Nexus sobre suas responsabilidades no Time Scrum individual. Essa priorização ajuda a assegurar que questões afetando múltiplos times sejam resolvidas de maneira eficaz, facilitando a entrega de Incrementos Integrados de qualidade.

Gabarito: Errado

---

14. **(PROF. DIEGO / INÉDITA – 2024)** O Scrum Master no Time de Integração do Nexus tem como única função a de facilitar reuniões Scrum para os Times Scrum no Nexus.

Comentários:

O Scrum Master no Time de Integração do Nexus tem responsabilidades que vão além de simplesmente facilitar reuniões Scrum. Este papel envolve garantir que o framework Nexus seja compreendido e implementado corretamente, além de potencialmente servir como Scrum Master para um ou mais Times Scrum dentro do Nexus. Isso inclui promover o entendimento e a adoção de práticas que suportem a integração e colaboração efetivas entre os times.

Gabarito: Errado

---

15. **(PROF. DIEGO / INÉDITA – 2024)** Uma vez designados para o Time de Integração do Nexus, os membros permanecem nessa função indefinidamente para manter a consistência.

Comentários:

A composição do Time de Integração do Nexus pode mudar ao longo do tempo para refletir as necessidades do Nexus. Isso significa que os membros podem alternar entre o Time de Integração e os Times Scrum individuais, conforme necessário, para garantir que o Nexus permaneça adaptável e capaz de atender às demandas dinâmicas do projeto e da organização. Tal flexibilidade ajuda a manter o Nexus ágil e responsivo às mudanças.

Gabarito: Errado

---

16. **(PROF. DIEGO / INÉDITA – 2024)** O Refinamento do Backlog do Produto em um contexto Nexus tem como único propósito preparar os itens para serem trabalhados por um único Time Scrum.

Comentários:

O Refinamento do Backlog do Produto em um contexto Nexus serve para dois propósitos principais: ajudar os Times Scrum a preverem qual time irá entregar cada item do Backlog do Produto e identificar dependências entre esses times, além de preparar os itens para serem trabalhados por um único Time Scrum. Este processo visa garantir a transparência e minimizar dependências, tornando os itens do Backlog suficientemente independentes para serem trabalhados sem conflitos excessivos.

Gabarito: Errado

---

17. **(PROF. DIEGO / INÉDITA – 2024)** Durante o Planejamento da Sprint do Nexus, apenas os representantes apropriados de cada Time Scrum devem participar para minimizar problemas de comunicação.

Comentários:

Durante o Planejamento da Sprint do Nexus, todos os membros dos Times Scrum devem participar. Esta prática ajuda a minimizar os problemas de comunicação, já que envolve a validação e ajustes na ordenação do trabalho por representantes apropriados de cada Time Scrum, além de discutir a Meta da Sprint do Nexus e as dependências entre os times.

Gabarito: Errado

---

18. (PROF. DIEGO / INÉDITA – 2024) A Meta da Sprint do Nexus é definida de forma independente pelas Metas das Sprints dos Times Scrum individuais.

Comentários:

A Meta da Sprint do Nexus não é definida de forma independente pelas Metas das Sprints dos Times Scrum individuais; ela é a soma de todo o trabalho e Metas das Sprints dos Times Scrum dentro do Nexus. Isso significa que a Meta da Sprint do Nexus reflete um objetivo comum que será alcançado coletivamente pelos Times Scrum durante a Sprint.

Gabarito: Errado

---

19. (PROF. DIEGO / INÉDITA – 2024) A Reunião Diária do Nexus é um evento exclusivamente para os Scrum Masters dos Times de Desenvolvimento discutirem o progresso em direção à Meta da Sprint.

Comentários:

A Reunião Diária do Nexus é um evento para os representantes adequados dos Times de Desenvolvimento (não exclusivamente para os Scrum Masters) inspecionarem a atual situação do Incremento Integrado e identificarem questões de integração ou descobertas sobre as dependências ou impactos entre os times. O foco está no impacto de cada time no Incremento Integrado e na comunicação de novas dependências ou informações que precisam ser compartilhadas entre as equipes no Nexus.

Gabarito: Errado

---

20. (PROF. DIEGO / INÉDITA – 2024) A Retrospectiva da Sprint do Nexus é realizada antes da Revisão da Sprint do Nexus e tem como único propósito discutir as melhorias para os processos do Nexus.

Comentários:

A Retrospectiva da Sprint do Nexus é realizada após a Revisão da Sprint do Nexus e antes do próximo Planejamento da Sprint do Nexus, não antes da Revisão. Seu propósito vai além de discutir apenas as melhorias para os processos do Nexus; ela serve como uma oportunidade formal para o Nexus inspecionar e adaptar a si mesmo, criar um plano de melhorias que serão implementadas na próxima Sprint, e garantir melhoria contínua. A Retrospectiva consiste em três partes, envolvendo a identificação de questões que impactam além de um único time, a realização de Retrospectivas individuais pelos Times Scrum, e a reunião final para acordo sobre ações de melhoria.

Gabarito: Errado

---

21. (PROF. DIEGO / INÉDITA – 2024) No Nexus, o Backlog do Produto é gerenciado exclusivamente por um único Product Owner, responsável por seu conteúdo, disponibilidade e ordenação.

Comentários:

Há um único Backlog do Produto para todo o Nexus, e que o Product Owner é o responsável por este, incluindo seu conteúdo, disponibilidade e ordenação. Isso ressalta a importância de uma gestão centralizada do Backlog do Produto em um ambiente Nexus.

Gabarito: Correto

---

22. (PROF. DIEGO / INÉDITA – 2024) A Reunião Diária do Nexus não desempenha um papel na atualização do Backlog da Sprint do Nexus.

Comentários:

O Backlog da Sprint do Nexus é atualizado pelo menos uma vez ao dia, frequentemente como parte da Reunião Diária do Nexus. Isso indica que a Reunião Diária do Nexus é um momento crucial para a atualização do Backlog da Sprint do Nexus, destacando dependências e fluxo de trabalho.

Gabarito: Errado

---

23. (PROF. DIEGO / INÉDITA – 2024) Um Incremento Integrado no Nexus não precisa ser utilizável nem atender à definição de "Pronto" para ser considerado completo.

Comentários:

Um Incremento Integrado deve ser utilizável e potencialmente possível de ser entregue, significando que deve atender à definição de "Pronto". Isso sublinha a exigência de que todo incremento integrado seja de alta qualidade e pronto para entrega.

Gabarito: Errado

---

24. (PROF. DIEGO / INÉDITA – 2024) A definição de "Pronto" no Nexus pode variar entre os diferentes Times Scrum que compõem o Nexus.

Comentários:

O Time de Integração é responsável por estabelecer a definição de "Pronto" para o incremento integrado desenvolvido em cada sprint, e todos os Times Scrum devem aderir a essa definição. Isso implica que não pode haver variações na definição de "Pronto" entre os diferentes Times Scrum dentro de um Nexus.

Gabarito: Errado

---

25. (PROF. DIEGO / INÉDITA – 2024) No Nexus, os Times Scrum individuais podem aplicar uma definição de "Pronto" mais rigorosa do que a definida para o incremento integrado, se assim escolherem.

Comentários:

Times Scrum individuais aplicam uma definição de "Pronto" mais rigorosa dentro de seus próprios times, mas proíbe a aplicação de critérios menos rigorosos do que os acordados para o incremento integrado. Isso proporciona flexibilidade aos times para manter ou elevar os padrões de qualidade.

Gabarito: Correto

---

26. (PROF. DIEGO / INÉDITA – 2024) Implementar apenas partes do Nexus ainda resulta na aplicação efetiva do framework Nexus.

Comentários:

Apesar de ser possível implementar partes do Nexus, o resultado não é considerado Nexus. Isso destaca a importância da integridade do framework e a necessidade de adotar todos os seus componentes para uma implementação eficaz.

Gabarito: Errado

Lista de Questões

1. **(PROF. DIEGO / INÉDITA – 2024)** O Nexus é um framework que permite a colaboração de até nove Times Scrum em um único Backlog de Produto para construir um incremento integrado.

2. **(PROF. DIEGO / INÉDITA – 2024)** A palavra "nexus" não tem relação com a natureza do framework, sendo apenas um nome escolhido arbitrariamente.

3. **(PROF. DIEGO / INÉDITA – 2024)** O Nexus elimina completamente as dependências entre os times trabalhando em um incremento integrado.

4. **(PROF. DIEGO / INÉDITA – 2024)** No Nexus, não há necessidade de organizar e sequenciar o trabalho, uma vez que múltiplos times Scrum trabalham de forma independente.

5. **(PROF. DIEGO / INÉDITA – 2024)** A distribuição do conhecimento do domínio entre os Times Scrum é uma estratégia do Nexus para minimizar interrupções durante uma Sprint.

6. **(PROF. DIEGO / INÉDITA – 2024)** O Time de Integração do Nexus é responsável apenas por supervisionar a aplicação do Nexus.

7. **(PROF. DIEGO / INÉDITA – 2024)** No Nexus, cada Time Scrum trabalha com seu próprio e exclusivo Backlog do Produto.

8. **(PROF. DIEGO / INÉDITA – 2024)** A Revisão da Sprint no Nexus é realizada de forma idêntica ao Scrum tradicional, sem modificações.

9. **(PROF. DIEGO / INÉDITA – 2024)** O Backlog da Sprint do Nexus é um artefato que substitui os Backlogs da Sprint individuais de cada Time Scrum.

10. **(PROF. DIEGO / INÉDITA – 2024)** O Time de Integração do Nexus consiste exclusivamente de Membros do Time de Integração.

11. **(PROF. DIEGO / INÉDITA – 2024)** O Time de Integração do Nexus tem a função exclusiva de realizar trabalho técnico, sem envolver-se na coordenação ou supervisão dos Times Scrum.

12. **(PROF. DIEGO / INÉDITA – 2024)** O Product Owner dentro do Nexus não tem influência sobre o Backlog do Produto, visto que as decisões são tomadas coletivamente por todos os membros do Nexus.

13. **(PROF. DIEGO / INÉDITA – 2024)** Os membros do Time de Integração do Nexus podem simultaneamente fazer parte de um Time Scrum individual e priorizar suas responsabilidades no Time Scrum sobre o trabalho no Time de Integração do Nexus.

14. **(PROF. DIEGO / INÉDITA – 2024)** O Scrum Master no Time de Integração do Nexus tem como única função a de facilitar reuniões Scrum para os Times Scrum no Nexus.

15. **(PROF. DIEGO / INÉDITA – 2024)** Uma vez designados para o Time de Integração do Nexus, os membros permanecem nessa função indefinidamente para manter a consistência.

16. **(PROF. DIEGO / INÉDITA – 2024)** O Refinamento do Backlog do Produto em um contexto Nexus tem como único propósito preparar os itens para serem trabalhados por um único Time Scrum.

17. **(PROF. DIEGO / INÉDITA – 2024)** Durante o Planejamento da Sprint do Nexus, apenas os representantes apropriados de cada Time Scrum devem participar para minimizar problemas de comunicação.

18. **(PROF. DIEGO / INÉDITA – 2024)** A Meta da Sprint do Nexus é definida de forma independente pelas Metas das Sprints dos Times Scrum individuais.

19. **(PROF. DIEGO / INÉDITA – 2024)** A Reunião Diária do Nexus é um evento exclusivamente para os Scrum Masters dos Times de Desenvolvimento discutirem o progresso em direção à Meta da Sprint.

20. **(PROF. DIEGO / INÉDITA – 2024)** A Retrospectiva da Sprint do Nexus é realizada antes da Revisão da Sprint do Nexus e tem como único propósito discutir as melhorias para os processos do Nexus.

21. **(PROF. DIEGO / INÉDITA – 2024)** No Nexus, o Backlog do Produto é gerenciado exclusivamente por um único Product Owner, responsável por seu conteúdo, disponibilidade e ordenação.

22. **(PROF. DIEGO / INÉDITA – 2024)** A Reunião Diária do Nexus não desempenha um papel na atualização do Backlog da Sprint do Nexus.

23. **(PROF. DIEGO / INÉDITA – 2024)** Um Incremento Integrado no Nexus não precisa ser utilizável nem atender à definição de "Pronto" para ser considerado completo.

24. **(PROF. DIEGO / INÉDITA – 2024)** A definição de "Pronto" no Nexus pode variar entre os diferentes Times Scrum que compõem o Nexus.

25. **(PROF. DIEGO / INÉDITA – 2024)** No Nexus, os Times Scrum individuais podem aplicar uma definição de "Pronto" mais rigorosa do que a definida para o incremento integrado, se assim escolherem.

26.(PROF. DIEGO / INÉDITA – 2024) Implementar apenas partes do Nexus ainda resulta na aplicação efetiva do framework Nexus.

GABARITO

1. CORRETO
2. ERRADO
3. ERRADO
4. ERRADO
5. CORRETO
6. ERRADO
7. ERRADO
8. ERRADO
9. ERRADO
10. ERRADO
11. ERRADO
12. ERRADO
13. ERRADO
14. ERRADO
15. ERRADO
16. ERRADO
17. ERRADO
18. ERRADO
19. ERRADO
20. ERRADO
21. CORRETO
22. ERRADO
23. ERRADO
24. ERRADO
25. CORRETO
26. ERRADO

SAFe

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

SAFe é um dos frameworks de Ágil em Escala mais amplamente adotadas, conhecida por sua abrangência e estrutura detalhada. Ela fornece um guia para escalar práticas ágeis em toda a empresa, abrangendo equipe, programa, e níveis de portfólio. SAFe inclui princípios para liderança Lean-Agile, práticas para desenvolvimento de software Lean, e diretrizes para implementar Ágil em larga escala. Ela é dividida em quatro níveis: Equipe, Programa, Valor de Solução e Portfolio.

BENEFÍCIOS	DESCRIÇÃO
TEMPO DE MERCADO MAIS RÁPIDO (TIME-TO-MARKET)	Um dos benefícios do Ágil Escalado com SAFe é o aumento do time-to-market. Ao alinhar equipes multifuncionais de equipes ágeis em torno do valor, as empresas líderes podem atender às necessidades dos clientes mais rapidamente. Aproveitar o poder do Scaled Agile Framework os ajuda a tomar decisões mais rápidas, se comunicar de forma mais eficaz, simplificar as operações e manter o foco no cliente.
MELHORIAS NA QUALIDADE	A qualidade é um dos principais valores da SAFe. Ela destaca a importância de integrar a qualidade em todas as etapas do ciclo de desenvolvimento. Dessa forma, o Ágil Escalado com o SAFe beneficia as organizações, mudando a qualidade para a responsabilidade de todos.
AUMENTO DE PRODUTIVIDADE	O SAFe fornece melhorias mensuráveis na produtividade, capacitando equipes de alto desempenho e equipes de equipes para eliminar o trabalho desnecessário, identificar e remover atrasos, melhorar continuamente e garantir que estejam construindo as coisas certas.
MELHOR ENGAJAMENTO DOS FUNCIONÁRIOS	Melhores formas de trabalhar rendem funcionários mais felizes e engajados. Um benefício do Ágil Escalado é ajudar os trabalhadores a alcançarem autonomia, domínio e propósito, que são os fatores-chave para desbloquear a motivação intrínseca. As organizações que praticam o SAFe têm as ferramentas para minimizar o burnout e aumentar a satisfação dos funcionários.

Temos também quatro valores fundamentais que orientam organizações na implementação de práticas ágeis em escala. Vejamos:

VALORES	DESCRIÇÃO
ALINHAMENTO	O alinhamento é crucial para o SAFe. Isso envolve comunicar claramente a visão, missão e estratégia da organização, de forma que todos possam compreendê-las. Também é importante conectar a estratégia à execução, garantindo que todos os esforços estejam alinhados aos objetivos de negócio. Utilizar uma linguagem comum e verificar constantemente o entendimento asseguram que todos estejam na mesma página. Compreender o cliente é fundamental para atender às suas necessidades e expectativas.
TRANSPARÊNCIA	A transparência cria um ambiente baseado na confiança. Isso significa comunicar-se de forma direta, aberta e honesta, sem ocultar informações relevantes. Transformar erros em

	momentos de aprendizado promove uma cultura de melhoria contínua. Visualizar o trabalho, por meio de quadros ou outras ferramentas, facilita o acompanhamento e a compreensão do progresso. Além disso, fornecer acesso fácil às informações necessárias para realizar o trabalho ajuda a evitar atrasos e facilita a colaboração
RESPEITO PELAS PESSOAS	No SAFe, valorizar o que significa ser humano é fundamental. Isso envolve valorizar a diversidade de pessoas e opiniões, reconhecendo que cada indivíduo traz perspectivas valiosas para a equipe. Desenvolver as pessoas por meio de orientação e mentoria permite que elas cresçam e se aprimorem profissionalmente. Abraçar a ideia de que "seu cliente é qualquer pessoa que consome o seu trabalho" enfatiza a importância de satisfazer as necessidades de todos os envolvidos. Construir parcerias de longo prazo baseadas em benefícios mútuos fortalece a colaboração e o sucesso compartilhado.
MELHORIA INCESSANTE	A busca pela melhoria contínua é um pilar do SAFe. Isso implica em criar uma sensação constante de urgência para impulsionar a evolução e a inovação. Cultivar uma cultura de resolução de problemas encoraja as equipes a identificar e solucionar obstáculos de forma proativa. Refletir e adaptar-se frequentemente permite ajustes e otimizações contínuas. No SAFe, as decisões são guiadas por fatos e dados, evitando a tomada de decisões baseadas em suposições. Fornecer tempo e espaço para a inovação estimula a criatividade e o desenvolvimento de soluções inovadoras.

Por fim, alguns princípios formam a espinha dorsal do SAFe, fornecendo uma base sólida para escalar práticas ágeis e lean dentro das organizações:

PRINCÍPIOS	DESCRIÇÃO
ADOTAR UMA VISÃO ECONÔMICA	Prioriza decisões que otimizam o valor do sistema como um todo, levando em consideração o custo de atraso, ciclo de vida do desenvolvimento, e uma compreensão de economia para tomada de decisão.
APLICAR PENSAMENTO SISTÊMICO	Reconhece que o sucesso de qualquer componente do sistema depende de sua relação com outros componentes e com o sistema como um todo, enfatizando a importância de entender os processos, as limitações e as interações dentro do ambiente de trabalho.
ASSUMIR A VARIABILIDADE; E PRESERVAR OPÇÕES	Incorpora a inovação e o desenvolvimento exploratório no início do processo de desenvolvimento para explorar soluções alternativas, permitindo que as decisões sejam tomadas com base em informações mais completas e reduzindo riscos.
CONSTRUIR DE FORMA INCREMENTAL COM CICLOS DE FEEDBACK RÁPIDOS E INTEGRADOS	Encoraja o desenvolvimento e a entrega de pequenos incrementos de valor, permitindo ciclos de feedback rápidos que informam decisões futuras e melhorias contínuas.
BASEAR MARCOS EM AVALIAÇÃO OBJETIVA DE SISTEMAS EM FUNCIONAMENTO	Defende a avaliação do progresso com base em sistemas tangíveis e funcionais, não apenas em documentações ou promessas, garantindo que os marcos reflitam o verdadeiro estado do projeto.
FAZER O VALOR FLUIR SEM INTERRUPÇÕES	Promove práticas que ajudam a identificar e eliminar gargalos, melhorando o fluxo de trabalho através do sistema de produção e aumentando a eficiência.

APLICAR CADÊNCIA, SINCRONIZAR COM O PLANEJAMENTO ENTRE DOMÍNIOS	Defende a importância de uma cadência regular e previsível para o trabalho, junto com a sincronização de planejamentos entre diferentes domínios, para garantir alinhamento e colaboração eficazes.
DESBLOQUEAR A MOTIVAÇÃO INTRÍNSECA DOS TRABALHADORES DO CONHECIMENTO	Reconhece a importância de criar um ambiente de trabalho que cultive a motivação intrínseca, promovendo autonomia, domínio e propósito entre os membros da equipe.
DESCENTRALIZAR A TOMADA DE DECISÕES	Encoraja a tomada de decisão no nível mais baixo possível para agilizar o processo e aproveitar o conhecimento e a experiência dos trabalhadores do conhecimento, ao mesmo tempo em que garante alinhamento e coesão em toda a organização.
ORGANIZAR EM TORNO DO VALOR	Exigir que as empresas se organizem em torno do valor para entregar mais rapidamente. E quando as demandas do mercado e do cliente mudam, a empresa deve se reorganizar rápida e perfeitamente em torno desse novo fluxo de valor.

QUESTÕES COMENTADAS

1. **(CESGRANRIO / TRANSPETRO – 2023)** A abordagem SAFe (Scaled Agile Framework) pode ser fundamental para garantir o sucesso de um projeto. Nessa abordagem, o Program Increment (PI) refere-se a um(a):

   a) conjunto de rituais diários que as equipes Agile realizam para sincronizar suas atividades.
   b) evento periódico que fornece uma oportunidade para a revisão e o ajuste do backlog do produto.
   c) período fixo de tempo durante o qual um conjunto de equipes entrega ao cliente um valor.
   d) plano detalhado de entregas que documenta todas as etapas de um projeto em escala.
   e) prática de planejamento estratégico para alinhar os projetos do portfólio com a estratégia de negócios.

Comentários:

(a) Errado. Rituais diários para sincronizar atividades são típicos de reuniões diárias (daily stand-ups), não do Program Increment;

(b) Errado. A revisão e ajuste do backlog do produto é geralmente feita durante as reuniões de revisão e planejamento, não especificamente no contexto de um Program Increment;

(c) Correto. O Program Increment (PI) é um período fixo de tempo, geralmente 8-12 semanas, durante o qual um conjunto de equipes trabalha para entregar um incremento de valor ao cliente;

(d) Errado. Um plano detalhado de entregas que documenta todas as etapas do projeto em escala se refere mais a um roadmap do que a um PI;

(e) Errado. O alinhamento estratégico de projetos do portfólio com a estratégia de negócios é mais uma função de gestão de portfólio, não especificamente do PI.

Gabarito: Letra C

---

2. **(PROF. DIEGO / INÉDITA – 2024)** SAFe é um framework que se aplica apenas ao nível de equipe dentro de uma organização.

Comentários:

SAFe é conhecido por sua capacidade de escalar práticas ágeis em toda a empresa, abrangendo equipe, programa e níveis de portfólio. Ele é projetado para alinhar equipes multifuncionais em torno de valor em diversos níveis da organização.

Gabarito: Errado

---

3. **(PROF. DIEGO / INÉDITA – 2024)** Um dos benefícios do SAFe é a melhoria do time-to-market.

Comentários:

SAFe ajuda as organizações a atender às necessidades dos clientes mais rapidamente, alavancando a agilidade em escala para tomar decisões mais rápidas e manter o foco no cliente, o que resulta em um time-to-market mais rápido.

Gabarito: Correto

---

4. **(PROF. DIEGO / INÉDITA – 2024)** SAFe desencoraja a melhoria contínua e a qualidade no processo de desenvolvimento.

Comentários:

A qualidade é um dos valores principais da SAFe, que enfatiza a integração da qualidade em todas as etapas do ciclo de desenvolvimento, tornando a qualidade uma responsabilidade compartilhada por todos.

Gabarito: Errado

---

5. **(PROF. DIEGO / INÉDITA – 2024)** SAFe reduz a produtividade das equipes ao adicionar mais camadas de gestão.

Comentários:

SAFe na verdade fornece melhorias mensuráveis na produtividade ao capacitar equipes de alto desempenho para eliminar trabalho desnecessário, identificar e remover atrasos, e melhorar continuamente.

Gabarito: Errado

---

6. **(PROF. DIEGO / INÉDITA – 2024)** A implementação do SAFe pode resultar em burnout e diminuição da satisfação dos funcionários.

Comentários:

Um dos benefícios do SAFe é o melhor engajamento dos funcionários, proporcionando formas de trabalho que aumentam a satisfação dos funcionários e minimizam o burnout, ajudando-os a alcançar autonomia, domínio e propósito.

Gabarito: Errado

---

7. **(PROF. DIEGO / INÉDITA – 2024)** SAFe é dividido em apenas dois níveis: Equipe e Programa.

Comentários:

SAFe é dividido em quatro níveis: Equipe, Programa, Valor de Solução e Portfólio, permitindo uma aplicação abrangente da agilidade em escala em toda a organização.

Gabarito: Errado

---

8. **(PROF. DIEGO / INÉDITA – 2024)** SAFe enfatiza a tomada de decisão centralizada para manter a consistência em toda a organização.

Comentários:

Embora SAFe promova alinhamento e consistência, ele também valoriza a descentralização da tomada de decisão, permitindo que decisões sejam tomadas no nível mais apropriado para agilizar processos e responder rapidamente a mudanças.

Gabarito: Errado

---

9. **(PROF. DIEGO / INÉDITA – 2024)** Um dos princípios do SAFe é mudar a responsabilidade pela qualidade apenas para os times de QA (Quality Assurance).

Comentários:

SAFe promove a ideia de que a qualidade é responsabilidade de todos os membros da equipe, integrando práticas de qualidade em todas as etapas do ciclo de desenvolvimento, ao invés de confinar a responsabilidade pela qualidade a um único departamento ou equipe.

Gabarito: Errado

---

10. **(PROF. DIEGO / INÉDITA – 2024)** SAFe não oferece nenhuma estrutura para gerenciar o portfólio de projetos de uma organização.

Comentários:

SAFe inclui explicitamente um nível de Portfólio, que fornece uma estrutura para gerenciamento de portfólio ágil, garantindo que os investimentos estejam alinhados com a estratégia de negócios e os objetivos organizacionais.

Gabarito: Errado

11. (PROF. DIEGO / INÉDITA – 2024) SAFe promove uma abordagem de "tamanho único" que não permite personalização.

Comentários:

Embora SAFe forneça uma estrutura detalhada, ele também reconhece a necessidade de adaptação e permite que as organizações ajustem práticas ágeis para atender às suas necessidades específicas, incentivando a flexibilidade dentro de sua estrutura abrangente.

Gabarito: Errado

---

12. (PROF. DIEGO / INÉDITA – 2024) No SAFe, a transparência é valorizada apenas em termos de comunicação direta, não sendo necessário visualizar o trabalho ou fornecer fácil acesso às informações.

Comentários:

A transparência no SAFe envolve não apenas a comunicação direta, aberta e honesta, mas também a visualização do trabalho por meio de quadros ou outras ferramentas e o fornecimento de fácil acesso às informações necessárias para realizar o trabalho. Isso ajuda a evitar atrasos e facilita a colaboração.

Gabarito: Errado

---

13. (PROF. DIEGO / INÉDITA – 2024) No SAFe, entender o cliente é considerado fundamental para atender às suas necessidades e expectativas.

Comentários:

Compreender o cliente é fundamental dentro do SAFe para atender às suas necessidades e expectativas, conectando isso ao valor de alinhamento, onde conectar a estratégia à execução e compreender o cliente são componentes chave.

Gabarito: Correto

---

14. (PROF. DIEGO / INÉDITA – 2024) A melhoria incessante no SAFe se foca exclusivamente em cultivar uma cultura de resolução de problemas sem a necessidade de reflexão ou adaptação frequente.

Comentários:

A melhoria incessante no SAFe não se limita apenas a cultivar uma cultura de resolução de problemas; também inclui a importância de refletir e adaptar-se frequentemente, o que permite ajustes e otimizações contínuas, além de ser guiada por fatos e dados.

Gabarito: Errado

---

15. (PROF. DIEGO / INÉDITA – 2024) No SAFe, o respeito pelas pessoas é demonstrado através da valorização da diversidade e do desenvolvimento de pessoas por meio de orientação e mentoria.

Comentários:

O respeito pelas pessoas é um valor fundamental no SAFe, o qual envolve valorizar a diversidade de pessoas e opiniões e desenvolver pessoas através de orientação e mentoria, enfatizando a importância de cada indivíduo e suas perspectivas valiosas para a equipe.

Gabarito: Correto

---

16. (PROF. DIEGO / INÉDITA – 2024) Segundo o SAFe, o alinhamento organizacional é alcançado principalmente pela competição interna e a segregação de informações entre os times.

Comentários:

O alinhamento é crucial para o SAFe e é alcançado comunicando claramente a visão, missão e estratégia da organização, garantindo que todos os esforços estejam alinhados aos objetivos de negócio, utilizando uma linguagem comum e verificando constantemente o entendimento, o que contradiz a noção de competição interna e segregação de informações.

Gabarito: Errado

---

17. (PROF. DIEGO / INÉDITA – 2024) A tomada de decisão no SAFe é guiada por fatos e dados, evitando suposições.

Comentários:

Dentro do valor de melhoria incessante no SAFe, as decisões são guiadas por fatos e dados, o que enfatiza a importância de evitar a tomada de decisões baseadas em suposições, promovendo uma abordagem mais objetiva e fundamentada.

Gabarito: Correto

---

18. (PROF. DIEGO / INÉDITA – 2024) O princípio de adotar uma visão econômica no SAFe ignora o custo de atraso nas decisões de projeto.

Comentários:

O princípio de adotar uma visão econômica no SAFe enfatiza a importância de levar em consideração o custo de atraso, o ciclo de vida do desenvolvimento, e uma compreensão de economia na tomada de decisão. Portanto, este princípio não ignora, mas prioriza decisões que otimizam o valor do sistema como um todo, incluindo o custo de atraso.

Gabarito: Errado

---

19. **(PROF. DIEGO / INÉDITA – 2024)** No SAFe, aplicar o pensamento sistêmico significa que o sucesso de um componente é independente de sua relação com outros componentes e com o sistema como um todo.

Comentários:

Aplicar o pensamento sistêmico, de acordo com o SAFe, reconhece que o sucesso de qualquer componente do sistema depende de sua relação com outros componentes e com o sistema como um todo. Este princípio enfatiza a importância de entender os processos, as limitações, e as interações dentro do ambiente de trabalho.

Gabarito: Errado

---

20. **(PROF. DIEGO / INÉDITA – 2024)** Assumir a variabilidade e preservar opções é um princípio do SAFe que incentiva a exploração de soluções alternativas no início do processo de desenvolvimento.

Comentários:

Este princípio incorpora a inovação e o desenvolvimento exploratório no início do processo de desenvolvimento para explorar soluções alternativas. Isso permite que as decisões sejam tomadas com base em informações mais completas e ajuda a reduzir riscos.

Gabarito: Correto

---

21. **(PROF. DIEGO / INÉDITA – 2024)** O SAFe promove a construção de forma incremental sem a necessidade de ciclos de feedback rápidos e integrados.

Comentários:

O princípio de construir de forma incremental com ciclos de feedback rápidos e integrados é fundamental no SAFe. Ele encoraja o desenvolvimento e a entrega de pequenos incrementos de valor, permitindo ciclos de feedback rápidos que informam decisões futuras e promovem melhorias contínuas.

Gabarito: Errado

---

22. **(PROF. DIEGO / INÉDITA – 2024)** Basear marcos em avaliação objetiva de sistemas em funcionamento é um princípio que defende a avaliação do progresso com base em documentações e promessas ao invés de sistemas tangíveis e funcionais.

Comentários:

Este princípio defende exatamente o oposto, enfatizando a avaliação do progresso com base em sistemas tangíveis e funcionais, não apenas em documentações ou promessas. Ele garante que os marcos reflitam o verdadeiro estado do projeto.

Gabarito: Errado

---

23. **(PROF. DIEGO / INÉDITA – 2024)** Descentralizar a tomada de decisões no SAFe encoraja a tomada de decisão no nível mais baixo possível para agilizar o processo e aproveitar o conhecimento local.

Comentários:

Este princípio encoraja a tomada de decisão no nível mais baixo possível para agilizar o processo e aproveitar o conhecimento e a experiência dos trabalhadores do conhecimento. Ele visa garantir alinhamento e coesão em toda a organização, aproveitando o conhecimento local.

Gabarito: Correto

Lista de Questões

1. **(CESGRANRIO / TRANSPETRO – 2023)** A abordagem SAFe (Scaled Agile Framework) pode ser fundamental para garantir o sucesso de um projeto. Nessa abordagem, o Program Increment (PI) refere-se a um(a):

   a) conjunto de rituais diários que as equipes Agile realizam para sincronizar suas atividades.
   b) evento periódico que fornece uma oportunidade para a revisão e o ajuste do backlog do produto.
   c) período fixo de tempo durante o qual um conjunto de equipes entrega ao cliente um valor.
   d) plano detalhado de entregas que documenta todas as etapas de um projeto em escala.
   e) prática de planejamento estratégico para alinhar os projetos do portfólio com a estratégia de negócios.

2. **(PROF. DIEGO / INÉDITA – 2024)** SAFe é um framework que se aplica apenas ao nível de equipe dentro de uma organização.

3. **(PROF. DIEGO / INÉDITA – 2024)** Um dos benefícios do SAFe é a melhoria do time-to-market.

4. **(PROF. DIEGO / INÉDITA – 2024)** SAFe desencoraja a melhoria contínua e a qualidade no processo de desenvolvimento.

5. **(PROF. DIEGO / INÉDITA – 2024)** SAFe reduz a produtividade das equipes ao adicionar mais camadas de gestão.

6. **(PROF. DIEGO / INÉDITA – 2024)** A implementação do SAFe pode resultar em burnout e diminuição da satisfação dos funcionários.

7. **(PROF. DIEGO / INÉDITA – 2024)** SAFe é dividido em apenas dois níveis: Equipe e Programa.

8. **(PROF. DIEGO / INÉDITA – 2024)** SAFe enfatiza a tomada de decisão centralizada para manter a consistência em toda a organização.

9. **(PROF. DIEGO / INÉDITA – 2024)** Um dos princípios do SAFe é mudar a responsabilidade pela qualidade apenas para os times de QA (Quality Assurance).

10. **(PROF. DIEGO / INÉDITA – 2024)** SAFe não oferece nenhuma estrutura para gerenciar o portfólio de projetos de uma organização.

11. **(PROF. DIEGO / INÉDITA – 2024)** SAFe promove uma abordagem de "tamanho único" que não permite personalização.

12. **(PROF. DIEGO / INÉDITA – 2024)** No SAFe, a transparência é valorizada apenas em termos de comunicação direta, não sendo necessário visualizar o trabalho ou fornecer fácil acesso às informações.

13. **(PROF. DIEGO / INÉDITA – 2024)** No SAFe, entender o cliente é considerado fundamental para atender às suas necessidades e expectativas.

14. **(PROF. DIEGO / INÉDITA – 2024)** A melhoria incessante no SAFe se foca exclusivamente em cultivar uma cultura de resolução de problemas sem a necessidade de reflexão ou adaptação frequente.

15. **(PROF. DIEGO / INÉDITA – 2024)** No SAFe, o respeito pelas pessoas é demonstrado através da valorização da diversidade e do desenvolvimento de pessoas por meio de orientação e mentoria.

16. **(PROF. DIEGO / INÉDITA – 2024)** Segundo o SAFe, o alinhamento organizacional é alcançado principalmente pela competição interna e a segregação de informações entre os times.

17. **(PROF. DIEGO / INÉDITA – 2024)** A tomada de decisão no SAFe é guiada por fatos e dados, evitando suposições.

18. **(PROF. DIEGO / INÉDITA – 2024)** O princípio de adotar uma visão econômica no SAFe ignora o custo de atraso nas decisões de projeto.

19. **(PROF. DIEGO / INÉDITA – 2024)** No SAFe, aplicar o pensamento sistêmico significa que o sucesso de um componente é independente de sua relação com outros componentes e com o sistema como um todo.

20. **(PROF. DIEGO / INÉDITA – 2024)** Assumir a variabilidade e preservar opções é um princípio do SAFe que incentiva a exploração de soluções alternativas no início do processo de desenvolvimento.

21. **(PROF. DIEGO / INÉDITA – 2024)** O SAFe promove a construção de forma incremental sem a necessidade de ciclos de feedback rápidos e integrados.

22. **(PROF. DIEGO / INÉDITA – 2024)** Basear marcos em avaliação objetiva de sistemas em funcionamento é um princípio que defende a avaliação do progresso com base em documentações e promessas ao invés de sistemas tangíveis e funcionais.

23. **(PROF. DIEGO / INÉDITA – 2024)** Descentralizar a tomada de decisões no SAFe encoraja a tomada de decisão no nível mais baixo possível para agilizar o processo e aproveitar o conhecimento local.

GABARITO

1. LETRA C
2. ERRADO
3. CORRETO
4. ERRADO
5. ERRADO
6. ERRADO
7. ERRADO
8. ERRADO
9. ERRADO
10. ERRADO
11. ERRADO
12. ERRADO
13. CORRETO
14. ERRADO
15. CORRETO
16. ERRADO
17. CORRETO
18. ERRADO
19. ERRADO
20. CORRETO
21. ERRADO
22. CORRETO
23. CORRETO

MANAGEMENT 3.0

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Management 3.0 é uma abordagem de gestão e liderança que se concentra em promover ambientes de trabalho ágeis, adaptativos e inovadores. Essa abordagem busca proporcionar uma resposta eficaz aos desafios enfrentados pelas organizações em um mundo cada vez mais complexo e dinâmico. Ele foi introduzido por Jurgen Appelo, um autor e palestrante internacionalmente reconhecido em temas de liderança, gestão e agilidade.

Ele propôs essa abordagem como uma resposta à necessidade de adaptar os modelos tradicionais de gestão para lidar com a crescente complexidade e mudança nos negócios. **Management 3.0 combina princípios da agilidade, gestão de mudanças, psicologia organizacional e outras disciplinas para criar um conjunto abrangente de práticas de liderança e gestão.** Vejamos alguns princípios que norteiam essa abordagem de gerenciamento:

PRINCÍPIOS	DESCRIÇÃO
FOCO NAS PESSOAS	Management 3.0 reconhece a importância de criar ambientes de trabalho onde as pessoas se sintam motivadas, engajadas e capacitadas. Isso envolve promover uma cultura de confiança, autonomia, colaboração e aprendizado contínuo.
GESTÃO DE MUDANÇAS	A abordagem reconhece a inevitabilidade da mudança e defende uma abordagem adaptativa para gerenciá-la. Os líderes são incentivados a promover uma cultura organizacional que abrace a mudança e esteja preparada para se adaptar rapidamente a novas circunstâncias.
DELEGAÇÃO DE AUTORIDADE	Em vez de concentrar o poder de tomada de decisões em uma hierarquia rígida, Management 3.0 promove a delegação de autoridade para as equipes. Isso permite que as equipes tenham mais autonomia e responsabilidade para tomar decisões que afetam seu trabalho.
FEEDBACK E COMUNICAÇÃO	A abordagem enfatiza a importância do feedback regular e da comunicação aberta e transparente entre líderes, equipes e outras partes interessadas. Isso ajuda a garantir que todos tenham uma compreensão clara dos objetivos, expectativas e desafios da organização.
LIDERANÇA SERVIDORA	Em vez de adotar uma abordagem de comando e controle, Management 3.0 promove a liderança servidora, na qual os líderes estão focados em apoiar e capacitar suas equipes para alcançar seus objetivos. Isso envolve remover obstáculos, fornecer recursos e orientação, e cultivar um ambiente de confiança e respeito mútuo.

Papéis e Responsabilidades

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Em equipes ágeis, a abordagem de papéis e responsabilidades difere significativamente dos modelos tradicionais de gestão hierárquica. Em geral, em equipes ágeis, os papéis e responsabilidades são mais flexíveis, adaptativos e compartilhados do que nos modelos tradicionais de gestão hierárquica. Isso permite uma maior agilidade, autonomia e eficácia na execução de projetos e na entrega de valor ao cliente.

PAPEIS E RESPONSABILIDADES	DESCRIÇÃO
EQUIPES MULTIFUNCIONAIS E AUTO-ORGANIZADAS	Em equipes ágeis, os membros são frequentemente organizados em equipes multifuncionais, onde cada membro contribui com habilidades diversas para alcançar os objetivos do projeto. Essas equipes são auto-organizadas, o que significa que têm autoridade para tomar decisões sobre como realizar o trabalho.
PAPEIS FLEXÍVEIS E ADAPTATIVOS	Os papéis em equipes ágeis tendem a ser mais flexíveis e adaptativos em comparação com os modelos tradicionais. Em vez de ter papéis rigidamente definidos, os membros da equipe podem assumir diferentes papéis conforme necessário para atender às demandas do projeto e do time.
PROPRIEDADE COLETIVA E RESPONSABILIDADE COMPARTILHADA	Em equipes ágeis, há um forte senso de propriedade coletiva e responsabilidade compartilhada pelo sucesso do projeto. Em vez de atribuir responsabilidades a indivíduos específicos, a equipe como um todo é responsável pelo cumprimento dos objetivos e pela entrega de valor ao cliente.
DISTRIBUIÇÃO DE AUTORIDADE E TOMADA DE DECISÕES	A autoridade e a tomada de decisões são distribuídas de forma mais ampla em equipes ágeis. Em vez de centralizar a autoridade em uma única figura de liderança, a autoridade é compartilhada entre os membros da equipe. As equipes são encorajadas a tomar decisões de forma colaborativa e descentralizada, aproveitando o conhecimento e a experiência de todos os membros.
LIDERANÇA SERVIDORA	Em equipes ágeis, os líderes adotam uma abordagem de liderança servidora, onde estão focados em apoiar e capacitar os membros da equipe para alcançar seus objetivos. Isso significa remover obstáculos, fornecer recursos e orientação, e promover um ambiente de confiança e colaboração.
FEEDBACK CONTÍNUO E APRENDIZADO	Em equipes ágeis, o feedback contínuo é fundamental para melhorar o desempenho e o desenvolvimento dos membros da equipe. Os líderes e os membros da equipe fornecem feedback uns aos outros regularmente, ajudando a identificar áreas de melhoria e oportunidades de aprendizado.

Motivação e Engajamento

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

A motivação e o engajamento das equipes são aspectos essenciais para criar um ambiente de trabalho produtivo, colaborativo e satisfatório. Ao aplicar essas técnicas e princípios, os líderes e gestores podem criar ambientes de trabalho mais motivadores e engajadores, onde as equipes se sintam inspiradas a alcançar seu pleno potencial e contribuir de forma significativa para o sucesso da organização.

TÉCNICAS	DESCRIÇÃO
PROPÓSITO E VISÃO COMPARTILHADA	Estabelecer um propósito claro e uma visão compartilhada para o trabalho ajuda a alinhar os membros da equipe em direção a objetivos comuns. Compreender o propósito e o impacto do trabalho pode aumentar significativamente a motivação e o engajamento.
AUTONOMIA E EMPODERAMENTO	Dar às equipes autonomia para tomar decisões sobre como realizar o trabalho e resolver desafios pode aumentar o senso de responsabilidade e comprometimento. Permitir que os membros da equipe se sintam capacitados para influenciar e contribuir para o processo de tomada de decisões promove um maior engajamento.
HABILIDADES E OPORTUNIDADES DE CRESCIMENTO	Investir no desenvolvimento profissional dos membros da equipe e oferecer oportunidades de aprendizado e crescimento pode aumentar a motivação e o engajamento. Isso pode incluir treinamentos, workshops, mentorias e atribuição de projetos desafiadores.
FEEDBACK CONTÍNUO E RECONHECIMENTO	Fornecer feedback regular e construtivo sobre o desempenho dos membros da equipe é fundamental para promover o engajamento e a melhoria contínua. Além disso, reconhecer e valorizar as contribuições individuais e coletivas reforça um ambiente de trabalho positivo e motivador.
CULTURA DE CONFIANÇA E COLABORAÇÃO	Promover uma cultura organizacional baseada na confiança, respeito mútuo e colaboração incentiva os membros da equipe a se sentirem mais seguros para expressar suas ideias, assumir riscos e contribuir ativamente para os objetivos da equipe.
TEORIAS DE MOTIVAÇÃO	Esta teoria postula que as pessoas são motivadas quando têm oportunidades de satisfazer três necessidades psicológicas fundamentais: autonomia, competência e relacionamento social. Logo, as técnicas de motivação devem se concentrar em fornecer aos membros da equipe autonomia para controlar suas atividades, oportunidades para desenvolver e demonstrar competências, e interações sociais positivas e significativas.

Feedback e Comunicação

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Dentro do contexto do Management 3.0, o feedback e a comunicação eficaz desempenham um papel fundamental na gestão ágil, promovendo a transparência, a colaboração e a melhoria contínua. Ao implementar essas estratégias de feedback e comunicação eficaz no contexto do Management 3.0, os líderes e gestores podem fortalecer os laços dentro da equipe, melhorar o desempenho e promover uma cultura de aprendizagem e melhoria contínua.

Importância do Feedback e da Comunicação Eficaz

- O feedback é essencial para promover a aprendizagem e o desenvolvimento contínuo das equipes. Ele fornece informações valiosas sobre o desempenho, as práticas de trabalho e as áreas de melhoria, permitindo que as equipes façam ajustes e aprimoramentos necessários;

- A comunicação eficaz é fundamental para alinhar expectativas, compartilhar informações relevantes e promover um ambiente de trabalho colaborativo. Ela ajuda a garantir que membros da equipe estejam na mesma página e compreendam objetivos, desafios e planos de ação.

Estratégias para Fornecer Feedback Construtivo

- **Seja específico:** ao fornecer feedback, seja específico sobre os comportamentos observados ou os resultados alcançados.
- **Foque no comportamento, não na pessoa:** concentre-se no comportamento ou nas ações observadas, em vez de fazer críticas pessoais.
- **Seja equilibrado:** forneça feedback positivo e reconhecimento pelo bom trabalho, bem como sugestões de melhoria.
- **Seja oportuno:** forneça feedback regularmente e no momento apropriado, para que seja mais relevante e útil para a pessoa ou equipe.
- **Solicite feedback:** encoraje a cultura de feedback bidirecional, onde todos os membros da equipe se sintam à vontade para compartilhar suas opiniões e contribuições.

Estratégias para Facilitar a Comunicação Aberta e Transparente:

- **Estabeleça canais de comunicação claros**: defina canais de comunicação formais e informais que permitam que as informações fluam livremente dentro da equipe.
- **Promova reuniões regulares:** realize reuniões regulares, como reuniões diárias, retrospectivas e revisões, para discutir o progresso, desafios e próximos passos.
- **Crie um ambiente seguro:** promova uma cultura onde os membros da equipe se sintam seguros para expressar suas opiniões, ideias e preocupações sem medo de retaliação ou julgamento.
- **Pratique a escuta ativa:** esteja presente e ouça atentamente as contribuições dos membros da equipe, demonstrando interesse genuíno e respeito por suas opiniões.

- **Compartilhe informações de forma transparente:** compartilhe informações transparentes sobre o projeto, objetivos organizacionais e decisões para promover confiança e colaboração.

Cultura Organizacional

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Dentro do contexto do Management 3.0, a cultura organizacional desempenha um papel crucial na promoção de ambientes de trabalho ágeis, adaptativos e inovadores. Ao criar e sustentar uma cultura organizacional ágil baseada nesses valores e princípios, as organizações podem promover ambientes de trabalho que são adaptativos, inovadores e centrados nas pessoas, permitindo que elas sejam mais ágeis e eficazes na realização de seus objetivos.

CONSIDERAÇÕES	DESCRIÇÃO
DEFINIÇÃO DE VALORES E PRINCÍPIOS	O primeiro passo para criar uma cultura organizacional ágil é definir claramente os valores e princípios que guiarão o comportamento e as práticas dentro da organização. Isso pode incluir valores como transparência, autonomia, colaboração, respeito mútuo e foco no cliente.
LIDERANÇA EXEMPLAR	A liderança desempenha um papel fundamental na criação e sustentação de uma cultura organizacional ágil. Os líderes devem exemplificar os valores e princípios da cultura ágil em seu próprio comportamento e tomada de decisões, servindo como modelos a serem seguidos pelos membros da equipe.
PROMOÇÃO DA AUTONOMIA E EMPODERAMENTO	Uma cultura ágil valoriza a autonomia e o empoderamento dos indivíduos e equipes. Isso envolve dar às equipes a liberdade para tomar decisões sobre como realizar o trabalho e resolver desafios, promovendo assim um maior senso de responsabilidade e comprometimento.
FOMENTO DA COLABORAÇÃO	Uma cultura ágil promove a colaboração e o compartilhamento de conhecimento entre os membros da equipe. Isso pode ser facilitado por meio de práticas como pair programming, revisões de código, sessões de brainstorming e reuniões regulares de compartilhamento de conhecimento.
INCENTIVO AO FEEDBACK E À MELHORIA CONTÍNUA	Uma cultura ágil incentiva o feedback regular e construtivo entre os membros da equipe, promovendo assim a melhoria contínua. Os líderes devem criar um ambiente seguro e encorajador onde os membros da equipe se sintam à vontade para compartilhar suas opiniões, ideias e preocupações.
FLEXIBILIDADE E ADAPTAÇÃO	Uma cultura ágil valoriza a flexibilidade e a capacidade de adaptação às mudanças no ambiente de negócios. Isso envolve encorajar a experimentação, a inovação e a disposição para ajustar planos e estratégias conforme necessário para alcançar os objetivos organizacionais.
RECONHECIMENTO E CELEBRAÇÃO DE CONQUISTAS	Uma cultura ágil reconhece e celebra as conquistas individuais e coletivas, promovendo assim um ambiente de trabalho positivo e motivador. Isso pode incluir práticas como premiações, reconhecimento público e celebrações de marcos alcançados.

Gestão de Mudanças

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Dentro do contexto do Management 3.0, a gestão de mudanças é uma área crucial, uma vez que as organizações ágeis estão constantemente se adaptando e evoluindo para enfrentar os desafios do ambiente de negócios em constante mudança. Vejamos algumas estratégias para gerenciar e liderar mudanças organizacionais, juntamente com abordagens para superar a resistência à mudança:

ESTRATÉGIAS	DESCRIÇÃO
COMUNICAÇÃO TRANSPARENTE	Comunique claramente os motivos para a mudança, os benefícios esperados e o impacto nos membros da equipe e na organização como um todo. É essencial fornecer informações detalhadas e honestas para promover a compreensão e o alinhamento com os objetivos da mudança.
ENVOLVIMENTO E PARTICIPAÇÃO	Envolver os membros da equipe no processo de mudança, permitindo que contribuam com ideias, sugestões e preocupações. Isso promove um senso de propriedade e comprometimento com a mudança, reduzindo a resistência e aumentando o apoio à implementação.
FORNECER APOIO E RECURSOS	Garantir que os membros da equipe tenham acesso aos recursos necessários, como treinamento, suporte técnico e orientação, para se adaptarem à mudança com sucesso. Isso ajuda a reduzir a ansiedade e a incerteza associadas à mudança, aumentando a confiança e a capacidade de enfrentar os desafios.
FOCO NOS BENEFÍCIOS E VALORES	Destaque os benefícios e valores da mudança, concentrando-se em como ela ajudará a organização a alcançar seus objetivos estratégicos e a melhorar a experiência dos clientes, dos membros da equipe e de outras partes interessadas. Isso ajuda a criar um senso de propósito e direção durante o processo de mudança.
GERENCIAR A RESISTÊNCIA	Reconheça e valide preocupações e resistências dos membros da equipe em relação à mudança. Procure entender as razões por trás da resistência e aborde-as de maneira empática. Compartilhe exemplos de sucesso e histórias inspiradoras para demonstrar os benefícios da mudança e incentivar uma atitude mais positiva em relação a ela.
PROMOVER CULTURA DE APRENDIZADO	Cultive uma cultura organizacional que valorize a aprendizagem contínua e a adaptação às mudanças. Encoraje a experimentação, a inovação e a disposição para ajustar abordagens e estratégias conforme necessário para alcançar os objetivos organizacionais.

Gestão de Conflitos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

A gestão de conflitos é uma habilidade essencial para promover um ambiente de trabalho saudável e produtivo. Ao aplicar essas técnicas e abordagens no contexto do Management 3.0, os líderes e gestores podem promover a resolução construtiva de conflitos e fortalecer os relacionamentos dentro da equipe, criando assim um ambiente de trabalho mais harmonioso, colaborativo e produtivo.

Vejamos algumas técnicas para lidar com conflitos de forma construtiva e promover a resolução de conflitos e a construção de relacionamentos saudáveis dentro da equipe:

TÉCNICAS	DESCRIÇÃO
COMUNICAÇÃO ABERTA E TRANSPARENTE	Promova uma cultura de comunicação aberta e transparente, onde os membros da equipe se sintam à vontade para expressar suas opiniões, preocupações e ideias. Uma comunicação clara ajuda a evitar mal-entendidos que podem levar a conflitos.
PRÁTICA DA ESCUTA ATIVA	Desenvolva a habilidade de ouvir ativamente os pontos de vista dos outros. Isso envolve prestar atenção genuína ao que está sendo dito, sem interromper, julgar ou pensar em respostas enquanto o outro está falando.
EMPATIA E PERSPECTIVA	Desenvolva empatia ao tentar compreender o ponto de vista e os sentimentos da outra pessoa. Tente ver a situação a partir da perspectiva dela para ganhar insights sobre as razões por trás de suas ações ou pontos de vista.
RESOLUÇÃO COLABORATIVA DE PROBLEMAS	Encoraje a resolução de problemas de forma colaborativa, onde os membros da equipe trabalham juntos para encontrar soluções que atendam às necessidades de todas as partes envolvidas. Isso promove o senso de colaboração e trabalho em equipe.
MEDIAÇÃO E FACILITAÇÃO	Em situações em que conflitos são mais complexos ou intensos, considere envolver um mediador neutro para facilitar a comunicação e encontrar soluções mutuamente aceitáveis. Um mediador pode ajudar a reduzir as tensões e promover um diálogo construtivo.
FOCO NOS INTERESSES COMUNS	Em vez de se concentrar nas posições ou demandas das partes envolvidas, concentre-se nos interesses comuns subjacentes. Identificar interesses compartilhados pode ajudar a encontrar soluções que atendam às necessidades de todas as partes envolvidas.
APRENDIZADO E CRESCIMENTO	Encoraje os membros da equipe a verem os conflitos como oportunidades de aprendizado e crescimento. Os conflitos bem gerenciados podem levar a insights, inovação e fortalecimento dos relacionamentos dentro da equipe.

Melhoria Contínua

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

A melhoria contínua é um princípio fundamental que visa promover o aprimoramento constante dos processos, práticas e resultados dentro da organização. Ao adotar essas práticas e metodologias, as organizações podem criar uma cultura de melhoria contínua que promova a inovação, a eficiência e a excelência operacional, permitindo que se adaptem rapidamente às mudanças no ambiente de negócios e alcancem um desempenho excepcional ao longo do tempo.

TÉCNICAS	DESCRIÇÃO
KAIZEN	Kaizen é uma filosofia japonesa que significa "melhoria contínua". Ele enfatiza a ideia de que pequenas mudanças incrementais ao longo do tempo podem levar a melhorias significativas e sustentáveis nos processos e resultados organizacionais. Práticas Kaizen geralmente envolvem a participação de todos os membros da equipe na identificação de oportunidades de melhoria, o estabelecimento de metas alcançáveis e a implementação de soluções de baixo custo e baixo risco.
PDCA	O PDCA (Plan, Do, Check, Act) é um ciclo de melhoria contínua composto por quatro etapas, que são repetidas continuamente para impulsionar melhorias contínuas nos processos e resultados. Na etapa de Planejar, são estabelecidos objetivos claros e definidos, identificando as ações necessárias para alcançá-los. Na etapa de Executar, as ações planejadas são implementadas. Na etapa de Verificar, os resultados são avaliados em relação aos objetivos estabelecidos. Na etapa de Agir, são tomadas medidas corretivas e preventivas para garantir que as melhorias sejam mantidas e ampliadas.
BRAINSTORMING E WORKSHOPS	Realize sessões de brainstorming e workshops de melhoria com a participação de membros da equipe de diferentes áreas e níveis hierárquicos. Isso pode gerar uma variedade de ideias criativas para aprimorar processos e resolver problemas existentes.
ANÁLISE DE CAUSA RAIZ	Use técnicas de análise de causa raiz para identificar as causas fundamentais de problemas e falhas nos processos. Isso permite que as equipes abordem os problemas em sua origem e implementem soluções eficazes para evitar sua recorrência.
MÉTRICAS DE DESEMPENHO	Estabeleça métricas e indicadores de desempenho claros e mensuráveis para monitorar o progresso em direção aos objetivos de melhoria. O acompanhamento regular dessas métricas permite avaliar o impacto das mudanças implementadas e identificar áreas adicionais para melhoria.
CULTURA DE EXPERIMENTAÇÃO	Promova uma cultura organizacional que valorize a experimentação e o aprendizado contínuo. Incentive os membros da equipe a tentarem novas abordagens, a aprender com os resultados e a compartilhar conhecimentos e experiências com os colegas.

Liderança Servidora

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

A liderança servidora é um estilo de liderança que se concentra em servir e apoiar os membros da equipe para alcançar seus objetivos, promovendo um ambiente de trabalho colaborativo e de confiança mútua. Ao adotar esses princípios e práticas, os líderes podem criar um ambiente de trabalho inspirador e motivador, onde as equipes se sintam valorizadas, apoiadas e capacitadas para alcançar o sucesso coletivo.

PRINCÍPIOS	DESCRIÇÃO
SERVIÇOS AOS OUTROS	O cerne da liderança servidora é a ideia de que os líderes estão lá para servir às necessidades dos membros da equipe, em vez de serem servidos por eles. Isso envolve colocar as necessidades, interesses e desenvolvimento dos membros da equipe em primeiro lugar.
EMPATIA E COMPREENSÃO	Os líderes servidores praticam a empatia e a compreensão, buscando entender as necessidades, preocupações e perspectivas dos membros da equipe. Isso cria um ambiente onde os membros da equipe se sentem valorizados e apoiados.
CAPACITAÇÃO E DESENVOLVIMENTO	Os líderes servidores capacitam e desenvolvem os membros da equipe, fornecendo-lhes os recursos, oportunidades e suporte necessários para alcançar seu pleno potencial. Isso pode incluir fornecer treinamento, orientação, feedback e oportunidades de aprendizado.
PROMOÇÃO DA AUTONOMIA	Os líderes servidores promovem a autonomia e a responsabilidade dos membros da equipe, permitindo que tenham voz nas decisões que afetam seu trabalho e dando-lhes liberdade para tomar iniciativas e resolver problemas de forma independente.
AMBIENTE DE CONFIANÇA	Os líderes servidores cultivam um ambiente de confiança e respeito mútuo, onde os membros da equipe se sintam seguros para expressar suas opiniões, assumir riscos e cometer erros sem medo de retaliação ou julgamento.
PODER E RECONHECIMENTO	Os líderes servidores compartilham o poder e o reconhecimento com os membros da equipe, valorizando suas contribuições e celebrando seus sucessos. Isso promove um senso de propriedade e pertencimento à equipe.
COMPORTAMENTOS POSITIVOS	Os líderes servidores servem como modelos de comportamentos positivos e éticos, demonstrando integridade, humildade, empatia e comprometimento com os valores e objetivos da equipe.
ESCUTA ATIVA	Os líderes servidores praticam a escuta ativa e fornecem feedback construtivo aos membros da equipe, demonstrando interesse genuíno em suas preocupações e oferecendo orientação para o seu desenvolvimento.

Gestão de Desempenho

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

A gestão de desempenho e avaliação assume uma abordagem mais adaptativa e orientada para o desenvolvimento em comparação com os métodos tradicionais de avaliação de desempenho. Dessa forma, organizações podem promover um ambiente de trabalho mais dinâmico, centrado nas pessoas e orientado para o crescimento e sucesso sustentável. Isso permite que as equipes alcancem seu pleno potencial e contribuam para os objetivos organizacionais.

CARACTERÍSTICAS	DESCRIÇÃO
AVALIAÇÃO CONTÍNUA	Em vez de depender de avaliações formais anuais, adote uma abordagem contínua de avaliação de desempenho, onde o feedback é fornecido regularmente ao longo do tempo. Isso permite que os membros da equipe recebam orientação oportuna e façam ajustes conforme necessário.
FOCO EM OBJETOS E RESULTADOS	Baseie a avaliação de desempenho em objetivos claros e mensuráveis, alinhados com os objetivos estratégicos da organização. Avalie o desempenho com base nos resultados alcançados e no impacto das contribuições individuais para o sucesso da equipe e da organização.
MÉTRICAS E INDICADORES ÁGEIS	Utilize métricas e indicadores de desempenho relevantes para equipes ágeis, como velocidade de entrega, qualidade do produto, satisfação do cliente, e capacidade de resposta a mudanças. Essas métricas fornecem uma visão holística do desempenho da equipe e podem orientar o processo de melhoria contínua.
AUTOAVALIAÇÃO E AVALIAÇÃO POR PARES	Incentive a autoavaliação e a avaliação por pares como parte do processo de avaliação de desempenho. Isso promove a responsabilidade individual e o desenvolvimento de habilidades de autogerenciamento e feedback entre os membros da equipe.
DESENVOLVIMENTO PROFISSIONAL	Além de avaliar o desempenho passado, concentre-se no desenvolvimento e crescimento profissional contínuo dos membros da equipe. Identifique oportunidades de aprendizado e desenvolvimento que ajudem os membros da equipe a expandir suas habilidades e competências para alcançar seus objetivos de carreira.
ABORDAGEM JUSTA E EQUITATIVA	Garanta que o processo de avaliação de desempenho seja justo, transparente e equitativo para todos os membros da equipe. Evite viéses e avalie o desempenho com base em critérios objetivos e relevantes.
APRENDIZADO E MELHORIA CONTÍNUA	Encare a avaliação de desempenho como uma oportunidade de aprendizado e melhoria contínua, tanto para os indivíduos quanto para a equipe como um todo. Use os insights obtidos durante o processo de avaliação para identificar áreas de força e oportunidades de desenvolvimento.

QUESTÕES COMENTADAS

1. **(PROF. DIEGO / INÉDITA – 2024)** Em equipes ágeis, os líderes geralmente têm autoridade centralizada para tomar decisões importantes.

Comentários:

Em equipes ágeis, a autoridade e a tomada de decisões são distribuídas entre os membros da equipe, promovendo uma abordagem descentralizada e colaborativa.

Gabarito: Errado

---

2. **(PROF. DIEGO / INÉDITA – 2024)** A delegação de autoridade em equipes ágeis geralmente resulta em mais autonomia para os membros da equipe.

Comentários:

A delegação de autoridade em equipes ágeis promove maior autonomia para os membros da equipe, permitindo que tenham mais controle sobre suas atividades e decisões relacionadas ao trabalho.

Gabarito: Correto

---

3. **(PROF. DIEGO / INÉDITA – 2024)** Em uma cultura de confiança e colaboração, os membros da equipe são menos propensos a expressar suas opiniões e assumir riscos.

Comentários:

Em uma cultura de confiança e colaboração, os membros da equipe se sentem mais seguros para expressar suas opiniões, ideias e assumir riscos, pois sabem que serão ouvidos e apoiados pela equipe.

Gabarito: Errado

---

4. **(PROF. DIEGO / INÉDITA – 2024)** Na abordagem de Liderança Servidora, os líderes estão mais preocupados em exercer controle e autoridade do que em apoiar e capacitar suas equipes.

Comentários:

Na abordagem de Liderança Servidora, os líderes estão focados em apoiar e capacitar suas equipes para alcançar seus objetivos, promovendo um ambiente de trabalho colaborativo e de confiança mútua.

Gabarito: Errado

---

5. **(PROF. DIEGO / INÉDITA – 2024)** Feedback regular e reconhecimento das contribuições individuais não têm impacto significativo no engajamento das equipes.

Comentários:

O feedback regular e o reconhecimento das contribuições individuais são fundamentais para promover o engajamento das equipes, pois reforçam um ambiente de trabalho positivo e motivador.

Gabarito: Errado

---

6. **(PROF. DIEGO / INÉDITA – 2024)** A teoria de motivação postula que as pessoas são motivadas principalmente pela remuneração financeira.

Comentários:

A teoria de motivação postula que as pessoas são motivadas quando têm oportunidades de satisfazer necessidades psicológicas fundamentais, como autonomia, competência e relacionamento social.

Gabarito: Errado

---

7. **(PROF. DIEGO / INÉDITA – 2024)** Em equipes ágeis, os papéis e responsabilidades dos membros da equipe são flexíveis e adaptativos.

Comentários:

Em equipes ágeis, os papéis e responsabilidades dos membros da equipe tendem a ser mais flexíveis e adaptativos, permitindo que assumam diferentes papéis conforme necessário para atender às demandas do projeto e do time.

Gabarito: Correto

---

8. **(PROF. DIEGO / INÉDITA – 2024)** Na gestão de desempenho em um contexto ágil, a ênfase é colocada principalmente em métricas quantitativas, ignorando aspectos qualitativos do trabalho.

Comentários:

Na gestão de desempenho em um contexto ágil, a ênfase é colocada tanto em métricas quantitativas quanto qualitativas, pois ambas são importantes para avaliar o progresso e o impacto do trabalho da equipe.

Gabarito: Errado

---

9. **(PROF. DIEGO / INÉDITA – 2024)** A delegação de autoridade em equipes ágeis geralmente resulta em uma tomada de decisão mais lenta e ineficiente.

Comentários:

A delegação de autoridade em equipes ágeis promove uma tomada de decisão mais rápida e eficiente, pois capacita os membros da equipe a tomar decisões de forma descentralizada e colaborativa.

Gabarito: Errado

---

10. **(PROF. DIEGO / INÉDITA – 2024)** Em uma cultura de confiança e colaboração, os conflitos entre os membros da equipe são menos prováveis de ocorrer.

Comentários:

Em uma cultura de confiança e colaboração, os conflitos entre os membros da equipe podem ocorrer, mas são abordados de forma construtiva e respeitosa, promovendo o aprendizado e o crescimento da equipe.

Gabarito: Errado

---

11. **(PROF. DIEGO / INÉDITA – 2024)** A comunicação aberta e transparente é uma estratégia eficaz para lidar com conflitos, mas pode não resolver conflitos de longa data entre membros da equipe.

Comentários:

Embora a comunicação aberta e transparente seja essencial para resolver conflitos, conflitos de longa data podem exigir abordagens mais complexas e intervenção externa, como mediação ou facilitação.

Gabarito: Correto

---

12. **(PROF. DIEGO / INÉDITA – 2024)** A prática da escuta ativa é útil para promover a empatia entre membros da equipe, mas pode não resolver conflitos interpessoais profundos.

Comentários:

A escuta ativa é fundamental para promover a empatia e compreender os sentimentos e perspectivas dos outros, o que pode ser crucial para resolver conflitos interpessoais profundos.

Gabarito: Errado

---

13. **(PROF. DIEGO / INÉDITA – 2024)** A resolução colaborativa de problemas é uma técnica eficaz para lidar com conflitos, mas pode ser prejudicada quando as partes envolvidas estão focadas apenas em suas próprias posições.

Comentários:

A resolução colaborativa de problemas depende da disposição das partes envolvidas em colaborar e buscar soluções que atendam aos interesses comuns, em vez de se concentrarem apenas em suas próprias posições.

Gabarito: Correto

---

14. **(PROF. DIEGO / INÉDITA – 2024)** A mediação e facilitação são abordagens semelhantes para resolver conflitos, mas a mediação geralmente envolve a presença de uma autoridade hierárquica para tomar decisões finais.

Comentários:

Tanto a mediação quanto a facilitação envolvem a presença de uma terceira parte neutra para ajudar a facilitar o diálogo e encontrar soluções, mas na mediação, o mediador pode ter autoridade para tomar decisões finais, enquanto na facilitação, o objetivo é facilitar o processo de tomada de decisão sem impor uma solução específica.

Gabarito: Errado

---

15. **(PROF. DIEGO / INÉDITA – 2024)** A concentração nos interesses comuns é uma abordagem eficaz para resolver conflitos, pois ajuda a identificar áreas de convergência e encontrar soluções mutuamente aceitáveis.

Comentários:

Ao concentrar-se nos interesses comuns, as partes envolvidas podem encontrar soluções que atendam às necessidades de todos, promovendo assim a resolução construtiva de conflitos.

Gabarito: Correto

---

16. **(PROF. DIEGO / INÉDITA – 2024)** A aprendizagem e o crescimento são frequentemente resultados positivos de conflitos bem gerenciados, pois podem levar a insights, inovação e fortalecimento dos relacionamentos dentro da equipe.

Comentários:

Conflitos bem gerenciados podem levar a aprendizado e crescimento, pois podem estimular a criatividade, levar a insights e promover um entendimento mais profundo entre os membros da equipe.

Gabarito: Errado

---

17. **(PROF. DIEGO / INÉDITA – 2024)** A liderança servidora promove a autonomia dos membros da equipe, enquanto a gestão de desempenho tradicional tende a ser mais autoritária.

Comentários:

A liderança servidora valoriza a autonomia e a responsabilidade dos membros da equipe, enquanto a gestão de desempenho tradicional muitas vezes é baseada em diretrizes e supervisão mais rígidas.

Gabarito: Correto

---

18. **(PROF. DIEGO / INÉDITA – 2024)** O feedback regular e construtivo é uma prática comum tanto na liderança servidora quanto na gestão de desempenho, mas sua ênfase pode ser diferente, sendo mais centrada no desenvolvimento pessoal na liderança servidora e mais voltada para resultados na gestão de desempenho.

Comentários:

Embora o feedback seja importante em ambos os contextos, na liderança servidora, o foco pode estar mais no crescimento pessoal e no desenvolvimento dos membros da equipe, enquanto na gestão de desempenho tradicional, pode estar mais na avaliação do cumprimento de metas e resultados.

Gabarito: Correto

---

19. **(PROF. DIEGO / INÉDITA – 2024)** A cultura de experimentação é mais prevalente na gestão de desempenho do que na melhoria contínua, uma vez que se concentra em encontrar novas abordagens para atingir objetivos organizacionais.

Comentários:

A cultura de experimentação é essencial tanto na melhoria contínua quanto na gestão de desempenho, pois ambas buscam constantemente maneiras de melhorar e alcançar resultados mais eficazes.

Gabarito: Errado

---

20. (PROF. DIEGO / INÉDITA – 2024) A análise de causa raiz é uma técnica comumente usada na gestão de desempenho para identificar as origens de problemas de desempenho individual, mas raramente é aplicada na melhoria contínua de processos organizacionais.

Comentários:

A análise de causa raiz é uma técnica valiosa tanto na gestão de desempenho quanto na melhoria contínua, pois ajuda a identificar as verdadeiras causas de problemas e falhas, permitindo que sejam abordadas de maneira eficaz e preventiva.

Gabarito: Errado

---

21. (PROF. DIEGO / INÉDITA – 2024) A cultura organizacional pode influenciar tanto a eficácia da liderança servidora quanto o sucesso da gestão de desempenho, pois afeta a forma como os membros da equipe percebem e respondem aos líderes e aos processos de avaliação.

Comentários:

A cultura organizacional desempenha um papel fundamental na maneira como os líderes são percebidos e como os processos de avaliação são recebidos pelos membros da equipe, impactando diretamente a eficácia da liderança servidora e da gestão de desempenho.

Gabarito: Correto

Lista de Questões

1. **(PROF. DIEGO / INÉDITA – 2024)** Em equipes ágeis, os líderes geralmente têm autoridade centralizada para tomar decisões importantes.

2. **(PROF. DIEGO / INÉDITA – 2024)** A delegação de autoridade em equipes ágeis geralmente resulta em mais autonomia para os membros da equipe.

3. **(PROF. DIEGO / INÉDITA – 2024)** Em uma cultura de confiança e colaboração, os membros da equipe são menos propensos a expressar suas opiniões e assumir riscos.

4. **(PROF. DIEGO / INÉDITA – 2024)** Na abordagem de Liderança Servidora, os líderes estão mais preocupados em exercer controle e autoridade do que em apoiar e capacitar suas equipes.

5. **(PROF. DIEGO / INÉDITA – 2024)** Feedback regular e reconhecimento das contribuições individuais não têm impacto significativo no engajamento das equipes.

6. **(PROF. DIEGO / INÉDITA – 2024)** A teoria de motivação postula que as pessoas são motivadas principalmente pela remuneração financeira.

7. **(PROF. DIEGO / INÉDITA – 2024)** Em equipes ágeis, os papéis e responsabilidades dos membros da equipe são flexíveis e adaptativos.

8. **(PROF. DIEGO / INÉDITA – 2024)** Na gestão de desempenho em um contexto ágil, a ênfase é colocada principalmente em métricas quantitativas, ignorando aspectos qualitativos do trabalho.

9. **(PROF. DIEGO / INÉDITA – 2024)** A delegação de autoridade em equipes ágeis geralmente resulta em uma tomada de decisão mais lenta e ineficiente.

10. **(PROF. DIEGO / INÉDITA – 2024)** Em uma cultura de confiança e colaboração, os conflitos entre os membros da equipe são menos prováveis de ocorrer.

11. **(PROF. DIEGO / INÉDITA – 2024)** A comunicação aberta e transparente é uma estratégia eficaz para lidar com conflitos, mas pode não resolver conflitos de longa data entre membros da equipe.

12. **(PROF. DIEGO / INÉDITA – 2024)** A prática da escuta ativa é útil para promover a empatia entre membros da equipe, mas pode não resolver conflitos interpessoais profundos.

13. **(PROF. DIEGO / INÉDITA – 2024)** A resolução colaborativa de problemas é uma técnica eficaz para lidar com conflitos, mas pode ser prejudicada quando as partes envolvidas estão focadas apenas em suas próprias posições.

14. **(PROF. DIEGO / INÉDITA – 2024)** A mediação e facilitação são abordagens semelhantes para resolver conflitos, mas a mediação geralmente envolve a presença de uma autoridade hierárquica para tomar decisões finais.

15. **(PROF. DIEGO / INÉDITA – 2024)** A concentração nos interesses comuns é uma abordagem eficaz para resolver conflitos, pois ajuda a identificar áreas de convergência e encontrar soluções mutuamente aceitáveis.

16. **(PROF. DIEGO / INÉDITA – 2024)** A aprendizagem e o crescimento são frequentemente resultados positivos de conflitos bem gerenciados, pois podem levar a insights, inovação e fortalecimento dos relacionamentos dentro da equipe.

17. **(PROF. DIEGO / INÉDITA – 2024)** A liderança servidora promove a autonomia dos membros da equipe, enquanto a gestão de desempenho tradicional tende a ser mais autoritária.

18. **(PROF. DIEGO / INÉDITA – 2024)** O feedback regular e construtivo é uma prática comum tanto na liderança servidora quanto na gestão de desempenho, mas sua ênfase pode ser diferente, sendo mais centrada no desenvolvimento pessoal na liderança servidora e mais voltada para resultados na gestão de desempenho.

19. **(PROF. DIEGO / INÉDITA – 2024)** A cultura de experimentação é mais prevalente na gestão de desempenho do que na melhoria contínua, uma vez que se concentra em encontrar novas abordagens para atingir objetivos organizacionais.

20. **(PROF. DIEGO / INÉDITA – 2024)** A análise de causa raiz é uma técnica comumente usada na gestão de desempenho para identificar as origens de problemas de desempenho individual, mas raramente é aplicada na melhoria contínua de processos organizacionais.

21. **(PROF. DIEGO / INÉDITA – 2024)** A cultura organizacional pode influenciar tanto a eficácia da liderança servidora quanto o sucesso da gestão de desempenho, pois afeta a forma como os membros da equipe percebem e respondem aos líderes e aos processos de avaliação.

GABARITO

1. ERRADO
2. CORRETO
3. ERRADO
4. ERRADO
5. ERRADO
6. ERRADO
7. CORRETO
8. ERRADO
9. ERRADO
10. ERRADO
11. CORRETO
12. ERRADO
13. CORRETO
14. ERRADO
15. CORRETO
16. ERRADO
17. CORRETO
18. CORRETO
19. ERRADO
20. ERRADO
21. CORRETO

TÉCNICAS DE PRIORIZAÇÃO DE BACKLOG

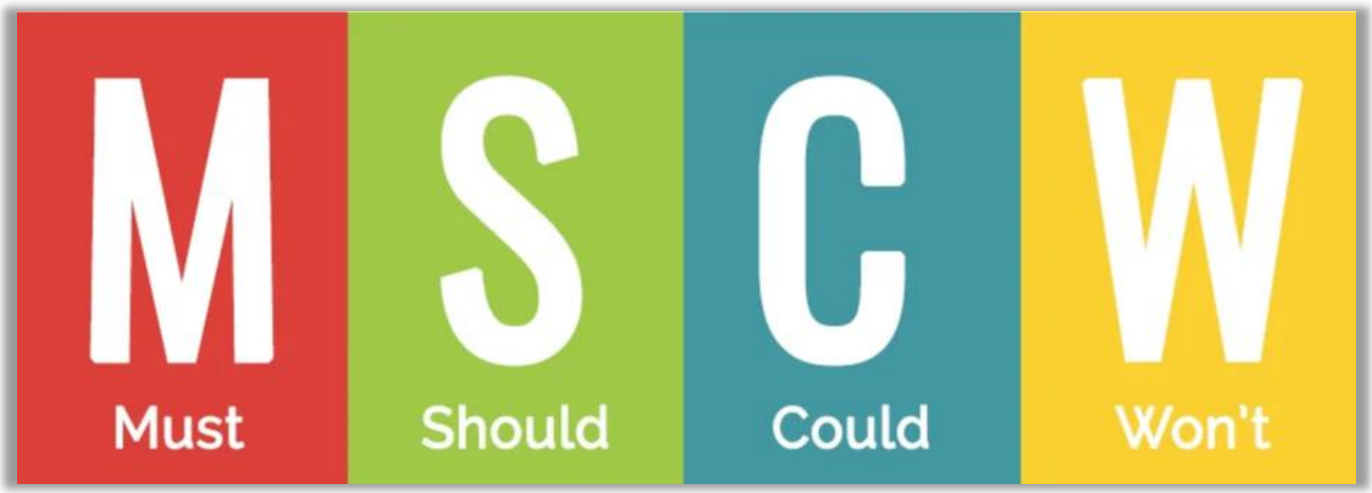
Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

A priorização do backlog do produto é uma atividade crítica no gerenciamento ágil de projetos, ajudando a garantir que a equipe se concentre nas tarefas mais importantes que agregam valor ao produto final. Galera, quem já trabalhou com projeto de software sabe como é difícil, custoso, complexo, trabalhoso, chato, árduo e ingrato fazer com que o cliente consiga priorizar suas necessidades. Em geral, clientes acreditam que absolutamente tudo é importante!

MoSCoW

INCIDÊNCIA EM PROVA: BAIXA



O Método MoSCoW é uma técnica de priorização que ajuda equipes de projeto a categorizar itens do backlog ou requisitos do projeto em quatro categorias distintas, baseadas em sua importância e urgência para a entrega do projeto. Essas categorias são: Must Have (Deve Ter), Should Have (Deveria Ter), Could Have (Poderia Ter) e Won't Have (Não Terá Desta Vez). Vamos ver um determinado contexto para explicar melhor cada categoria.

EXEMPLO CONCRETO

Imagine que uma equipe de desenvolvimento de software esteja trabalhando em um projeto para criar um novo aplicativo de gerenciamento de projetos. Durante a fase de planejamento do projeto, eles usam o Método MoSCoW para priorizar os recursos e funcionalidades que o aplicativo deve ter. Nos parágrafos seguintes, nós vamos ver a definição de cada uma das categorias e veremos exemplos dessas categorias no exemplo da equipe de desenvolvimento de software.

MUST HAVE [DEVE TER]	Itens classificados como "Must Have" são essenciais para o sucesso do projeto. Sem eles, o projeto não pode ser considerado completo ou bem-sucedido. Esses itens são críticos e não negociáveis.

- **Criação e Gerenciamento de Tarefas:** capacidade de criar, atribuir e gerenciar tarefas dentro de projetos.
- **Controle de Prazos e Calendário:** ferramentas para definir prazos e visualizar tarefas em um calendário.
- **Sistema de Notificações:** notificações para alertar os usuários sobre prazos iminentes e mudanças nas tarefas.

SHOULD HAVE [DEVERIA TER]	Itens "Should Have" são importantes, mas não críticos. São recursos ou requisitos que agregam valor significativo ao projeto, mas sua ausência não inviabiliza o lançamento do produto ou serviço.

- **Relatórios de Progresso do Projeto:** funcionalidade para gerar relatórios visuais sobre o progresso do projeto.
- **Integração com E-mails:** capacidade de enviar atualizações de tarefas e notificações diretamente por e-mail.

COULD HAVE [PODERIA TER]	Itens classificados como "Could Have" são desejáveis, mas menos importantes e com menor impacto imediato no projeto. Geralmente, são incluídos se o tempo e os recursos permitirem.

- **Relatórios de Progresso do Projeto:** funcionalidade para gerar relatórios visuais sobre o progresso do projeto.
- **Integração com E-mails:** capacidade de enviar atualizações de tarefas e notificações diretamente por e-mail.

WON'T HAVE [NÃO TERÁ DESTA VEZ]	Itens "Won't Have" são aqueles que foram identificados e reconhecidos como de menor prioridade para o ciclo atual do projeto. São excluídos da entrega iminente, mas podem ser reconsiderados em fases futuras.

- **Relatórios de Progresso do Projeto:** funcionalidade para gerar relatórios visuais sobre o progresso do projeto.
- **Integração com E-mails:** capacidade de enviar atualizações de tarefas e notificações diretamente por e-mail.

Ao categorizar as funcionalidades usando o Método MoSCoW, a equipe de desenvolvimento pode claramente identificar quais recursos são essenciais para o lançamento inicial do aplicativo e quais podem ser adiados para versões futuras. **Isso ajuda a equipe a focar seus esforços nos aspectos mais críticos do aplicativo, garantindo que o produto final atenda às necessidades básicas dos usuários enquanto permanece dentro do orçamento e do cronograma do projeto.**

Além disso, ajuda na comunicação com as partes interessadas, oferecendo uma visão clara do que esperar no lançamento e o que pode ser incluído em atualizações futuras.

Scorecard

INCIDÊNCIA EM PROVA: BAIXA

A técnica de Scorecard para priorização de backlog é um método estruturado que ajuda equipes a avaliar e priorizar itens de backlog com base em um conjunto de critérios pré-definidos. Diferentemente de abordagens mais subjetivas, o Scorecard atribui valores numéricos ou pontuações a cada item do backlog segundo critérios específicos, facilitando comparações objetivas entre diferentes tarefas ou histórias de usuário.

Definição de Critérios: primeiramente, a equipe define critérios relevantes para a avaliação dos itens do backlog. Esses critérios podem incluir:

DEFINIÇÃO DE CRITÉRIOS	Primeiramente, a equipe define critérios relevantes para a avaliação dos itens do backlog. Esses critérios podem incluir: valor para o cliente; objetivos do negócio; esforço ou complexidade; risco; e dependências.
ATRIBUIÇÃO DE PONTUAÇÕES	Cada item de backlog é avaliado contra os critérios definidos, com pontuações atribuídas de acordo com o grau em que o item atende a cada critério. As pontuações podem ser simples (Ex: de 1 a 5) ou baseadas em uma escala mais complexa.
CÁLCULO DA PONTUAÇÃO	Após avaliar todos os itens contra cada critério e considerar os pesos atribuídos, calcula-se a pontuação total para cada item. Isso é feito multiplicando a pontuação de cada critério pelo seu peso correspondente e somando os resultados.
RESULTADO DA PRIORIZAÇÃO	Os itens do backlog são então classificados com base em suas pontuações totais, do mais alto ao mais baixo, permitindo à equipe identificar quais itens devem ser priorizados para desenvolvimento ou ação.

EXEMPLO CONCRETO

Imaginemos uma equipe de desenvolvimento de software que está trabalhando em um novo aplicativo de finanças pessoais. Com muitas ideias para funcionalidades, a equipe precisa priorizar o backlog de produto de maneira eficaz. Eles decidem usar a técnica de Scorecard para ajudar na priorização. **Na Definição de Critérios, a equipe define os seguintes critérios, cada um com uma escala de 1 a 5, onde 5 indica a maior importância ou impacto, e 1 a menor:**

Definição de Critérios | A equipe define os seguintes critérios, cada um com uma escala de 1 a 5, onde 5 indica a maior importância ou impacto, e 1 a menor:

- **Valor para o Cliente/Usuário:** o quanto a funcionalidade agrega valor ou resolve um problema para o usuário final;
- **Alinhamento Estratégico:** como a funcionalidade se alinha com os objetivos de longo prazo da empresa;
- **Viabilidade Técnica:** trata-se da facilidade ou dificuldade técnica de implementar a funcionalidade;
- **Impacto no Mercado:** o potencial da funcionalidade em aumentar a participação de mercado ou atrair novos usuários;

- **Risco:** os riscos associados à implementação da funcionalidade, incluindo riscos técnicos e de negócios.

Itens do Backlog para Priorização | A equipe seleciona três funcionalidades para avaliação:

- **Funcionalidade A:** implementação de um recurso de orçamento automatizado;
- **Funcionalidade B:** adição de uma ferramenta de investimento com inteligência artificial;
- **Funcionalidade C:** integração de pagamentos móveis;

Atribuição e Cálculo da Pontuação | A equipe avalia cada funcionalidade contra os critérios definidos e faz a conta total:

CRITÉRIO/CATEGORIA	FUNCIONALIDADE A	FUNCIONALIDADE B	FUNCIONALIDADE C
VALOR PARA O CLIENTE	5	4	3
ALINHAMENTO ESTRATÉGICO	4	3	5
VIABILIDADE TÉCNICA	3	2	4
IMPACTO NO MERCADO	3	5	4
RISCO	2	4	3
TOTAL	17	18	19

Resultado da Priorização | Com base nas pontuações totais, a Funcionalidade C (Integração de pagamentos móveis) é a mais alta prioridade, seguida pela Funcionalidade B (ferramenta de investimento com IA) e, por último, a Funcionalidade A (recurso de orçamento automatizado).

Este exemplo mostra como a técnica de Scorecard pode ser usada para avaliar objetivamente diferentes funcionalidades contra critérios relevantes, ajudando a equipe a tomar decisões informadas sobre a priorização do backlog. **Essa abordagem garante que os itens mais valiosos e estratégicos sejam priorizados, levando em conta não apenas o valor para o cliente, mas também a viabilidade, o impacto no mercado e os riscos associados.**

BUC

INCIDÊNCIA EM PROVA: BAIXA

O BUC (Benefício x Custo x Urgência) é uma técnica estruturada que ajuda as equipes a avaliar e classificar itens do backlog de um projeto com base em três critérios principais: Benefício, Custo e Urgência. Essa metodologia proporciona uma visão equilibrada sobre quais funcionalidades ou tarefas devem ser priorizadas, considerando não apenas o valor que elas trazem, mas também os recursos necessários para sua implementação e a importância do tempo.

BENEFÍCIO	Este critério avalia o valor ou o impacto positivo que a implementação de um item do backlog trará para o projeto, a organização, os usuários finais ou os clientes. Itens que oferecem maiores benefícios geralmente recebem uma priorização mais alta.
CUSTO	Refere-se aos recursos necessários para implementar um item do backlog. Isso pode incluir tempo de desenvolvimento, esforço da equipe, custos financeiros, entre outros. Itens que exigem menos recursos (custo menor) podem ser mais atraentes para implementação rápida.
URGÊNCIA	Este critério considera o quão crítico é implementar um item do backlog dentro de um determinado período de tempo. Itens urgentes podem ser aqueles que são necessários para cumprir prazos legais, de mercado ou internos específicos.

Vejamos as etapas do processo de priorização do BUC:

AVALIAÇÃO	Cada item do backlog é avaliado em relação a cada um dos três critérios. Isso geralmente é feito através de discussões em equipe, podendo-se utilizar escalas numéricas para quantificar cada critério.
PONTUAÇÃO E COMPARAÇÃO	Após a avaliação, cada item recebe uma pontuação geral que combina os três critérios. Há maneiras de combinar essas pontuações, incluindo fórmulas matemáticas simples ou modelos ponderados que dão mais importância a critérios específicos.
PRIORIZAÇÃO	Os itens contidos do backlog são – portanto – classificados com base em suas pontuações totais. Dessa forma, é possível ajudar a equipe a identificar quais devem ser abordados primeiro.

O sucesso da aplicação do BUC depende da capacidade da equipe de avaliar precisamente cada um dos critérios e de aplicar consistentemente essas avaliações em todo o backlog. Além disso, a técnica requer um equilíbrio cuidadoso entre os critérios para evitar sobrevalorizar um em detrimento dos outros, garantindo que as decisões de priorização se alinhem estrategicamente com os objetivos de longo prazo do projeto.

EXEMPLO CONCRETO

Imaginemos uma equipe de desenvolvimento de software encarregada de criar um novo sistema de gerenciamento de Recursos Humanos (RH) para uma grande empresa. O backlog do projeto está cheio de funcionalidades sugeridas, mas a equipe precisa priorizá-las para o próximo

ciclo de desenvolvimento. Para fazer isso, eles decidem aplicar a técnica BUC. Vejamos a definição dos Critérios BUC:

BENEFÍCIO [B]	O impacto positivo que a funcionalidade terá na eficiência operacional do departamento de RH e na satisfação dos funcionários.
URGÊNCIA [U]	A necessidade de implementar a funcionalidade para atender a prazos legais ou de negócios.
CUSTO [C]	Os recursos necessários para desenvolver a funcionalidade, incluindo tempo, mão de obra e outros custos associados.

Dito isso, vamos considerar que temos os seguintes itens do backlog para priorização:

1. **Portal do Funcionário para Autoatendimento**
2. **Integração com Software de Folha de Pagamento**
3. **Sistema de Rastreamento de Candidatos**

Agora para cada item, nós vamos fazer a Avaliação BUC e Priorização:

1. Portal do Funcionário para Autoatendimento

BENEFÍCIO	ALTO [5]	Melhorará significativamente a satisfação dos funcionários e reduzirá a carga de trabalho administrativo do RH.
URGÊNCIA	MÉDIO [3]	Importante para a satisfação dos funcionários, mas não existe um prazo legal.
CUSTO	MÉDIO [3]	Significa que requer uma quantidade relativamente moderada de desenvolvimento.

2. Integração com Software de Folha de Pagamento

BENEFÍCIO	ALTO [5]	Eliminação de processos manuais e redução de erros.
URGÊNCIA	ALTO [5]	Necessário para cumprir os novos regulamentos fiscais.
CUSTO	ALTO [4]	Complexidade técnica e tempo necessário são consideráveis.

3. Sistema de Rastreamento de Candidatos

BENEFÍCIO	MÉDIO [3]	Melhora o processo de recrutamento, mas não afeta diretamente a maioria dos funcionários existentes.
URGÊNCIA	BAIXO [2]	Não há prazos imediatos ou regulamentações que exijam essa funcionalidade.
CUSTO	BAIXO [2]	Relativamente simples de implementar, usando soluções de mercado.

Com base na Avaliação BUC, a equipe poderá decidir priorizar as funcionalidades da seguinte forma:

- **Integração com Software de Folha de Pagamento:** devido à sua alta urgência e benefício significativo, apesar do custo alto.
- **Portal do Funcionário para Autoatendimento:** devido ao seu alto benefício e custo moderado, apesar da urgência média.
- **Sistema de Rastreamento de Candidatos:** devido ao seu benefício e urgência mais baixos, junto com um custo baixo.

Este exemplo demonstra como a Técnica BUC permite à equipe fazer escolhas estratégicas e informadas sobre a priorização do backlog, considerando uma visão equilibrada de benefício, urgência e custo para cada funcionalidade proposta. **Por meio dessa técnica, garante-se que recursos limitados sejam alocados de maneira eficaz para maximizar o valor entregue pelo projeto.**

Testes de Suposição

INCIDÊNCIA EM PROVA: BAIXA

A técnica de Testes de Suposição para a priorização de backlog é uma abordagem que se concentra na validação de hipóteses ou suposições subjacentes aos itens do backlog, especialmente em contextos em que a incerteza é alta. Esta técnica é particularmente relevante para projetos que adotam metodologias ágeis e Lean Startup, onde o aprendizado rápido e a adaptação com base em feedback real são valorizados.

O objetivo é identificar quais funcionalidades devem ser priorizadas para teste ou desenvolvimento, com base na necessidade de validar suposições críticas para o sucesso do projeto. As etapas são:

ETAPAS	DESCRIÇÃO
IDENTIFICAÇÃO DE SUPOSIÇÕES	A equipe lista as suposições importantes que estão por trás de cada item do backlog. Essas suposições podem estar relacionadas ao valor que um recurso oferecerá aos usuários, à viabilidade técnica de uma funcionalidade ou à demanda do mercado por um produto.
AVALIAÇÃO DE RISCO E IMPACTO	Para cada suposição, a equipe avalia o risco associado (a probabilidade de a suposição estar errada) e o impacto potencial no projeto se a suposição for incorreta. Itens com suposições de alto risco e alto impacto são candidatos prioritários para testes.
DESIGN DE TESTES	Para as suposições identificadas como prioritárias, a equipe projeta experimentos ou testes mínimos viáveis para validar rapidamente essas suposições. Isso pode incluir protótipos, pesquisas com usuários, análises de dados de mercado ou outros métodos de feedback rápido.
EXECUÇÃO E APRENDIZADO	A equipe executa os testes projetados e coleta dados que ajudam a validar ou refutar as suposições. Com base nos resultados, ajustes são feitos no backlog e na estratégia do projeto conforme necessário.

Essa técnica ajuda a identificar e mitigar riscos no início do processo de desenvolvimento, focando na validação de suposições críticas. Ela também garante que o esforço de desenvolvimento esteja focado em funcionalidades que oferecem valor real aos usuários e ao negócio, evitando desperdício de recursos. Por fim, ela promove uma cultura de aprendizado e adaptação contínua, essencial para ambientes dinâmicos e incertos.

Por outro lado, essa técnica também requer uma cultura organizacional que valorize a experimentação e esteja confortável com a falha como uma forma de aprendizado. É importante gerenciar as expectativas das partes interessadas sobre o processo de teste e os ajustes que podem ser necessários no projeto com base nos resultados dos testes. Dito isso, vamos ver um exemplo concreto de uso dessa técnica.

EXEMPLO CONCRETO

Imaginemos uma startup que está desenvolvendo um novo aplicativo móvel para ajudar pessoas a encontrar companheiros de corrida locais. A equipe tem várias ideias sobre funcionalidades que poderiam tornar o aplicativo valioso para seus usuários, mas antes de se

comprometerem com um desenvolvimento extenso, eles decidem usar a técnica de Testes de Suposição para validar suas ideias principais. Vejamos as suposições do projeto:

SUPOSIÇÃO A	Usuários estão dispostos a pagar por funcionalidades premium, como análise avançada do desempenho em corridas.
SUPOSIÇÃO B	Existe uma demanda significativa por um sistema de matchmaking que conecta corredores com habilidades similares e interesses em corridas.
SUPOSIÇÃO C	Os usuários utilizarão ativamente a funcionalidade de eventos para organizar e participar de corridas locais.

Agora fazemos uma avaliação de Risco/Impacto:

SUPOSIÇÃO A	**ALTO RISCO/ ALTO IMPACTO**	Se os usuários não estiverem dispostos a pagar, o modelo de receita previsto pode falhar.
SUPOSIÇÃO B	**MÉDIO RISCO/ ALTO IMPACTO**	Essencial para a proposta de valor do aplicativo, mas alternativas de validação são viáveis.
SUPOSIÇÃO C	**BAIXO RISCO/ MÉDIO IMPACTO**	Contribui para a retenção de usuários, mas não é crucial para o lançamento inicial.

Feito isso, agora vamos falar sobre Design de Testes:

▪ **PARA A SUPOSIÇÃO A**	A equipe decide criar uma pesquisa online direcionada aos potenciais usuários do aplicativo, questionando sobre a disposição de pagar por funcionalidades premium e quais funcionalidades seriam mais valiosas.
PARA A SUPOSIÇÃO B	A equipe desenvolve um protótipo mínimo viável (MVP) do sistema de matchmaking e convida um pequeno grupo de usuários locais para testá-lo, observando a frequência de uso e coletando feedback direto.
PARA A SUPOSIÇÃO C	Decide-se postergar testes mais elaborados até que as duas suposições mais críticas sejam validadas, usando inicialmente feedback de fóruns e grupos de corrida online para medir o interesse.

Finalizado, vamos falar sobre Execução e Aprendizado:

SUPOSIÇÃO A	A pesquisa revela que a maioria dos usuários potenciais não está disposta a pagar por funcionalidades premium, preferindo um modelo suportado por anúncios. A equipe ajusta o modelo de receita.
SUPOSIÇÃO B	O teste do MVP mostra que os usuários estão muito engajados com o sistema de matchmaking, validando a demanda por essa funcionalidade. A equipe prioriza o desenvolvimento completo dessa funcionalidade.
SUPOSIÇÃO C	Feedback preliminar indica interesse na funcionalidade de eventos, mas com menor prioridade do que o sistema de matchmaking.

Ao aplicar Testes de Suposição, a equipe foi capaz de validar suas ideias críticas de forma eficaz, ajustar suas estratégias com base em dados reais e priorizar o desenvolvimento de

funcionalidades que oferecem o maior valor para seus usuários. Isso permitiu que a startup utilizasse seus recursos de forma mais eficiente e aumentasse as chances de sucesso do aplicativo no mercado.

Valor de Negócio x Risco

INCIDÊNCIA EM PROVA: BAIXA

Essa técnica de priorização de backlog é um método estratégico utilizado para classificar e priorizar itens de backlog com base em dois critérios principais: o valor que cada item traz para o negócio e o risco associado à sua implementação. Essa abordagem ajuda as equipes a tomar decisões informadas sobre quais funcionalidades ou tarefas devem ser desenvolvidas primeiro, equilibrando potenciais benefícios com os desafios ou incertezas envolvidos. *E como funciona?*

ETAPAS	DESCRIÇÃO
AVALIAÇÃO DO VALOR DE NEGÓCIO	O valor de negócio é determinado considerando o impacto que a implementação de um item do backlog terá sobre os objetivos estratégicos da empresa, a satisfação do cliente, a receita, a eficiência operacional, entre outros fatores. Itens que são considerados de alto valor são aqueles que contribuem significativamente para o sucesso e crescimento do negócio.
AVALIAÇÃO DO RISCO	O risco envolve avaliar as incertezas e desafios associados à implementação de um item do backlog. Isso pode incluir riscos técnicos, riscos de mercado, dependências de outras tarefas ou funcionalidades, e qualquer outro fator que possa impactar a entrega ou o sucesso do item. Itens com riscos altos requerem atenção especial e podem necessitar de mais recursos ou preparação.

***E como ocorre o processo de priorização?* Após avaliar cada item do backlog em termos de valor de negócio e risco, a equipe utiliza essas informações para priorizar o trabalho.** Isso geralmente é feito de forma a maximizar o valor entregue, minimizando ao mesmo tempo os riscos para o projeto. Uma forma comum de visualizar isso é através de uma matriz 2x2, onde um eixo representa o valor de negócio e o outro o risco.

Itens de alto valor e baixo risco são priorizados, enquanto itens de baixo valor e alto risco podem ser reavaliados ou adiados. Essa técnica tem os seguintes benefícios:

- **Foco no Valor:** garante que a equipe se concentre em itens que oferecem o maior retorno sobre o investimento.
- **Gerenciamento de Riscos:** ajuda a identificar e mitigar riscos no início do ciclo de desenvolvimento.
- **Alocação Eficiente de Recursos:** direciona recursos para áreas que proporcionam o maior impacto positivo no negócio, ao mesmo tempo em que considera os desafios envolvidos.
- **Decisões Informadas:** fornece uma base sólida para tomada de decisões sobre o que incluir em sprints ou iterações futuras.

Embora essa técnica ofereça uma abordagem equilibrada para a priorização do backlog, ela requer uma compreensão clara dos objetivos estratégicos do negócio e uma avaliação precisa tanto do valor quanto do risco. Além disso, é importante revisar e ajustar a priorização regularmente à medida que novas informações se tornam disponíveis ou quando mudanças ocorrem no ambiente de negócios ou tecnológico.

EXEMPLO CONCRETO

Vamos considerar uma empresa de software que está desenvolvendo uma nova plataforma de e-commerce destinada a pequenas e médias empresas (PMEs). O backlog do projeto está repleto de funcionalidades propostas, mas a equipe de desenvolvimento precisa priorizá-las para garantir que o produto final atenda às necessidades do mercado de forma eficaz, dentro do orçamento e do prazo estabelecidos. A equipe decide aplicar a técnica, começando pela definição dos critérios:

DEFINIÇÃO DOS CRITÉRIOS	DESCRIÇÃO
VALOR DE NEGÓCIO	Contribuição de cada funcionalidade para os objetivos estratégicos da empresa, como aumentar a base de clientes, melhorar a satisfação do cliente, aumentar a receita, etc.
RISCO	Considerações incluem complexidade técnica, incertezas de desenvolvimento, dependências externas, e potenciais impactos no prazo e custo.

ITENS DO BACKLOG PARA PRIORIZAÇÃO	DESCRIÇÃO
INTEGRAÇÃO C/ SISTEMAS DE PAGAMENTO POPULARES	Permitir que os usuários escolham entre múltiplos sistemas de pagamento.
SISTEMA DE RECOMENDAÇÃO DE PRODUTOS BASEADO EM IA	Recomendar produtos aos usuários com base em seu histórico de navegação e compra.
FERRAMENTA DE ANÁLISE DE DADOS DE VENDAS	Oferecer aos vendedores análises detalhadas de suas vendas e tendências do mercado.

INTEGRAÇÃO C/ SISTEMAS DE PAGAMENTO POPULARES	DESCRIÇÃO
VALOR DE NEGÓCIO	Alto: essencial para facilitar as vendas e melhorar a experiência do cliente.
RISCO	Baixo: existem muitas soluções prontas que podem ser integradas com esforço técnico moderado.

SISTEMA DE RECOMENDAÇÃO DE PRODUTOS BASEADO EM IA	DESCRIÇÃO
VALOR DE NEGÓCIO	Médio: pode aumentar as vendas por meio de compras impulsivas e melhorar a experiência do usuário.
RISCO	Alto: requer um desenvolvimento complexo e pode envolver questões de privacidade dos dados.

FERRAMENTA DE ANÁLISE DE DADOS DE VENDAS	DESCRIÇÃO
VALOR DE NEGÓCIO	Médio: útil para vendedores, mas não essencial para operações do dia a dia.

RISCO	Médio: desenvolvimento técnico é desafiador, mas existem bibliotecas e ferramentas que podem ser utilizadas.

Com base nesta análise, a equipe decide priorizar as funcionalidades da seguinte maneira:

PRIORIZAÇÃO BASEADA EM VALOR DE NEGÓCIO X RISCO	DESCRIÇÃO
INTEGRAÇÃO C/ SISTEMAS DE PAGAMENTO POPULARES	Devido ao seu alto valor de negócio e baixo risco, é escolhida como a prioridade mais alta para garantir uma entrada suave no mercado e uma ótima experiência do usuário desde o lançamento.
SISTEMA DE RECOMENDAÇÃO DE PRODUTOS BASEADO EM IA	Embora o valor de negócio seja considerado médio, o risco técnico moderado torna esta funcionalidade uma segunda prioridade, fornecendo valor adicional aos vendedores sem um risco excessivo.
FERRAMENTA DE ANÁLISE DE DADOS DE VENDAS	Apesar de seu potencial para aumentar as vendas, o alto risco associado a complexidades técnicas e questões de privacidade faz com que seja priorizado por último. A equipe pode optar por revisitar esta funcionalidade após avaliar o feedback do usuário e considerar abordagens para mitigar os riscos.

QUESTÕES COMENTADAS

1. **(PROF. DIEGO / INÉDITA – 2024)** O Método MoSCoW pode ser usado para identificar funcionalidades que são essenciais para o lançamento inicial de um produto.

Comentários:

O Método MoSCoW é eficazmente utilizado para identificar funcionalidades que são essenciais para o lançamento inicial de um produto, categorizando-as como "Must Have". Este processo ajuda a garantir que o foco esteja nas funcionalidades críticas que precisam estar presentes para que o produto seja viável e atenda aos requisitos básicos dos usuários e do negócio.

Gabarito: Correto

---

2. **(PROF. DIEGO / INÉDITA – 2024)** Itens classificados como "Should Have" no Método MoSCoW são essenciais para o sucesso do projeto e não podem ser omitidos.

Comentários:

No Método MoSCoW, itens classificados como "Should Have" são importantes para o projeto, mas não são essenciais no mesmo sentido que os itens "Must Have". Os itens "Should Have" são aqueles que deveriam ser incluídos se possível, mas podem ser omitidos se necessário, sem inviabilizar o projeto. Eles são priorizados abaixo dos "Must Have", que são críticos e essenciais para o sucesso do projeto. Portanto, embora importantes, os itens "Should Have" podem ser adiados ou omitidos se restrições de tempo, recursos ou outras prioridades mais críticas surgirem.

Gabarito: Errado

---

3. **(PROF. DIEGO / INÉDITA – 2024)** No Scorecard para priorização de backlog, a atribuição de pontuações é feita sem considerar critérios pré-definidos.

Comentários:

No Scorecard para priorização de backlog, a atribuição de pontuações é feita justamente considerando critérios pré-definidos. Cada item do backlog é avaliado e pontuado com base em um conjunto de critérios estabelecidos pela equipe, que podem incluir fatores como valor de negócio, complexidade técnica, impacto no usuário, e urgência, entre outros. Essa abordagem estruturada e baseada em critérios é essencial para garantir avaliações objetivas e consistentes dos itens do backlog.

Gabarito: Errado

---

4. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de Scorecard permite comparações objetivas entre diferentes tarefas ao atribuir valores numéricos a cada item do backlog.

Comentários:

A técnica de Scorecard permite comparações objetivas entre diferentes tarefas ao atribuir valores numéricos a cada item do backlog com base em critérios previamente definidos. Essa abordagem quantitativa ajuda as equipes a avaliar e priorizar itens de forma sistemática e transparente, facilitando a tomada de decisões informadas sobre quais tarefas devem ser abordadas primeiro.

Gabarito: Correto

---

5. **(PROF. DIEGO / INÉDITA – 2024)** No BUC, itens com maior urgência sempre recebem a prioridade mais alta, independentemente de seu benefício ou custo.

Comentários:

No método BUC (Benefício, Urgência, Custo), itens com maior urgência não recebem automaticamente a prioridade mais alta, independentemente de seu benefício ou custo. A priorização é determinada pela avaliação balanceada de todos os três critérios. Embora a urgência seja um fator importante, ela é considerada em conjunto com o benefício que o item traz para o negócio e o custo ou recursos necessários para sua implementação. Itens que oferecem um alto benefício com urgência moderada e custo razoável podem, em muitos casos, ser priorizados sobre itens urgentes, mas com benefício limitado ou custo proibitivo.

Gabarito: Errado

---

6. **(PROF. DIEGO / INÉDITA – 2024)** O critério de Benefício no BUC avalia o impacto positivo que a implementação de um item do backlog trará para o projeto.

Comentários:

O critério de Benefício no método BUC efetivamente avalia o impacto positivo que a implementação de um item do backlog trará para o projeto, considerando aspectos como a contribuição para os objetivos do negócio, a melhoria na experiência do usuário, o aumento na eficiência operacional, entre outros. Esse critério ajuda a identificar quais funcionalidades ou tarefas oferecem o maior valor agregado, facilitando a priorização com base no potencial de retorno sobre o investimento.

Gabarito: Correto

---

7. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de Testes de Suposição foca exclusivamente na viabilidade técnica das funcionalidades, ignorando seu valor para os usuários.

Comentários:

A técnica de Testes de Suposição não se foca exclusivamente na viabilidade técnica das funcionalidades; ela também considera seu valor para os usuários, entre outros fatores críticos. O objetivo dos Testes de Suposição é validar hipóteses abrangentes sobre o projeto, que podem incluir suposições sobre a demanda do usuário, o impacto no mercado, além da viabilidade técnica. Portanto, essa técnica busca uma compreensão holística que engloba tanto aspectos técnicos quanto o valor oferecido aos usuários.

Gabarito: Errado

---

8. **(PROF. DIEGO / INÉDITA – 2024)** Testes de Suposição priorizam funcionalidades para desenvolvimento com base na necessidade de validar hipóteses críticas para o sucesso do projeto.

Comentários:

Os Testes de Suposição priorizam funcionalidades para desenvolvimento justamente com base na necessidade de validar hipóteses críticas para o sucesso do projeto. Essa abordagem permite que as equipes identifiquem e testem suposições fundamentais o mais cedo possível, focando no desenvolvimento de funcionalidades que necessitam de validação para garantir que o projeto esteja no caminho certo e alinhado com as necessidades reais dos usuários e objetivos do negócio.

Gabarito: Correto

---

9. **(PROF. DIEGO / INÉDITA – 2024)** No Método MoSCoW, "Won't Have" indica itens que são imperativos para a entrega atual do projeto.

Comentários:

No Método MoSCoW, "Won't Have" refere-se a itens que não são considerados necessários para a entrega atual do projeto. Esses itens são identificados como de menor prioridade ou como melhorias que podem ser adiadas para versões futuras do projeto. Portanto, ao contrário da afirmação, "Won't Have" indica itens que não são imperativos para a entrega atual.

Gabarito: Errado

---

10. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de Valor de Negócio x Risco ajuda a balancear o potencial de retorno com os desafios associados à implementação de itens do backlog.

Comentários:

A técnica de "Valor de Negócio x Risco" efetivamente ajuda a balancear o potencial de retorno com os desafios associados à implementação de itens do backlog. Ao avaliar itens com base no valor que proporcionam ao negócio contra os riscos de sua implementação, as equipes podem tomar decisões mais informadas sobre quais itens priorizar. Isso assegura que os recursos sejam alocados de maneira eficiente, maximizando o retorno sobre o investimento (ROI) enquanto se gerenciam os riscos de forma proativa.

Gabarito: Correto

---

11. (PROF. DIEGO / INÉDITA – 2024) Itens "Could Have" no Método MoSCoW são tratados como prioritários sobre os itens "Should Have".

Comentários:

No Método MoSCoW, itens "Could Have" não são tratados como prioritários sobre os itens "Should Have". Na verdade, a prioridade é justamente o oposto: os itens "Should Have" são considerados mais importantes e prioritários do que os itens "Could Have". A hierarquia de prioridades no Método MoSCoW segue a ordem de Must Have (Deve Ter), Should Have (Deveria Ter), Could Have (Poderia Ter), e Won't Have (Não Terá Desta Vez), com os itens "Should Have" tendo prioridade sobre os "Could Have".

Gabarito: Errado

---

12. (PROF. DIEGO / INÉDITA – 2024) A avaliação do risco no BUC inclui considerar dependências de outras tarefas ou funcionalidades.

Comentários:

Na avaliação do risco no Método BUC (Benefício, Urgência, Custo), de fato inclui-se a consideração de dependências de outras tarefas ou funcionalidades. Essas dependências podem impactar significativamente a execução de um item do backlog, afetando tanto o seu custo quanto a sua viabilidade de implementação dentro do cronograma projetado. Reconhecer e avaliar essas dependências é crucial para uma gestão de risco eficaz e para a priorização informada dos itens do backlog.

Gabarito: Correto

---

13. (PROF. DIEGO / INÉDITA – 2024) No Scorecard, o cálculo da pontuação total de cada item é feito sem levar em conta os pesos atribuídos aos critérios.

Comentários:

No Scorecard, o cálculo da pontuação total de cada item geralmente leva em conta os pesos atribuídos aos critérios. A ponderação permite que a equipe de projeto reflita a importância relativa de cada critério de acordo com os objetivos estratégicos do projeto ou as prioridades da organização. Ao aplicar pesos diferenciados, a equipe pode garantir que os critérios mais críticos tenham um impacto proporcionalmente maior na pontuação total, facilitando uma priorização mais alinhada com os valores e necessidades do negócio.

Gabarito: Errado

---

14. (PROF. DIEGO / INÉDITA – 2024) A técnica de Testes de Suposição ajuda a equipe a aprender e adaptar-se rapidamente, validando suposições importantes por meio de experimentos.

Comentários:

A técnica de Testes de Suposição de fato ajuda a equipe a aprender e adaptar-se rapidamente, pois foca na validação de suposições importantes através de experimentos. Este método permite identificar rapidamente quais aspectos do projeto ou produto necessitam de ajustes, reduzindo assim o tempo e os recursos desperdiçados em direções que talvez não sejam viáveis ou valiosas. Ao validar suposições críticas desde cedo, as equipes podem fazer iterações em suas abordagens com base em dados reais e feedback do usuário, promovendo uma cultura de aprendizado contínuo e adaptação ágil.

Gabarito: Correto

---

15. (PROF. DIEGO / INÉDITA – 2024) Itens de alto valor e baixo risco são considerados menos prioritários na técnica de Valor de Negócio x Risco.

Comentários:

Na técnica de "Valor de Negócio x Risco", itens de alto valor e baixo risco são considerados mais prioritários, não menos. Esta abordagem prioriza itens que oferecem o maior benefício para o negócio com o menor risco associado, maximizando assim o retorno sobre o investimento e minimizando potenciais obstáculos para a implementação. Portanto, itens de alto valor e baixo risco são vistos como oportunidades ideais para alcançar impactos positivos significativos no projeto ou no negócio.

Gabarito: Errado

---

16. (PROF. DIEGO / INÉDITA – 2024) Utilizar o Método MoSCoW ajuda na comunicação com as partes interessadas, oferecendo uma visão clara do que esperar na entrega do projeto.

Comentários:

Utilizar o Método MoSCoW de fato ajuda na comunicação com as partes interessadas, oferecendo uma visão clara do que esperar na entrega do projeto. Ao categorizar as tarefas em Must have, Should have, Could have e Won't have, a equipe de projeto pode comunicar eficazmente as prioridades e estabelecer expectativas realistas sobre os recursos e funcionalidades que serão entregues. Isso facilita o alinhamento entre a equipe de projeto e as partes interessadas, ajudando a gerenciar expectativas e a focar nos aspectos mais críticos do projeto.

Gabarito: Correto

---

17. (PROF. DIEGO / INÉDITA – 2024) Qual das seguintes afirmações melhor descreve a categoria "Must Have" no Método MoSCoW?

a) Funcionalidades que podem ser adiadas para versões futuras do projeto.
b) Funcionalidades que agregam valor significativo, mas não são essenciais.
c) Funcionalidades essenciais sem as quais o projeto não pode ser considerado completo.
d) Funcionalidades que não serão incluídas na entrega atual do projeto.
e) Funcionalidades desejáveis que serão incluídas se o tempo e os recursos permitirem.

Comentários:

(a) Correto. A categoria "Must Have" no Método MoSCoW refere-se a funcionalidades essenciais sem as quais o projeto não pode ser considerado completo. Estes requisitos são críticos para o lançamento do projeto e não podem ser adiados;

(b) Errado. Funcionalidades que podem ser adiadas para versões futuras do projeto são tipicamente classificadas como "Should Have" ou "Could Have", não "Must Have";

(c) Errado. Funcionalidades que agregam valor significativo, mas não são essenciais, geralmente se enquadram nas categorias "Should Have" ou "Could Have";

(d) Errado. Funcionalidades que não serão incluídas na entrega atual do projeto são classificadas como "Won't Have this time";

(e) Errado. Funcionalidades desejáveis que serão incluídas se o tempo e os recursos permitirem são classificadas como "Could Have".

Gabarito: Letra A

---

18. (PROF. DIEGO / INÉDITA – 2024) No contexto do Scorecard para priorização de backlog, como as pontuações são atribuídas a cada item?

a) Com base em preferências pessoais da equipe de desenvolvimento.
b) Através da comparação direta com itens semelhantes de outros projetos.

c) Avaliando cada item contra critérios definidos, com pontuações de acordo com o grau de atendimento.
d) Utilizando uma abordagem de consenso geral sem critérios específicos.
e) Atribuindo valores aleatórios e ajustando-os conforme o progresso do projeto.

Comentários:

(a) Errado. As pontuações no contexto do Scorecard para priorização de backlog não são baseadas em preferências pessoais da equipe de desenvolvimento, mas sim em critérios objetivos pré-definidos;

(b) Errado. Embora a comparação com itens semelhantes de outros projetos possa fornecer insights, as pontuações no Scorecard são atribuídas com base em critérios específicos para o projeto em questão, e não através de comparação direta

(c) Correto. No Scorecard para priorização de backlog, cada item é avaliado contra um conjunto de critérios definidos. Pontuações são atribuídas com base no grau em que cada item atende a esses critérios, possibilitando uma priorização objetiva e justa;

(d) Errado. A abordagem de consenso geral pode ser utilizada em algumas técnicas de priorização, mas no Scorecard, as pontuações são atribuídas com base em critérios específicos, não apenas em consenso geral sem critérios definidos;

(e) Errado. Valores aleatórios não são utilizados na metodologia do Scorecard. As pontuações são cuidadosamente atribuídas com base em como os itens atendem aos critérios estabelecidos, permitindo uma priorização mais objetiva e estratégica.

Gabarito: Letra C

---

19. **(PROF. DIEGO / INÉDITA – 2024)** Qual critério NÃO é comumente utilizado na técnica BUC para avaliação de itens do backlog?

a) Benefício para o cliente final.
b) Custo de implementação em termos de recursos.
c) Urgência baseada em tendências de mercado.
d) A cor preferida do cliente para a interface do usuário.
e) Risco associado à implementação do item.

Comentários:

(a) Errado. O benefício para o cliente final é um critério comumente utilizado na técnica BUC (Benefício, Urgência, Custo) para avaliar itens do backlog, pois foca em maximizar o valor entregue ao cliente;

(b) Errado. O custo de implementação em termos de recursos é um dos critérios fundamentais na técnica BUC, ajudando a determinar se o retorno justifica o investimento necessário;

(c) Errado. A urgência baseada em tendências de mercado é outro critério relevante na técnica BUC, uma vez que pode determinar a priorização de itens que aproveitam oportunidades de mercado ou que respondem a necessidades imediatas do negócio;

(d) Correto. A cor preferida do cliente para a interface do usuário não é um critério utilizado na técnica BUC para avaliação de itens do backlog. A técnica BUC foca em aspectos mais estratégicos e de alto nível, como benefício, urgência e custo, em vez de preferências específicas de design;

(e) Errado. O risco associado à implementação do item é considerado na técnica BUC, já que pode afetar diretamente o custo e a viabilidade de realizar o item do backlog, bem como seu potencial benefício.

Gabarito: Letra D

---

20. (PROF. DIEGO / INÉDITA – 2024) O que a técnica de Testes de Suposição visa validar em um projeto?

a) A capacidade técnica da equipe de desenvolvimento.
b) O orçamento total necessário para a conclusão do projeto.
c) As hipóteses subjacentes aos itens do backlog, especialmente em termos de valor e viabilidade.
d) O cronograma de entrega para todas as funcionalidades planejadas.
e) A preferência de cores e design entre os usuários finais.

Comentários:

(a) Errado. A técnica de Testes de Suposição não se concentra diretamente na avaliação da capacidade técnica da equipe de desenvolvimento, mas sim na validação das suposições feitas durante o planejamento do projeto;

(b) Errado. Embora o orçamento seja um componente crucial de qualquer projeto, os Testes de Suposição são mais focados em validar as hipóteses relacionadas aos aspectos do projeto, como valor para o cliente e viabilidade técnica, do que em estimar custos

(c) Correto. A técnica de Testes de Suposição é utilizada para validar as hipóteses subjacentes aos itens do backlog, especialmente em termos de valor para o cliente e viabilidade técnica ou de negócios. Isso ajuda a garantir que o projeto esteja no caminho certo para entregar valor real e mitigar riscos;

(d) Errado. O foco dos Testes de Suposição não está em validar o cronograma de entrega de todas as funcionalidades planejadas, mas sim em testar a validade das suposições feitas sobre o projeto, o que pode, indiretamente, afetar o cronograma;

(e) Errado. Preferências de cores e design entre os usuários finais podem ser importantes, mas os Testes de Suposição visam validar hipóteses mais críticas para o sucesso do projeto, como a necessidade do produto e sua viabilidade, não aspectos específicos de design.

Gabarito: Letra C

---

21. (PROF. DIEGO / INÉDITA – 2024) Na técnica de priorização de Valor de Negócio x Risco, como os itens são classificados?

a) Itens de alto risco são sempre priorizados, independentemente de seu valor de negócio.
b) Itens de baixo valor de negócio e alto risco são priorizados para maximizar o desafio.
c) Itens de baixo valor de negócio e baixo risco são os primeiros a serem desenvolvidos.
d) Itens são classificados aleatoriamente para incentivar a inovação.
e) Itens de alto valor de negócio e baixo risco são priorizados para entrega.

Comentários:

(a) Errado. A técnica de Valor de Negócio x Risco não prioriza itens de alto risco automaticamente; a priorização depende de uma avaliação conjunta do valor de negócio e do risco associado;

(b) Errado. Itens de baixo valor de negócio e alto risco geralmente não são priorizados, pois essa combinação não é ideal para maximizar o retorno sobre o investimento ou mitigar riscos significativos

(c) Errado. Itens de baixo valor de negócio e baixo risco geralmente não são os primeiros a serem desenvolvidos, pois essa abordagem não maximiza o valor para o negócio nem efetivamente utiliza recursos para mitigar riscos importantes;

(d) Errado. A classificação não é feita de maneira aleatória; ela é cuidadosamente determinada com base no valor de negócio e no risco de cada item para garantir uma tomada de decisão estratégica;

(e) Correto. Itens de alto valor de negócio e baixo risco são priorizados para entrega, pois essa combinação maximiza o retorno sobre o investimento e minimiza os riscos, alinhando-se efetivamente com os objetivos estratégicos do negócio.

Gabarito: Letra E

---

22. (PROF. DIEGO / INÉDITA – 2024) Qual é o objetivo principal do Método MoSCoW na priorização de requisitos de projeto?

a) Identificar quais requisitos são essenciais para o sucesso do projeto e quais podem ser adiados.
b) Eliminar completamente o risco associado a cada requisito do projeto.
c) Assegurar que todos os requisitos sejam tratados como igualmente importantes.
d) Determinar o orçamento necessário para a implementação de cada requisito.
e) Estabelecer uma sequência cronológica exata para o desenvolvimento de cada requisito.

Comentários:

(a) Correto. O Método MoSCoW é uma técnica de priorização usada para determinar a importância de cada requisito de projeto, identificando quais são essenciais (Must have) para o sucesso do projeto e quais podem ser adiados (Should have, Could have, Won't have this time). Este método ajuda a focar nos requisitos mais críticos;

(b) Errado. O Método MoSCoW não tem como objetivo eliminar completamente o risco, mas priorizar requisitos com base em sua importância e urgência, não sua associação ao risco;

(c) Errado. Este método especificamente não trata todos os requisitos como igualmente importantes; pelo contrário, classifica os requisitos em categorias de prioridade;

(d) Errado. Determinar o orçamento necessário para a implementação de cada requisito não é o objetivo principal do Método MoSCoW. Este método foca na priorização de requisitos, não no custo diretamente;

(e) Errado. Estabelecer uma sequência cronológica exata para o desenvolvimento de cada requisito não é o foco do Método MoSCoW. O método ajuda a definir a prioridade dos requisitos, mas não especifica a ordem exata de desenvolvimento.

Gabarito: Letra A

Lista de Questões

1. **(PROF. DIEGO / INÉDITA – 2024)** O Método MoSCoW pode ser usado para identificar funcionalidades que são essenciais para o lançamento inicial de um produto.

2. **(PROF. DIEGO / INÉDITA – 2024)** Itens classificados como "Should Have" no Método MoSCoW são essenciais para o sucesso do projeto e não podem ser omitidos.

3. **(PROF. DIEGO / INÉDITA – 2024)** No Scorecard para priorização de backlog, a atribuição de pontuações é feita sem considerar critérios pré-definidos.

4. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de Scorecard permite comparações objetivas entre diferentes tarefas ao atribuir valores numéricos a cada item do backlog.

5. **(PROF. DIEGO / INÉDITA – 2024)** No BUC, itens com maior urgência sempre recebem a prioridade mais alta, independentemente de seu benefício ou custo.

6. **(PROF. DIEGO / INÉDITA – 2024)** O critério de Benefício no BUC avalia o impacto positivo que a implementação de um item do backlog trará para o projeto.

7. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de Testes de Suposição foca exclusivamente na viabilidade técnica das funcionalidades, ignorando seu valor para os usuários.

8. **(PROF. DIEGO / INÉDITA – 2024)** Testes de Suposição priorizam funcionalidades para desenvolvimento com base na necessidade de validar hipóteses críticas para o sucesso do projeto.

9. **(PROF. DIEGO / INÉDITA – 2024)** No Método MoSCoW, "Won't Have" indica itens que são imperativos para a entrega atual do projeto.

10. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de Valor de Negócio x Risco ajuda a balancear o potencial de retorno com os desafios associados à implementação de itens do backlog.

11. **(PROF. DIEGO / INÉDITA – 2024)** Itens "Could Have" no Método MoSCoW são tratados como prioritários sobre os itens "Should Have".

12. **(PROF. DIEGO / INÉDITA – 2024)** A avaliação do risco no BUC inclui considerar dependências de outras tarefas ou funcionalidades.

13. **(PROF. DIEGO / INÉDITA – 2024)** No Scorecard, o cálculo da pontuação total de cada item é feito sem levar em conta os pesos atribuídos aos critérios.

14. **(PROF. DIEGO / INÉDITA – 2024)** A técnica de Testes de Suposição ajuda a equipe a aprender e adaptar-se rapidamente, validando suposições importantes por meio de experimentos.

15. **(PROF. DIEGO / INÉDITA – 2024)** Itens de alto valor e baixo risco são considerados menos prioritários na técnica de Valor de Negócio x Risco.

16. **(PROF. DIEGO / INÉDITA – 2024)** Utilizar o Método MoSCoW ajuda na comunicação com as partes interessadas, oferecendo uma visão clara do que esperar na entrega do projeto.

17. **(PROF. DIEGO / INÉDITA – 2024)** Qual das seguintes afirmações melhor descreve a categoria "Must Have" no Método MoSCoW?

a) Funcionalidades que podem ser adiadas para versões futuras do projeto.
b) Funcionalidades que agregam valor significativo, mas não são essenciais.
c) Funcionalidades essenciais sem as quais o projeto não pode ser considerado completo.
d) Funcionalidades que não serão incluídas na entrega atual do projeto.
e) Funcionalidades desejáveis que serão incluídas se o tempo e os recursos permitirem.

18. **(PROF. DIEGO / INÉDITA – 2024)** No contexto do Scorecard para priorização de backlog, como as pontuações são atribuídas a cada item?

a) Com base em preferências pessoais da equipe de desenvolvimento.
b) Através da comparação direta com itens semelhantes de outros projetos.
c) Avaliando cada item contra critérios definidos, com pontuações de acordo com o grau de atendimento.
d) Utilizando uma abordagem de consenso geral sem critérios específicos.
e) Atribuindo valores aleatórios e ajustando-os conforme o progresso do projeto.

19. **(PROF. DIEGO / INÉDITA – 2024)** Qual critério NÃO é comumente utilizado na técnica BUC para avaliação de itens do backlog?

a) Benefício para o cliente final.
b) Custo de implementação em termos de recursos.
c) Urgência baseada em tendências de mercado.
d) A cor preferida do cliente para a interface do usuário.
e) Risco associado à implementação do item.

20. **(PROF. DIEGO / INÉDITA – 2024)** O que a técnica de Testes de Suposição visa validar em um projeto?

a) A capacidade técnica da equipe de desenvolvimento.
b) O orçamento total necessário para a conclusão do projeto.
c) As hipóteses subjacentes aos itens do backlog, especialmente em termos de valor e viabilidade.

d) O cronograma de entrega para todas as funcionalidades planejadas.
e) A preferência de cores e design entre os usuários finais.

21. **(PROF. DIEGO / INÉDITA – 2024)** Na técnica de priorização de Valor de Negócio x Risco, como os itens são classificados?

a) Itens de alto risco são sempre priorizados, independentemente de seu valor de negócio.
b) Itens de baixo valor de negócio e alto risco são priorizados para maximizar o desafio.
c) Itens de baixo valor de negócio e baixo risco são os primeiros a serem desenvolvidos.
d) Itens são classificados aleatoriamente para incentivar a inovação.
e) Itens de alto valor de negócio e baixo risco são priorizados para entrega.

22. **(PROF. DIEGO / INÉDITA – 2024)** Qual é o objetivo principal do Método MoSCoW na priorização de requisitos de projeto?

a) Identificar quais requisitos são essenciais para o sucesso do projeto e quais podem ser adiados.
b) Eliminar completamente o risco associado a cada requisito do projeto.
c) Assegurar que todos os requisitos sejam tratados como igualmente importantes.
d) Determinar o orçamento necessário para a implementação de cada requisito.
e) Estabelecer uma sequência cronológica exata para o desenvolvimento de cada requisito.

GABARITO

1. CORRETO
2. ERRADO
3. ERRADO
4. CORRETO
5. ERRADO
6. CORRETO
7. ERRADO
8. CORRETO
9. ERRADO
10. CORRETO
11. ERRADO
12. CORRETO
13. ERRADO
14. CORRETO
15. ERRADO
16. CORRETO
17. LETRA A
18. LETRA C
19. LETRA D
20. LETRA C
21. LETRA E
22. LETRA A

TÉCNICAS DE COLABORAÇÃO DE TIMES ÁGEIS

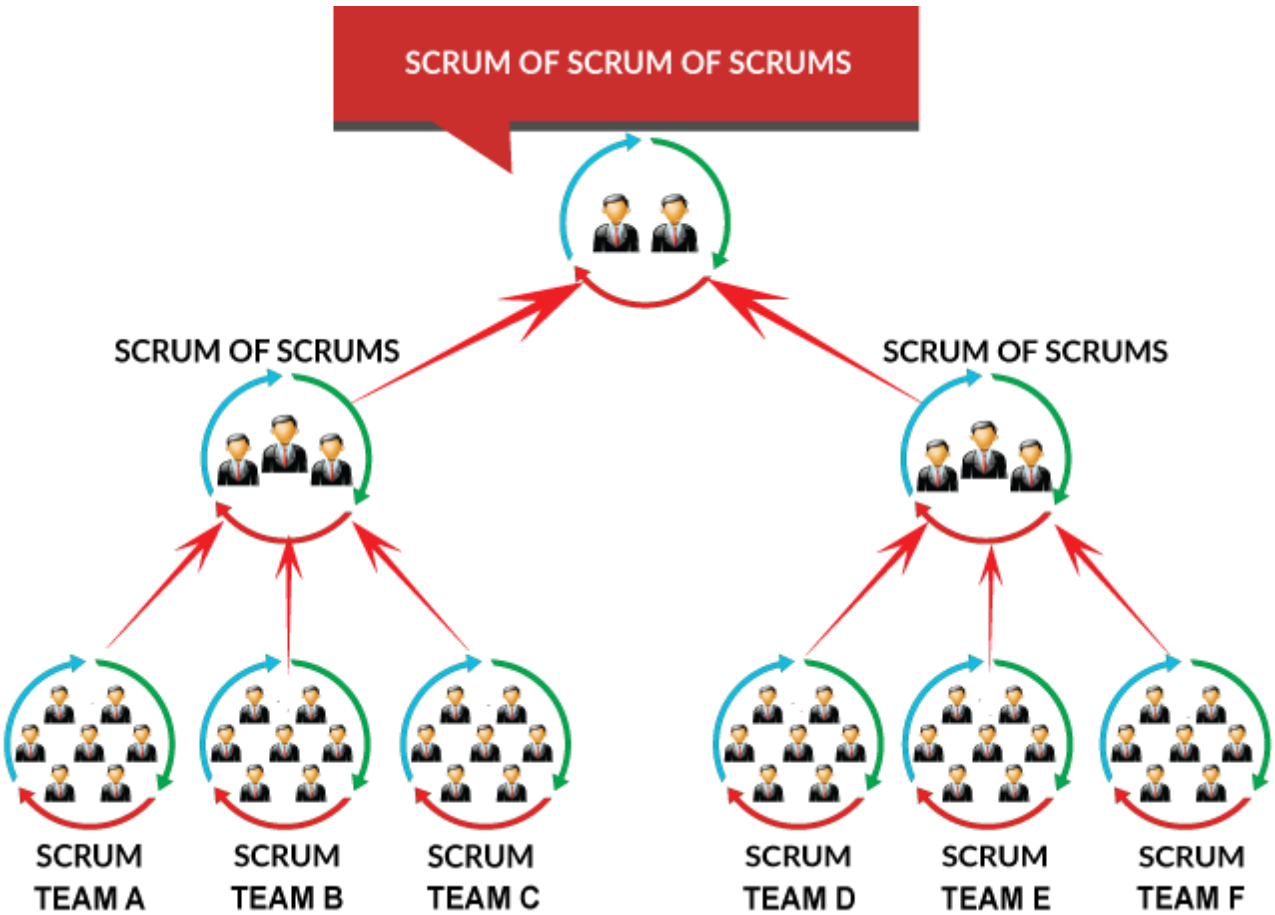
Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXA

Em ambientes ágeis, a colaboração e integração de equipes são fundamentais para o sucesso do projeto. Essas práticas são suportadas por várias técnicas e cerimônias ágeis projetadas para promover comunicação eficaz, alinhamento e entrega contínua de valor. Elas são projetadas para melhorar a colaboração entre os membros da equipe e outras partes interessadas, otimizar o processo de desenvolvimento e garantir que a equipe possa responder rapidamente às mudanças.

Scrum of Scrums (SoS)

INCIDÊNCIA EM PROVA: BAIXA



Trata-se de uma técnica projetada para coordenar o trabalho entre múltiplas equipes Scrum que trabalham no mesmo projeto, programa ou produto. À medida que organizações e projetos crescem, a necessidade de sincronização e comunicação entre equipes também aumenta. O SoS atende a essa necessidade, permitindo que representantes de cada equipe se reúnam para discutir progressos, desafios e dependências. Vejamos seus objetivos, características e benefícios:

OBJETIVOS	DESCRIÇÃO
COORDENAÇÃO	Garantir que as equipes estejam alinhadas em termos de objetivos, prazos e entregáveis.
COMUNICAÇÃO	Facilitar a comunicação entre equipes para identificar e resolver impedimentos que possam afetar várias equipes.
IDENTIFICAÇÃO DE DEPENDÊNCIAS	Discutir e gerenciar dependências entre equipes para evitar atrasos e bloqueios.
COMPARTILHAMENTO DE MELHORES PRÁTICAS	Proporcionar um fórum para compartilhar aprendizados e melhores práticas entre as equipes.

CARACTERÍSTICAS	DESCRIÇÃO
FREQUÊNCIA	O SoS geralmente ocorre regularmente, muitas vezes diariamente ou algumas vezes por semana, dependendo da complexidade e das necessidades do projeto.
PARTICIPANTES	Cada equipe Scrum escolhe um representante (geralmente o Scrum Master ou um membro da equipe técnica) para participar do SoS. Este representante é responsável por comunicar as atualizações, desafios e dependências de sua equipe.
FORMATO	As reuniões são mantidas curtas e focadas, seguindo um formato semelhante ao da Daily Meeting, onde cada representante discute o progresso de sua equipe, impedimentos que possam afetar outras equipes e como esses impedimentos serão resolvidos.
DOCUMENTAÇÃO	As discussões e acordos do SoS são documentados e compartilhados com todas as equipes envolvidas, garantindo transparência e comunicação clara.

BENEFÍCIOS	DESCRIÇÃO
MELHORIA DA SINCRONIZAÇÃO	O SoS melhora a coordenação entre equipes, ajudando a sincronizar entregas e marcos importantes.
RESOLUÇÃO DE IMPEDIMENTOS	Facilita a identificação e resolução rápida de impedimentos que afetam múltiplas equipes.
OTIMIZAÇÃO DA COLABORAÇÃO	Promove uma cultura de colaboração e suporte mútuo entre equipes, incentivando a solução conjunta de problemas.

Escolher o representante certo para o Scrum of Scrums é crucial, uma vez que essa pessoa deve ter uma boa visão geral do trabalho da equipe e ser capaz de se comunicar eficazmente com outras equipes. Além disso, manter o Scrum of Scrums focado e dentro do tempo programado pode ser extremamente desafiador, especialmente com múltiplas equipes discutindo diversos tópicos.

O SoS é valioso para projetos ágeis em grande escala, fornecendo um mecanismo eficaz para garantir que equipes diferentes permaneçam alinhadas ao longo do ciclo de vida do projeto.

EXEMPLO CONCRETO

Imaginemos uma empresa de desenvolvimento de software que está trabalhando em um grande projeto de plataforma de comércio eletrônico. O projeto é complexo e envolve várias equipes trabalhando em diferentes componentes do sistema, como interface do usuário, back-end, sistema de pagamento e logística. Para coordenar efetivamente o trabalho entre essas equipes e garantir que o projeto avance de forma coesa, a empresa adota a prática do Scrum of Scrums.

Vejamos a configuração do projeto:

- **Equipe de Interface do Usuário:** responsável pelo desenvolvimento da frente da loja, incluindo design e experiência do usuário.
- **Equipe de Backend:** foca na lógica de negócios, processamento de dados e Aplication Programming Intercace (APIs).
- **Equipe de Sistema de Pagamento:** trabalha na integração de gateways de pagamento e na segurança das transações.
- **Equipe de Logística:** desenvolve soluções para gerenciamento de estoque, envio e rastreamento de pedidos.

Implementação do Scrum of Scrums:

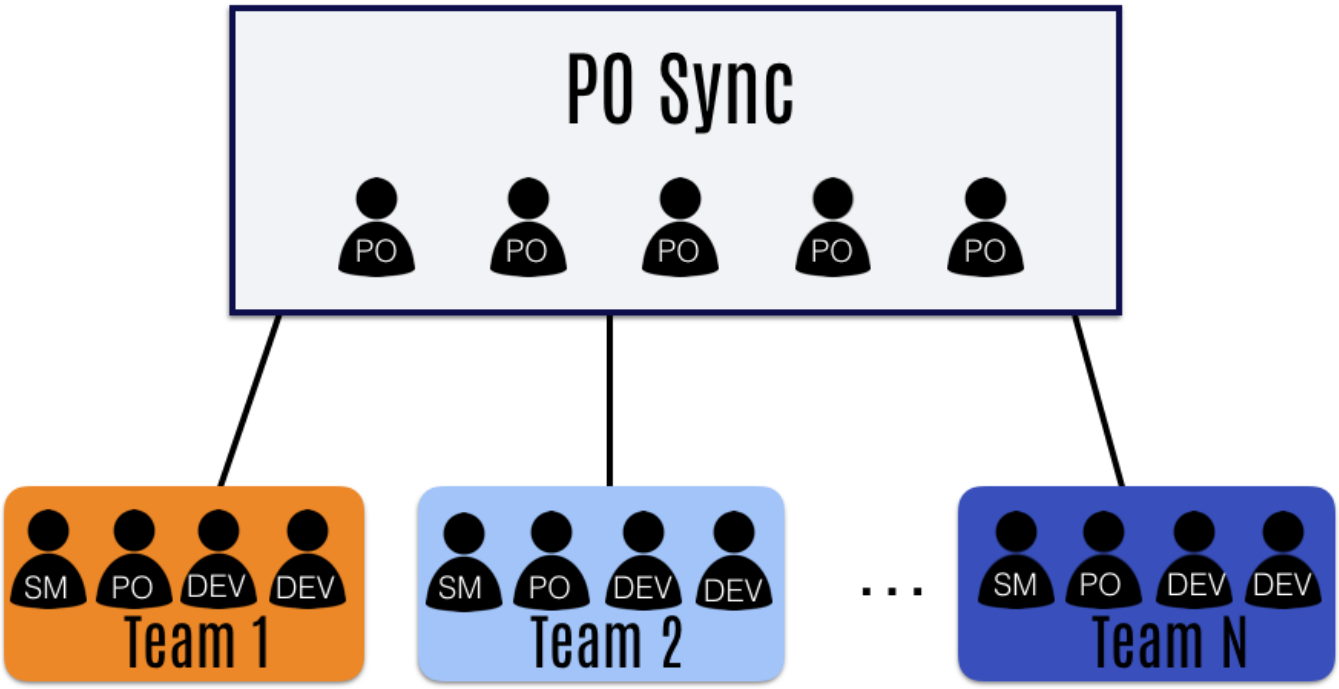
- **Representantes:** Cada equipe seleciona um representante, geralmente o Scrum Master ou um membro técnico com visão geral do trabalho da equipe, para participar do Scrum of Scrums.
- **Frequência das Reuniões:** O Scrum of Scrums ocorre três vezes por semana, permitindo atualizações regulares sem sobrecarregar as equipes com reuniões diárias.
- **Agenda da Reunião:**
    - **Atualizações Rápidas:** cada representante fornece uma atualização sobre o progresso da sua equipe, destacando conquistas e entregas completadas;
    - **Identificação de Dependências:** representantes discutem dependências entre as equipes. Por exemplo, a Equipe de Backend precisa concluir uma API específica para que a Equipe de Sistema de Pagamento possa avançar com sua integração;
    - **Resolução de Impediementos:** questões ou bloqueios que afetam múltiplas equipes são discutidos. Um impedimento na integração do gateway de pagamento que afeta tanto a Equipe de Sistema de Pagamento quanto a Equipe de Logística é identificado e discutido.

- **Planejamento de Ação Conjunta:** as equipes coordenam esforços para resolver os impedimentos identificados e planejam ações conjuntas para as dependências destacadas.

Dito isso, a prática do Scrum of Scrums melhora significativamente a comunicação entre as equipes, assegurando que todos estejam alinhados com os objetivos gerais do projeto. Ademais, Impedimentos que afetam múltiplas equipes são rapidamente trazidos à tona e resolvidos, minimizando atrasos no projeto. Por fim, as dependências críticas entre as tarefas das equipes são gerenciadas de forma proativa, garantindo um fluxo de trabalho suave e contínuo.

PoSync

INCIDÊNCIA EM PROVA: BAIXA



A Product Owner Sync Meeting é uma cerimônia de colaboração em ambientes ágeis que envolve a reunião regular dos Product Owners (POs) de diferentes equipes ou projetos com o propósito de garantir que todos os POs estejam alinhados em relação à visão geral do produto, estratégias, prioridades e roadmaps. **Essa sincronização é crucial em organizações que trabalham com múltiplos produtos/projetos interdependentes.** Seus objetivos são:

OBJETIVOS	DESCRIÇÃO
ALINHAMENTO ESTRATÉGICO	Assegurar que todos os POs estejam alinhados com a visão e os objetivos de negócios da organização.
COORDENAÇÃO DE ROADMAPS	Discutir e coordenar os roadmaps dos produtos para identificar e resolver sobreposições e lacunas, garantindo uma entrega coesa e otimizada.
GERENCIAMENTO DE DEPENDÊNCIAS	Identificar e planejar em torno de dependências cruzadas entre equipes, facilitando a colaboração e minimizando bloqueios.
PRIORIZAÇÃO E PLANEJAMENTO CONTÍNUO	Alinhar sobre a priorização de funcionalidades e recursos, especialmente quando impactam ou são compartilhados entre diferentes produtos ou projetos.
COMPARTILHAMENTO DE CONHECIMENTO	Trocar informações sobre desafios, sucessos, aprendizados e melhores práticas.

A frequência das reuniões pode variar de acordo com as necessidades do projeto ou da organização, podendo ser semanal, quinzenal ou mensal. A duração também é flexível, mas

geralmente é mantida curta para garantir foco e eficiência. Além dos Product Owners, às vezes, membros sêniores da gestão, Scrum Masters, ou outros stakeholders chave podem participar para fornecer insights ou tomar decisões estratégicas. A estrutura típica de uma reunião é:

- **Atualizações Rápidas:** cada PO oferece uma atualização curta sobre o progresso recente, desafios e sucessos do seu produto ou projeto.

- **Discussão de Dependências e Coordenação:** foco na identificação e resolução de dependências entre equipes, bem como na coordenação de esforços onde necessário.

- **Revisão de Roadmap e Prioridades:** avaliação conjunta dos roadmaps para garantir que estão alinhados e que as prioridades estão claramente definidas.

- **Resolução de Problemas:** discussão de problemas específicos que possam afetar múltiplos produtos ou equipes e desenvolvimento de planos de ação.

Essa cerimônia promove uma compreensão compartilhada da direção estratégica e dos objetivos de negócios entre os POs e facilita a tomada de decisão baseada em uma visão completa do portfólio de produtos. **Além disso, ela ajuda a evitar retrabalho e a otimizar o uso de recursos ao coordenar esforços entre equipes e permite alinhar estratégias, priorizar o trabalho e garantir que todos estejam se movendo na mesma direção para atingir os objetivos do negócio.**

EXEMPLO CONCRETO

Imagine uma empresa de software que está desenvolvendo uma suíte integrada de aplicativos de produtividade, incluindo um processador de texto, uma planilha e uma ferramenta de apresentação. O projeto é grande e dividido em várias equipes ágeis, cada uma responsável por um dos aplicativos, mas com o objetivo comum de garantir que os aplicativos funcionem de forma coesa dentro da suíte.

Para coordenar o trabalho entre as equipes e garantir a consistência do produto, a empresa implementa reuniões regulares de Product Owner Synchronization (PoSync). Vejamos o contexto:

- **Equipe do Processador de Texto:** essa equipe se foca no desenvolvimento de um processador de texto robusto e fácil de usar;
- **Equipe da Planilha:** essa equipe trabalha na criação de uma ferramenta de planilha versátil e poderosa;
- **Equipe da Ferramenta de Apresentação:** essa equipe dedica-se ao desenvolvimento de uma solução de apresentação intuitiva e rica em recursos;

Todas as equipes precisam garantir que seus aplicativos não apenas atendam às necessidades específicas dos usuários, mas também ofereçam uma experiência de usuário integrada e uniforme quando usados como parte da suíte de produtividade. As reuniões de PoSync ocorrem

a cada duas semanas, proporcionando um equilíbrio entre o trabalho focado e a necessidade de sincronização. Vejamos a estrutura típica da reunião:

- **Atualizações de Progresso:** cada Product Owner compartilha atualizações sobre o desenvolvimento de seu aplicativo, incluindo novos recursos implementados, desafios enfrentados e vitórias alcançadas;

- **Discussão de Interdependências:** Product Owners discutem dependências entre os aplicativos, como a necessidade de funcionalidades compartilhadas de edição ou formatos de arquivo comuns, e planejam como abordar essas questões de forma colaborativa;

- **Planejamento de Recursos Compartilhados:** decisões sobre a utilização de bibliotecas comuns, componentes de interface do usuário ou serviços de back-end são feitas para garantir consistência e eficiência;

- **Feedback e Melhorias:** compartilhamento de feedback dos usuários e discussão sobre como os insights podem ser utilizados para melhorar a suíte de produtos.

Dessa forma, a suíte de aplicativos oferece uma experiência de usuário consistente, com funcionalidades bem integradas entre os aplicativos. Há também uma redução de esforços duplicados e aproveitamento de componentes comuns, acelerando o desenvolvimento. A capacidade de responder rapidamente a feedbacks dos usuários e ajustar o planejamento do produto de acordo.

PI Planning

INCIDÊNCIA EM PROVA: BAIXA

O PI Planning, ou Planejamento Iterativo do Programa, é uma cerimônia central na metodologia Scaled Agile Framework (SAFe), que é uma abordagem para escalar práticas ágeis em grandes organizações. O PI Planning é um evento de dois dias (geralmente) que serve para alinhar todas as equipes de um Agile Release Train (ART) em torno dos objetivos do programa para a próxima Iteração do Programa (PI), que normalmente abrange um período de 8 a 12 semanas.

OBJETIVOS	DESCRIÇÃO
ALINHAMENTO	Assegurar que todas as equipes no ART estejam alinhadas com a visão do programa e os objetivos de negócios para o próximo PI.
PLANEJAMENTO COLABORATIVO	Facilitar o planejamento colaborativo entre as equipes para identificar dependências e coordenar entregas.
COMPROMISSO COM OBJETIVOS	Permitir que as equipes façam um compromisso coletivo com os objetivos do PI, baseando-se em sua capacidade e nas prioridades do negócio.

Dia 01:

- **Visão do Programa e Contexto:** os líderes do programa ou Product Managers apresentam a visão geral do programa, atualizações de mercado, e objetivos estratégicos para o próximo PI;
- **Apresentações das Equipes:** as equipes recebem informações detalhadas sobre as features e histórias de usuário planejadas para o PI, fornecidas pelos Product Owners;
- **Draft de Planejamento:** as equipes iniciam o planejamento de suas sprints, identificando dependências e discutindo potenciais desafios com outras equipes;

Dia 02:

- **Revisão e Ajuste de Planejamento:** as equipes continuam seu planejamento, ajustando conforme necessário com base em feedback e discussões;
- **Identificação de Riscos e Dependências:** as dependências entre as equipes são discutidas e registradas, e os riscos são identificados e categorizados;
- **Revisão de Planejamento e Comprometimento:** as equipes apresentam seus planos, incluindo objetivos e entregas esperadas, e riscos são discutidos e estratégias de mitigação são propostas;
- **Comprometimento das Equipes:** as equipes se comprometem com os objetivos do PI, considerando a capacidade e os recursos disponíveis;

ETAPAS	DESCRIÇÃO
PREPARAÇÃO	Antes do evento, as equipes devem se preparar revisando suas metas e objetivos individuais. Também é importante identificar quaisquer dependências externas e preparar uma lista de discussão para o evento.
REUNIÃO DE ABERTURA	O evento começa com uma reunião de abertura, onde a alta administração define a visão geral e as metas estratégicas. Isso estabelece o tom para o restante do planejamento.

PLANEJAMENTO DETALHADO	As equipes se reúnem para um planejamento detalhado, onde discutem suas metas específicas para o próximo Program Increment (PI). Elas também identificam e abordam possíveis problemas e riscos.
CERIMÔNIA DE PI PLANNING	Durante esta cerimônia, cada equipe apresenta seus planos e objetivos para todo o grupo. As dependências são identificadas e resolvidas, e as prioridades são estabelecidas para o próximo PI.
RETROSPECTIVA	Após o evento, as equipes realizam uma retrospectiva para revisar o PI Planning e identificar áreas de melhoria para o próximo ciclo.

BENEFÍCIOS	DESCRIÇÃO
VISIBILIDADE	O PI Planning oferece visibilidade clara dos planos de trabalho para todas as equipes, lideranças e stakeholders.
SINCRONIZAÇÃO	Facilita a sincronização das entregas entre as equipes, identificando e planejando em torno de dependências cruzadas.
ADAPTAÇÃO	Permite que a organização se adapte a mudanças no ambiente de negócios ou na estratégia de produto de forma ágil.
ENGAJAMENTO	Engaja as equipes no planejamento e na tomada de decisão, aumentando a sensação de propriedade e comprometimento com os objetivos do programa.
ALINHAMENTO ESTRATÉGICO	O PI Planning permite que todas as equipes trabalhem em direção aos mesmos objetivos estratégicos.

Sprint Planning

INCIDÊNCIA EM PROVA: BAIXA

O Sprint Planning é uma cerimônia essencial em metodologias ágeis, particularmente no Scrum, que marca o início de cada Sprint (ciclo de desenvolvimento). Esta reunião envolve toda a equipe Scrum, incluindo o Product Owner, o Scrum Master e o time de desenvolvimento. O objetivo principal é definir o que será entregue na próxima Sprint e como o trabalho será realizado. Vejamos suas características:

OBJETIVOS	DESCRIÇÃO
DEFINIR O OBJETIVO DA SPRINT	Um objetivo claro é estabelecido para a Sprint, orientando a equipe sobre o que precisa ser alcançado.
SELECIONAR ITENS DO BACKLOG	O Product Owner apresenta os itens de maior prioridade no Product Backlog. A equipe, então, seleciona os itens que podem ser concluídos durante a Sprint, baseando-se em sua capacidade e nas estimativas de esforço.
PLANEJAR O TRABALHO	A equipe de desenvolvimento discute e planeja como irá atingir o objetivo da Sprint, decompondo os itens do backlog selecionados em tarefas executáveis.

PARTICIPANTES	DESCRIÇÃO
PRODUCT OWNER	Responsável pela apresentação do Backlog, pelo esclarecimento de dúvidas e pela negociação e priorização.
EQUIPE DE DESENVOLVIMENTO	Responsável pela seleção de itens de Backlog, pelas estimativas e planejamento técnico, e pela definição de comprometimentos.
SCRUM MASTER	Responsável pela facilitação da reunião, pelo suporte à equipe de desenvolvimento e pela garantia da compreensão dos itens do Backlog e do objetivo da Sprint.

ESTRUTURA	DESCRIÇÃO
PARTE 1: O QUE SERÁ FEITO?	O Product Owner explica a prioridade dos itens no Product Backlog e o que ele gostaria que fosse realizado na Sprint. A equipe seleciona os itens do backlog que acredita ser capaz de completar durante a Sprint, considerando sua capacidade e as estimativas de esforço.
PARTE 2: COMO SERÁ FEITO?	Uma vez selecionados os itens do backlog para a Sprint, a equipe planeja como irá entregar esses itens, decompondo-os em tarefas mais detalhadas. Este planejamento pode incluir discussões técnicas detalhadas sobre a implementação e a identificação de qualquer dependência ou recurso necessário.

A duração é proporcional à duração da Sprint. Por exemplo: para uma Sprint de duas semanas, o Scrum Guide recomenda uma duração máxima de quatro horas para o planejamento.

BENEFÍCIOS	DESCRIÇÃO
CLARIDADE E FOCO	Estabelece um objetivo claro para a Sprint, ajudando a equipe a manter o foco nas tarefas mais importantes.

COMPROMETIMENTO DA EQUIPE	Ao envolver a equipe no processo de seleção e planejamento de tarefas, aumenta-se o comprometimento e a propriedade sobre o trabalho a ser realizado.
TRANSPARÊNCIA E COMUNICAÇÃO	Promove uma compreensão compartilhada dos objetivos, tarefas e desafios, melhorando a comunicação dentro da equipe.

Sprint Review

INCIDÊNCIA EM PROVA: BAIXA

O Sprint Review é uma cerimônia essencial no framework Scrum, realizada ao final de cada Sprint. Seu propósito é inspecionar o incremento do produto e adaptar o Product Backlog se necessário. Diferentemente da Retrospectiva da Sprint, que é focada na melhoria do processo e na performance da equipe, o Sprint Review concentra-se na revisão do trabalho concluído e na coleta de feedback para guiar os próximos passos do projeto. Vejamos suas características:

OBJETIVOS	DESCRIÇÃO
INSPEÇÃO DO INCREMENTO	Avaliar o trabalho que foi concluído durante a Sprint e determinar se atende aos critérios de aceitação definidos.
ADAPTAÇÃO DO BACKLOG DO PRODUTO	Com base no feedback recebido e no progresso feito, adaptar o Product Backlog para refletir as prioridades e direcionamentos estratégicos mais atuais.
COLABORAÇÃO E FEEDBACK	Envolver stakeholders, incluindo clientes e outros departamentos, no processo de revisão para coletar suas impressões, sugestões e necessidades.

PARTICIPANTES	DESCRIÇÃO
PRODUCT OWNER	Apresenta os itens do backlog que foram concluídos e explica o contexto atual do mercado e das metas de negócios.
EQUIPE DE DESENVOLVIMENTO	Demonstra o trabalho realizado e responde a perguntas técnicas ou relacionadas à implementação.
SCRUM MASTER	Facilita a reunião, assegurando que todos tenham oportunidade de contribuir.

ESTRUTURA	DESCRIÇÃO
APRESENTAÇÃO DO INCREMENTO	A equipe demonstra o que foi desenvolvido durante a Sprint, destacando as funcionalidades e melhorias implementadas.
DISCUSSÃO DO PROGRESSO	O Product Owner discute o estado atual do Product Backlog, incluindo o que mudou baseado no trabalho recente e no feedback externo.
PLANEJAMENTO DO PRÓXIMO PASSO	Discussão coletiva sobre as prioridades futuras e adaptações necessárias, tendo como base o feedback recebido durante a review e qualquer mudança no ambiente de negócios ou nas expectativas do usuário.

A duração recomendada para o Sprint Review é proporcional à duração da Sprint. Para uma Sprint de duas semanas, por exemplo, o guia Scrum sugere uma reunião de até duas horas.

BENEFÍCIOS	DESCRIÇÃO
TRANSPARÊNCIA	Proporciona uma visão clara do progresso do projeto para todos os envolvidos, aumentando a confiança e a clareza sobre o direcionamento do produto.

FEEDBACK EM TEMPO REAL	Permite que a equipe ajuste rapidamente suas prioridades e planos com base no feedback direto dos stakeholders, garantindo que o produto continue a atender às necessidades do mercado e dos usuários.
COLABORAÇÃO	Fortalece a colaboração entre a equipe de desenvolvimento, o Product Owner e os stakeholders, criando um ambiente propício para a inovação e melhoria contínua.

Sprint Retrospective

INCIDÊNCIA EM PROVA: BAIXA

A Sprint Retrospective é uma cerimônia do Scrum que ocorre ao final de cada Sprint para refletir sobre a Sprint que acaba de terminar, identificando o que funcionou bem, o que pode ser melhorado e como a equipe pode implementar essas melhorias em Sprints futuras. Diferentemente da Sprint Review, a Retrospectiva da Sprint foca na equipe e nos processos internos, visando à melhoria contínua. Vejamos suas características:

OBJETIVOS	DESCRIÇÃO
REFLEXÃO	Analisar a eficiência da equipe durante a Sprint, incluindo processos, ferramentas, comunicação e relações interpessoais.
IDENTIFICAÇÃO DE MELHORIAS	Identificar áreas específicas que funcionaram bem e aquelas que precisam de ajustes ou melhorias.
PLANEJAMENTO DE AÇÕES	Desenvolver um plano de ação para implementar melhorias identificadas, aumentando a eficácia e eficiência da equipe em Sprints futuras.

PARTICIPANTES	DESCRIÇÃO
EQUIPE DE DESENVOLVIMENTO	Participa ativamente, compartilhando insights e sugestões para melhorias.
SCRUM MASTER	Facilita a reunião, ajudando a manter o foco e garantindo que todos tenham a oportunidade de contribuir.
PRODUCT OWNER	Pode participar para oferecer perspectivas sobre o trabalho da equipe, embora o foco esteja principalmente nos processos internos da equipe.

ESTRUTURA	DESCRIÇÃO
REVISÃO DA ÚLTIMA RETROSPECTIVA	Começa-se revisando as ações de melhoria identificadas na última retrospectiva e avaliando se foram implementadas com sucesso.
COLETA DE DADOS E INSIGHTS	A equipe discute e documenta o que funcionou bem, o que não funcionou e quaisquer ideias para melhorias.
DEFINIÇÃO DE AÇÕES DE MELHORIA	A equipe prioriza os itens discutidos e seleciona as ações de melhoria mais críticas para serem implementadas na próxima Sprint.

A duração recomendada é proporcional à duração da Sprint, geralmente variando de 1,5 a 3 horas para Sprints de duas semanas.

BENEFÍCIOS	DESCRIÇÃO
MELHORIA CONTÍNUA	Promove um ciclo de feedback que leva à melhoria contínua dos processos da equipe e do bem-estar dos membros da equipe.
EMPODERAMENTO DA EQUIPE	Ao permitir que a equipe identifique e resolva seus próprios desafios, aumenta-se o senso de propriedade e responsabilidade.

COMUNICAÇÃO E COLABORAÇÃO	Fortalece a comunicação e a colaboração dentro da equipe, construindo um ambiente de trabalho mais aberto e confiável.

Daily Meeting

INCIDÊNCIA EM PROVA: BAIXA

A Daily Meeting, também conhecida como Daily Stand-up ou Daily Scrum, é uma técnica de colaboração rápida e eficaz amplamente utilizada por equipes ágeis, particularmente na metodologia Scrum. Esta reunião curta, realizada diariamente, tem como objetivo promover a comunicação transparente entre os membros da equipe, identificar rapidamente impedimentos e alinhar as atividades do dia para avançar em direção aos objetivos da Sprint.

OBJETIVOS	DESCRIÇÃO
SINCRONIZAÇÃO DIÁRIA	Permitir que cada membro da equipe compartilhe progressos e planos com o restante da equipe.
IDENTIFICAÇÃO DE IMPEDIMENTOS	Revelar rapidamente quaisquer obstáculos que possam impedir a equipe de alcançar seus objetivos, permitindo que o Scrum Master e a equipe trabalhem juntos para resolvê-los.
PROMOÇÃO DA COLABORAÇÃO	Encorajar a colaboração e o apoio mútuo entre os membros da equipe, especialmente se alguém estiver enfrentando desafios.

PARTICIPANTES	DESCRIÇÃO
EQUIPE DE DESENVOLVIMENTO	Cada membro compartilha suas atualizações, mantendo o foco nas três perguntas-chave.
SCRUM MASTER	Facilita a reunião, garantindo que permaneça dentro do tempo e focada. Também toma nota dos impedimentos mencionados para ajudar a resolvê-los após a reunião.
PRODUCT OWNER	Pode participar para ouvir as atualizações e oferecer esclarecimentos se necessário, mas o foco principal é a coordenação entre os membros da equipe de desenvolvimento.

ESTRUTURA	DESCRIÇÃO
PARTICIPANTES	Todos os membros da equipe de desenvolvimento, Scrum Master e, opcionalmente, o Product Owner.
FORMATO	Cada membro da equipe responde a três perguntas fundamentais: O que eu fiz ontem para ajudar a equipe a atingir Objetivo da Sprint? O que eu vou fazer hoje para ajudar a equipe a atingir o Objetivo da Sprint? Há algum impedimento que está me impedindo ou pode impedir a equipe de atingir o Objetivo da Sprint?

Dessa vez, a duração recomendada não é exatamente proporcional à duração da Sprint – geralmente dura cerca de 15 minutos.

BENEFÍCIOS	DESCRIÇÃO
COMUNICAÇÃO MELHORADA	Assegura que todos na equipe estejam informados sobre o progresso e os desafios diários.
ALINHAMENTO E FOCO	Mantém a equipe alinhada com o Objetivo da Sprint e focada nas tarefas mais críticas.

DETECÇÃO E RESOLUÇÃO RÁPIDA DE PROBLEMAS	Facilita a identificação e a resolução rápida de impedimentos, evitando atrasos no projeto.

QUESTÕES COMENTADAS

1. **(PROF. DIEGO / INÉDITA – 2024)** O Scrum of Scrums é uma cerimônia que facilita a comunicação entre várias equipes Scrum trabalhando no mesmo projeto.

Comentários:

O Scrum of Scrums facilita a coordenação e comunicação entre múltiplas equipes Scrum, promovendo alinhamento e sincronização em projetos grandes.

Gabarito: Correto

---

2. **(PROF. DIEGO / INÉDITA – 2024)** A principal finalidade da Product Owner Sync Meeting é resolver impedimentos técnicos que as equipes de desenvolvimento enfrentam.

Comentários:

A Product Owner Sync Meeting visa alinhar Product Owners em relação à visão, estratégias, e prioridades, e não diretamente resolver impedimentos técnicos.

Gabarito: Errado

---

3. **(PROF. DIEGO / INÉDITA – 2024)** O PI Planning é uma cerimônia exclusiva do framework Scrum.

Comentários:

O PI Planning é parte do Scaled Agile Framework (SAFe), não do Scrum, e serve para planejar atividades em grande escala, envolvendo várias equipes.

Gabarito: Errado

---

4. **(PROF. DIEGO / INÉDITA – 2024)** Durante o Sprint Planning, apenas o Product Owner define o que será trabalhado na próxima Sprint.

Comentários:

Durante o Sprint Planning, tanto o Product Owner quanto a equipe de desenvolvimento colaboram para definir o trabalho a ser feito na Sprint, incluindo como será realizado.

Gabarito: Errado

---

5. **(PROF. DIEGO / INÉDITA – 2024)** O Sprint Review foca na inspeção do incremento do produto e na coleta de feedback dos stakeholders.

Comentários:

O Sprint Review é dedicado a revisar o trabalho concluído e coletar feedback, visando adaptar o Product Backlog e melhorar entregas futuras.

Gabarito: Correto

---

6. **(PROF. DIEGO / INÉDITA – 2024)** A Sprint Retrospective destina-se a inspecionar o produto entregue durante a Sprint.

Comentários:

A Sprint Retrospective é focada na equipe e nos processos, buscando identificar o que foi bem e o que pode ser melhorado, e não na inspeção do produto.

Gabarito: Errado

---

7. **(PROF. DIEGO / INÉDITA – 2024)** A Daily Meeting deve durar no máximo 15 minutos, independentemente do tamanho da equipe.

Comentários:

A Daily Meeting, ou Daily Scrum, deve ser curta, não excedendo 15 minutos, para discutir progressos, planos e impedimentos, mantendo eficiência.

Gabarito: Correto

---

8. **(PROF. DIEGO / INÉDITA – 2024)** No Scrum of Scrums, apenas os Scrum Masters das equipes participam.

Comentários:

No Scrum of Scrums, representantes de cada equipe, que podem ser Scrum Masters ou outros membros, participam para discutir progresso e dependências.

Gabarito: Errado

---

9. **(PROF. DIEGO / INÉDITA – 2024)** A Product Owner Sync Meeting ajuda a sincronizar a visão do produto entre diferentes Product Owners.

Comentários:

A reunião de sincronização dos Product Owners ajuda a garantir que todos estejam trabalhando com a mesma visão de produto e estratégias alinhadas.

Gabarito: Correto

---

10. **(PROF. DIEGO / INÉDITA – 2024)** O PI Planning não inclui a identificação de riscos e dependências entre equipes.

Comentários:

O PI Planning inclui discussões sobre riscos e dependências entre equipes para garantir um planejamento coeso e alinhado para o Program Increment.

Gabarito: Errado

---

11. **(PROF. DIEGO / INÉDITA – 2024)** No Sprint Planning, a equipe de desenvolvimento ajuda a definir como o trabalho será realizado.

Comentários:

O Sprint Planning envolve a equipe de desenvolvimento no planejamento de como os itens do backlog serão entregues, promovendo comprometimento e clareza.

Gabarito: Correto

---

12. **(PROF. DIEGO / INÉDITA – 2024)** O Sprint Review é realizado antes do início de uma nova Sprint.

Comentários:

O Sprint Review acontece ao final de cada Sprint, e não antes de iniciar uma nova, servindo para avaliar o trabalho feito e obter feedback.

Gabarito: Errado

---

13. **(PROF. DIEGO / INÉDITA – 2024)** A Sprint Retrospective ajuda a equipe a planejar melhorias para o próximo ciclo de trabalho.

Comentários:

A Sprint Retrospective concentra-se em como a equipe pode melhorar seu trabalho e processos para a próxima Sprint, promovendo melhoria contínua.

Gabarito: Correto

---

14. (PROF. DIEGO / INÉDITA – 2024) A Daily Meeting é uma cerimônia fechada onde somente os membros da equipe de desenvolvimento podem falar.

Comentários:

A Daily Meeting é aberta para todos os membros da equipe, incluindo o Scrum Master e o Product Owner, para discutir progresso e impedimentos.

Gabarito: Errado

---

15. (PROF. DIEGO / INÉDITA – 2024) O Scrum of Scrums substitui a necessidade de Daily Meetings individuais em cada equipe.

Comentários:

O Scrum of Scrums não substitui as Daily Meetings individuais de cada equipe, mas serve como um complemento para coordenação entre equipes.

Gabarito: Errado

---

16. (PROF. DIEGO / INÉDITA – 2024) O objetivo do PI Planning é alinhar todas as equipes de um Agile Release Train para o próximo Program Increment.

Comentários:

O PI Planning alinha as equipes dentro de um Agile Release Train (ART) aos objetivos do próximo Program Increment, garantindo foco e direção unificados.

Gabarito: Correto

---

17. (PROF. DIEGO / INÉDITA – 2024) A Product Owner Sync Meeting ocorre diariamente para revisar o progresso de cada equipe.

Comentários:

A Product Owner Sync Meeting geralmente ocorre com menos frequência (semanal, quinzenal ou mensal) para alinhar estratégias e não diariamente.

Gabarito: Correto

---

18. (PROF. DIEGO / INÉDITA – 2024) No Sprint Planning, a equipe de desenvolvimento seleciona itens do Product Backlog que podem ser concluídos durante a Sprint.

Comentários:

A equipe de desenvolvimento, no Sprint Planning, seleciona itens do backlog que acredita poder concluir, baseando-se na capacidade e prioridades.

Gabarito: Correto

---

19. (PROF. DIEGO / INÉDITA – 2024) Durante o Sprint Review, são discutidas e planejadas as melhorias dos processos internos da equipe.

Comentários:

O Sprint Review foca na revisão do incremento do produto e no feedback dos stakeholders, diferentemente da retrospectiva que foca em melhorias de processo.

Gabarito: Errado

---

20. (PROF. DIEGO / INÉDITA – 2024) A Sprint Retrospective é realizada ao final de cada Sprint para refletir sobre a eficiência da equipe e identificar áreas de melhoria.

Comentários:

A Sprint Retrospective permite que a equipe reflita sobre a última Sprint, identificando sucessos e áreas para melhorias, visando à evolução contínua.

Gabarito: Correto

---

21. (PROF. DIEGO / INÉDITA – 2024) Qual é o objetivo principal do Scrum of Scrums?

a) Resolver problemas técnicos específicos da equipe de desenvolvimento.
b) Coordenar a publicação do produto final.
c) Garantir que as equipes estejam alinhadas em termos de objetivos e entregáveis.
d) Revisar e adaptar o Product Backlog individualmente para cada equipe.
e) Facilitar a integração de código entre as equipes.

Comentários:

(a) Errado. Resolver problemas técnicos específicos da equipe de desenvolvimento é geralmente o foco de reuniões de equipe individuais ou sessões de problem solving, não do Scrum of Scrums;

(b) Errado. Coordenar a publicação do produto final é uma responsabilidade mais ampla que pode envolver o Scrum of Scrums como um componente, mas não é seu objetivo principal;

(c) Correto. O objetivo principal do Scrum of Scrums é garantir que as equipes estejam alinhadas em termos de objetivos e entregáveis. Ele serve como uma reunião de coordenação para representantes de várias equipes Scrum, facilitando a comunicação e a resolução de impedimentos que afetam mais de uma equipe;

(d) Errado. Revisar e adaptar o Product Backlog é uma responsabilidade do Product Owner com feedback da equipe de desenvolvimento, e não um foco principal do Scrum of Scrums, que se destina mais à coordenação entre equipes;

(e) Errado. Facilitar a integração de código entre as equipes pode ser uma das questões abordadas no Scrum of Scrums, mas não é seu objetivo principal. O foco é mais amplo, incluindo a coordenação de tarefas, alinhamento de cronograma, e resolução de dependências entre equipes.

Gabarito: Letra C

---

22. (PROF. DIEGO / INÉDITA – 2024) Qual das seguintes opções não é um objetivo da Product Owner Sync Meeting?

a) Coordenar os roadmaps de produtos.
b) Discutir estratégias de marketing.
c) Gerenciar dependências cruzadas entre produtos.
d) Alinhar sobre a priorização de funcionalidades.
e) Compartilhar conhecimento sobre desafios e sucessos.

Comentários:

(a) Errado. Coordenar os roadmaps de produtos é um dos objetivos da Product Owner Sync Meeting, onde os Product Owners podem discutir e alinhar seus planos para garantir que os esforços estejam sincronizados entre os produtos;

(b) Correto. Discutir estratégias de marketing geralmente não é um objetivo da Product Owner Sync Meeting. Estas reuniões focam mais no alinhamento de aspectos relacionados ao desenvolvimento do produto, priorização de funcionalidades e gerenciamento de dependências, não em estratégias de marketing;

(c) Errado. Gerenciar dependências cruzadas entre produtos é definitivamente um objetivo da Product Owner Sync Meeting, permitindo que os Product Owners identifiquem e abordem quaisquer dependências ou conflitos entre as equipes de produtos;

(d) Errado. Alinhar sobre a priorização de funcionalidades é uma atividade central da Product Owner Sync Meeting, onde os Product Owners discutem a importância relativa de várias funcionalidades para garantir uma entrega coordenada e eficaz;

(e) Errado. Compartilhar conhecimento sobre desafios e sucessos é também um dos objetivos da Product Owner Sync Meeting, promovendo aprendizado e colaboração entre os Product Owners através do compartilhamento de experiências.

Gabarito: Letra B

---

23. (PROF. DIEGO / INÉDITA – 2024) O que caracteriza o PI Planning no contexto do SAFe?

a) Uma reunião diária para acompanhar o progresso do desenvolvimento.
b) Um evento de dois dias para planejamento e alinhamento de múltiplas equipes.
c) Uma retrospectiva de fim de sprint para avaliar o desempenho da equipe.
d) Uma revisão do sprint para apresentar o trabalho concluído aos stakeholders.
e) Uma cerimônia para celebrar o lançamento de novas funcionalidades.

Comentários:

(a) Errado. Uma reunião diária para acompanhar o progresso do desenvolvimento descreve mais adequadamente a Daily Stand-up ou Daily Scrum, não o PI Planning;

(b) Correto. O PI Planning é um evento de dois dias no contexto do SAFe (Scaled Agile Framework) destinado ao planejamento e alinhamento de múltiplas equipes que trabalham juntas em um Agile Release Train (ART). É um momento crítico para a definição de objetivos, alinhamento em torno de entregas e identificação de dependências;

(c) Errado. Uma retrospectiva de fim de sprint para avaliar o desempenho da equipe refere-se à cerimônia de Retrospectiva do Sprint, e não ao PI Planning;

(d) Errado. Uma revisão do sprint para apresentar o trabalho concluído aos stakeholders é conhecida como a Revisão do Sprint ou Sprint Review, não como PI Planning;

(e) Errado. Uma cerimônia para celebrar o lançamento de novas funcionalidades pode ser parte de várias práticas ágeis, mas não define especificamente o PI Planning no contexto do SAFe.

Gabarito: Letra B

---

24. (PROF. DIEGO / INÉDITA – 2024) Durante o Sprint Planning, o que a equipe de desenvolvimento faz?

a) Avalia o desempenho do último sprint.
b) Seleciona itens do Product Backlog que serão concluídos na próxima Sprint.
c) Apresenta o incremento do produto aos stakeholders.
d) Discute mudanças estratégicas com a alta gerência.
e) Planeja a festa de lançamento do produto.

Comentários:

(a) Errado. Avaliar o desempenho do último sprint é o propósito da Reunião de Retrospectiva do Sprint, não do Sprint Planning;

(b) Correto. Durante o Sprint Planning, a equipe de desenvolvimento, juntamente com o Product Owner, seleciona itens do Product Backlog que acredita ser capaz de completar durante a próxima Sprint, transformando-os em um Sprint Backlog. Este processo envolve o planejamento de como o trabalho será realizado;

(c) Errado. Apresentar o incremento do produto aos stakeholders é realizado durante a Sprint Review, não durante o Sprint Planning;

(d) Errado. Discussões sobre mudanças estratégicas com a alta gerência não são o foco do Sprint Planning. Este evento é focado em planejar o trabalho para a próxima Sprint;

(e) Errado. Planejar a festa de lançamento do produto não é parte do Sprint Planning. Este evento é dedicado ao planejamento do trabalho que será realizado na próxima Sprint.

Gabarito: Letra B

---

25. (PROF. DIEGO / INÉDITA – 2024) Qual é o foco principal do Sprint Review?

a) Identificar e resolver impedimentos técnicos.
b) Inspecionar o incremento do produto e adaptar o Product Backlog.
c) Discutir o desempenho individual dos membros da equipe.
d) Priorizar itens para o próximo Product Backlog.
e) Celebrar os sucessos do último sprint.

Comentários:

(a) Errado. Identificar e resolver impedimentos técnicos é frequentemente um foco da Daily Scrum e pode ser discutido durante o Sprint Retrospective, mas não é o foco principal do Sprint Review;

(b) Correto. O foco principal do Sprint Review é inspecionar o incremento do produto realizado durante o Sprint e, com base nessa inspeção, adaptar o Product Backlog se necessário. Este é um momento para a equipe, o Product Owner, e os stakeholders colaborarem e discutirem o progresso do projeto;

(c) Errado. Discutir o desempenho individual dos membros da equipe não é o objetivo do Sprint Review. Questões de desempenho individual são mais apropriadamente abordadas em outros contextos, como avaliações de desempenho ou one-on-ones com a gerência;

(d) Errado. Embora a priorização de itens para o próximo Product Backlog possa ser influenciada pelas discussões durante o Sprint Review, o foco principal desse evento não é a priorização, mas sim a revisão do que foi feito no Sprint atual;

(e) Errado. Celebrar os sucessos do último sprint pode ser parte do Sprint Review, mas não é seu foco principal. O evento é destinado mais a revisar o trabalho feito e a planejar os próximos passos com base nesse trabalho.

Gabarito: Letra B

---

26. (PROF. DIEGO / INÉDITA – 2024) O que não é discutido durante a Sprint Retrospective?

a) Como melhorar a eficiência da equipe.
b) A qualidade do código desenvolvido no último sprint.
c) Planos de ação para implementar melhorias identificadas.
d) O progresso do projeto em relação ao cronograma geral.
e) Estratégias de vendas para o produto desenvolvido.

Comentários:

(a) Errado. Como melhorar a eficiência da equipe é exatamente um dos focos da Sprint Retrospective, onde a equipe reflete sobre o último sprint para identificar e planejar maneiras de se tornar mais eficaz;

(b) Errado. A qualidade do código desenvolvido no último sprint pode ser discutida durante a Sprint Retrospective, especialmente se estiver relacionada a como a equipe pode melhorar seu trabalho no futuro;

(c) Errado. Planos de ação para implementar melhorias identificadas são uma parte crucial da Sprint Retrospective, ajudando a equipe a concretizar os passos a serem tomados para melhorar em sprints futuros;

(d) Errado. O progresso do projeto em relação ao cronograma geral pode ser considerado durante a Sprint Retrospective, especialmente se houver preocupações sobre como os processos da equipe estão afetando o cumprimento dos prazos;

(e) Correto. Estratégias de vendas para o produto desenvolvido não são discutidas durante a Sprint Retrospective. Este evento é focado em refletir sobre o processo da equipe de desenvolvimento, sua eficiência e colaboração, não sobre estratégias comerciais ou de vendas do produto.

Gabarito: Letra E

---

27. (PROF. DIEGO / INÉDITA – 2024) Quem participa da Daily Meeting?

a) Apenas o Scrum Master e o Product Owner.
b) Todos os membros da equipe de desenvolvimento.
c) Stakeholders externos e clientes.
d) A alta gerência da empresa.
e) Apenas os membros da equipe de desenvolvimento designados pelo Scrum Master.

Comentários:

(a) Errado. A Daily Meeting (ou Daily Scrum) não é limitada apenas ao Scrum Master e ao Product Owner. O foco principal está na equipe de desenvolvimento;

(b) Correto. Todos os membros da equipe de desenvolvimento participam da Daily Meeting (ou Daily Scrum). Este é um evento curto para que a equipe sincronize suas atividades e planeje o trabalho das próximas 24 horas;

(c) Errado. Stakeholders externos e clientes geralmente não participam da Daily Meeting. Este evento é destinado para a equipe de desenvolvimento sincronizar o progresso e planejar o trabalho diário;

(d) Errado. A alta gerência da empresa normalmente não participa da Daily Meeting, a menos que estejam diretamente envolvidos no desenvolvimento como parte da equipe;

(e) Errado. Todos os membros da equipe de desenvolvimento devem participar da Daily Meeting, não apenas aqueles designados pelo Scrum Master. O evento é para toda a equipe de desenvolvimento.

Gabarito: Letra B

---

28. (PROF. DIEGO / INÉDITA – 2024) Qual das seguintes não é uma característica da Daily Meeting?

a) Ser realizada no mesmo local e horário todos os dias.
b) Ter duração máxima de 15 minutos.
c) Discutir detalhadamente novas ideias de funcionalidades.
d) Focar no progresso feito desde a última reunião.
e) Identificar rapidamente os impedimentos.

Comentários:

(a) Errado. Ser realizada no mesmo local e horário todos os dias é uma característica recomendada para a Daily Meeting (ou Daily Scrum), pois ajuda a manter a consistência e facilita a rotina da equipe;

(b) Errado. Ter duração máxima de 15 minutos é uma das características definidoras da Daily Meeting, garantindo que a reunião seja rápida e focada;

(c) Correto. Discutir detalhadamente novas ideias de funcionalidades não é uma característica da Daily Meeting. Este evento é destinado a atualizações rápidas sobre o que foi feito, o que será feito e identificação de impedimentos, não para discussões profundas ou planejamento;

(d) Errado. Focar no progresso feito desde a última reunião é exatamente um dos objetivos da Daily Meeting, permitindo que a equipe acompanhe o progresso diário e ajuste seus planos conforme necessário;

(e) Errado. Identificar rapidamente os impedimentos é uma parte importante da Daily Meeting. Isso permite que a equipe aborde os problemas de forma proativa e busque soluções em tempo hábil.

Gabarito: Letra C

---

29. (PROF. DIEGO / INÉDITA – 2024) O que é PI Planning?

a) Planejamento individual de cada membro da equipe.
b) Uma reunião de retrospectiva de todo o programa.
c) Um evento para alinhamento e planejamento de equipes em SAFe.
d) Uma cerimônia para revisar os incrementos de produto de cada equipe.
e) Uma reunião diária de sincronização para o Agile Release Train.

Comentários:

(a) Errado. PI Planning não é sobre o planejamento individual de cada membro da equipe, mas sim um evento de planejamento em larga escala para múltiplas equipes;

(b) Errado. Uma reunião de retrospectiva de todo o programa pode ser parte do ciclo de feedback em SAFe, mas não é o que define o PI Planning;

(c) Correto. O PI Planning é um evento central no framework SAFe (Scaled Agile Framework) que serve para o alinhamento e planejamento de todas as equipes envolvidas no Agile Release Train (ART). Durante este evento, que geralmente dura dois dias, as equipes alinham seus objetivos para o próximo Program Increment (PI), discutem dependências e planejam entregas;

(d) Errado. Uma cerimônia para revisar os incrementos de produto de cada equipe descreve mais apropriadamente a Sprint Review ou a System Demo em SAFe, e não o PI Planning;

(e) Errado. Uma reunião diária de sincronização para o Agile Release Train é mais como uma Daily Stand-up para equipes que operam sob o SAFe, não o PI Planning, que é um evento extenso e detalhado de planejamento.

Gabarito: Letra C

---

30. (PROF. DIEGO / INÉDITA – 2024) Como o Sprint Planning contribui para o Scrum?

a) Estabelecendo metas financeiras para o projeto.
b) Definindo o escopo de trabalho para a próxima Sprint.
c) Revisando o desempenho financeiro da última Sprint.
d) Selecionando as festividades de fim de projeto.
e) Ajustando a estratégia de marketing com base no feedback do cliente.

Comentários:

(a) Errado. Estabelecer metas financeiras para o projeto não é o foco do Sprint Planning dentro do framework Scrum. O planejamento financeiro geralmente ocorre em um nível mais estratégico e não é restrito a um único evento de Sprint;

(b) Correto. O Sprint Planning contribui para o Scrum definindo o escopo de trabalho para a próxima Sprint. Durante esta sessão, a equipe de desenvolvimento e o Product Owner discutem e selecionam as histórias do Product Backlog que serão trabalhadas na Sprint, estabelecendo um objetivo claro para o que se espera alcançar;

(c) Errado. Revisar o desempenho financeiro da última Sprint não é parte do Sprint Planning. O foco deste evento é planejar o trabalho para a próxima Sprint, não revisar aspectos financeiros;

(d) Errado. Selecionar as festividades de fim de projeto não é um objetivo do Sprint Planning. Este evento é dedicado ao planejamento do trabalho e não à organização de eventos sociais ou celebrações;

(e) Errado. Ajustar a estratégia de marketing com base no feedback do cliente pode ser uma atividade importante dentro de uma organização, mas não é o foco do Sprint Planning. O Sprint Planning está centrado em definir o que será desenvolvido na próxima Sprint.

Gabarito: Letra B

Lista de Questões

1. **(PROF. DIEGO / INÉDITA – 2024)** O Scrum of Scrums é uma cerimônia que facilita a comunicação entre várias equipes Scrum trabalhando no mesmo projeto.

2. **(PROF. DIEGO / INÉDITA – 2024)** A principal finalidade da Product Owner Sync Meeting é resolver impedimentos técnicos que as equipes de desenvolvimento enfrentam.

3. **(PROF. DIEGO / INÉDITA – 2024)** O PI Planning é uma cerimônia exclusiva do framework Scrum.

4. **(PROF. DIEGO / INÉDITA – 2024)** Durante o Sprint Planning, apenas o Product Owner define o que será trabalhado na próxima Sprint.

5. **(PROF. DIEGO / INÉDITA – 2024)** O Sprint Review foca na inspeção do incremento do produto e na coleta de feedback dos stakeholders.

6. **(PROF. DIEGO / INÉDITA – 2024)** A Sprint Retrospective destina-se a inspecionar o produto entregue durante a Sprint.

7. **(PROF. DIEGO / INÉDITA – 2024)** A Daily Meeting deve durar no máximo 15 minutos, independentemente do tamanho da equipe.

8. **(PROF. DIEGO / INÉDITA – 2024)** No Scrum of Scrums, apenas os Scrum Masters das equipes participam.

9. **(PROF. DIEGO / INÉDITA – 2024)** A Product Owner Sync Meeting ajuda a sincronizar a visão do produto entre diferentes Product Owners.

10. **(PROF. DIEGO / INÉDITA – 2024)** O PI Planning não inclui a identificação de riscos e dependências entre equipes.

11. **(PROF. DIEGO / INÉDITA – 2024)** No Sprint Planning, a equipe de desenvolvimento ajuda a definir como o trabalho será realizado.

12. **(PROF. DIEGO / INÉDITA – 2024)** O Sprint Review é realizado antes do início de uma nova Sprint.

13. **(PROF. DIEGO / INÉDITA – 2024)** A Sprint Retrospective ajuda a equipe a planejar melhorias para o próximo ciclo de trabalho.

14. **(PROF. DIEGO / INÉDITA – 2024)** A Daily Meeting é uma cerimônia fechada onde somente os membros da equipe de desenvolvimento podem falar.

15. **(PROF. DIEGO / INÉDITA – 2024)** O Scrum of Scrums substitui a necessidade de Daily Meetings individuais em cada equipe.

16. **(PROF. DIEGO / INÉDITA – 2024)** O objetivo do PI Planning é alinhar todas as equipes de um Agile Release Train para o próximo Program Increment.

17. **(PROF. DIEGO / INÉDITA – 2024)** A Product Owner Sync Meeting ocorre diariamente para revisar o progresso de cada equipe.

18. **(PROF. DIEGO / INÉDITA – 2024)** No Sprint Planning, a equipe de desenvolvimento seleciona itens do Product Backlog que podem ser concluídos durante a Sprint.

19. **(PROF. DIEGO / INÉDITA – 2024)** Durante o Sprint Review, são discutidas e planejadas as melhorias dos processos internos da equipe.

20. **(PROF. DIEGO / INÉDITA – 2024)** A Sprint Retrospective é realizada ao final de cada Sprint para refletir sobre a eficiência da equipe e identificar áreas de melhoria.

21. **(PROF. DIEGO / INÉDITA – 2024)** Qual é o objetivo principal do Scrum of Scrums?

a) Resolver problemas técnicos específicos da equipe de desenvolvimento.
b) Coordenar a publicação do produto final.
c) Garantir que as equipes estejam alinhadas em termos de objetivos e entregáveis.
d) Revisar e adaptar o Product Backlog individualmente para cada equipe.
e) Facilitar a integração de código entre as equipes.

22. **(PROF. DIEGO / INÉDITA – 2024)** Qual das seguintes opções não é um objetivo da Product Owner Sync Meeting?

a) Coordenar os roadmaps de produtos.
b) Discutir estratégias de marketing.
c) Gerenciar dependências cruzadas entre produtos.
d) Alinhar sobre a priorização de funcionalidades.
e) Compartilhar conhecimento sobre desafios e sucessos.

23. **(PROF. DIEGO / INÉDITA – 2024)** O que caracteriza o PI Planning no contexto do SAFe?

a) Uma reunião diária para acompanhar o progresso do desenvolvimento.
b) Um evento de dois dias para planejamento e alinhamento de múltiplas equipes.
c) Uma retrospectiva de fim de sprint para avaliar o desempenho da equipe.
d) Uma revisão do sprint para apresentar o trabalho concluído aos stakeholders.

e) Uma cerimônia para celebrar o lançamento de novas funcionalidades.

24. **(PROF. DIEGO / INÉDITA – 2024)** Durante o Sprint Planning, o que a equipe de desenvolvimento faz?

a) Avalia o desempenho do último sprint.
b) Seleciona itens do Product Backlog que serão concluídos na próxima Sprint.
c) Apresenta o incremento do produto aos stakeholders.
d) Discute mudanças estratégicas com a alta gerência.
e) Planeja a festa de lançamento do produto.

25. **(PROF. DIEGO / INÉDITA – 2024)** Qual é o foco principal do Sprint Review?

a) Identificar e resolver impedimentos técnicos.
b) Inspecionar o incremento do produto e adaptar o Product Backlog.
c) Discutir o desempenho individual dos membros da equipe.
d) Priorizar itens para o próximo Product Backlog.
e) Celebrar os sucessos do último sprint.

26. **(PROF. DIEGO / INÉDITA – 2024)** O que não é discutido durante a Sprint Retrospective?

a) Como melhorar a eficiência da equipe.
b) A qualidade do código desenvolvido no último sprint.
c) Planos de ação para implementar melhorias identificadas.
d) O progresso do projeto em relação ao cronograma geral.
e) Estratégias de vendas para o produto desenvolvido.

27. **(PROF. DIEGO / INÉDITA – 2024)** Quem participa da Daily Meeting?

a) Apenas o Scrum Master e o Product Owner.
b) Todos os membros da equipe de desenvolvimento.
c) Stakeholders externos e clientes.
d) A alta gerência da empresa.
e) Apenas os membros da equipe de desenvolvimento designados pelo Scrum Master.

28. **(PROF. DIEGO / INÉDITA – 2024)** Qual das seguintes não é uma característica da Daily Meeting?

a) Ser realizada no mesmo local e horário todos os dias.
b) Ter duração máxima de 15 minutos.
c) Discutir detalhadamente novas ideias de funcionalidades.
d) Focar no progresso feito desde a última reunião.
e) Identificar rapidamente os impedimentos.

29. (PROF. DIEGO / INÉDITA – 2024) O que é PI Planning?

a) Planejamento individual de cada membro da equipe.
b) Uma reunião de retrospectiva de todo o programa.
c) Um evento para alinhamento e planejamento de equipes em SAFe.
d) Uma cerimônia para revisar os incrementos de produto de cada equipe.
e) Uma reunião diária de sincronização para o Agile Release Train.

30. (PROF. DIEGO / INÉDITA – 2024) Como o Sprint Planning contribui para o Scrum?

a) Estabelecendo metas financeiras para o projeto.
b) Definindo o escopo de trabalho para a próxima Sprint.
c) Revisando o desempenho financeiro da última Sprint.
d) Selecionando as festividades de fim de projeto.
e) Ajustando a estratégia de marketing com base no feedback do cliente.

GABARITO

1. CORRETO
2. ERRADO
3. ERRADO
4. ERRADO
5. CORRETO
6. ERRADO
7. CORRETO
8. ERRADO
9. CORRETO
10. ERRADO
11. CORRETO
12. ERRADO
13. CORRETO
14. ERRADO
15. ERRADO
16. CORRETO
17. CORRETO
18. CORRETO
19. ERRADO
20. CORRETO
21. LETRA A
22. LETRA B
23. LETRA B
24. LETRA B
25. LETRA B
26. LETRA E
27. LETRA B
28. LETRA C
29. LETRA C
30. LETRA B

DISCIPLINED AGILE DELIVERY

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

O Disciplined Agile Delivery (DAD) é uma abordagem ágil híbrida e abrangente para o desenvolvimento de soluções, desenvolvida por Scott Ambler e Mark Lines. Ele faz parte do Disciplined Agile Toolkit e é considerado um framework para guiar equipes no desenvolvimento ágil de software, indo além de práticas tradicionais como Scrum, Kanban e SAFe.

O DAD é projetado para ser flexível e adaptável, permitindo que as equipes escolham as práticas que melhor atendam às suas necessidades, em vez de seguir um único método prescritivo. Ele oferece uma visão holística que cobre todo o ciclo de vida de entrega de soluções, desde a concepção até a entrega ao cliente.

CARACTERÍSTICAS	DESCRIÇÃO
CICLO DE VIDA COMPLETO	Diferentemente de metodologias como Scrum, que se concentram principalmente na gestão de tarefas, o DAD aborda o ciclo de vida completo de um projeto, incluindo fases de iniciação, construção, transição e entrega contínua.
HÍBRIDO E FLEXÍVEL	O DAD combina práticas de várias metodologias ágeis, como Scrum, Kanban, XP (eXtreme Programming) e até mesmo abordagens tradicionais, como Waterfall, criando uma abordagem híbrida que pode ser ajustada conforme o contexto.
FOCO EM PESSOAS	Coloca as pessoas no centro do processo, reconhecendo que equipes auto-organizadas e colaborativas são fundamentais para o sucesso de projetos ágeis.
ESCOLHAS BASEADAS EM CONTEXTO	O DAD incentiva equipes a avaliarem seu contexto antes de adotar práticas específicas, em vez de seguir um único método como um "tamanho único".
ABORDAGEM BASEADA EM OBJETIVOS	Orientado por objetivos em cada etapa do ciclo de vida, permitindo que as equipes ajustem suas práticas para atender a metas específicas de maneira eficaz.
GOVERNANÇA INCLUSIVA	Inclui práticas de governança para garantir alinhamento organizacional, conformidade e rastreabilidade, algo que frequentemente falta em metodologias ágeis puras.
AGILIDADE EMPRESARIAL	Permite a escalabilidade da agilidade em toda a organização, conectando equipes de desenvolvimento a outras áreas, como negócios, operações e suporte.

Analogia:

Imagine que você está gerenciando um restaurante. Cozinhar e entregar refeições para os clientes é como desenvolver e entregar software em uma equipe ágil. O Disciplined Agile Delivery (DAD) seria o método que você usa para organizar toda a operação da cozinha e garantir que as refeições sejam entregues com qualidade, eficiência e no tempo certo.

No restaurante, o cardápio representa as opções de práticas e métodos disponíveis para cozinhar. O DAD não obriga você a seguir uma receita específica (como Scrum ou Kanban), mas oferece um "buffet" de técnicas que você pode escolher de acordo com o tipo de cliente, os ingredientes disponíveis e a equipe que você tem.

Por exemplo: se um cliente quer algo rápido, você pode optar por pratos simples (Kanban); se o cliente pede algo elaborado, você escolhe um método mais estruturado (Scrum com sprints bem organizadas). O DAD garante que você possa escolher a melhor prática para cada situação.

Diferente de métodos como o Scrum, que se concentra em como preparar o prato principal (a execução), o DAD cuida de todo o processo de refeição: Entrada – Planejamento inicial, como definir os objetivos e entender o pedido do cliente; Prato Principal – Desenvolvimento e montagem do prato, onde a maior parte do trabalho acontece; Sobremesa – Finalização, revisão e entrega do prato ao cliente. No DAD, todo o ciclo de vida da entrega é considerado, desde a ideia inicial até a entrega final e o feedback.

O chef principal precisa adaptar a operação da cozinha ao que está acontecendo naquele momento. Isso inclui considerar: disponibilidade de ingredientes (recursos); experiência da equipe de cozinha (maturidade da equipe); preferências do cliente (requisitos do projeto). O DAD funciona exatamente assim: o foco está no contexto. Ele permite que você adapte as práticas às circunstâncias, em vez de seguir uma receita rígida.

No restaurante, existem regras de higiene, padrões de qualidade e controle de custos que precisam ser seguidos. No DAD, isso é representado pela governança inclusiva, que garante que o trabalho da equipe siga boas práticas, esteja alinhado com os objetivos do negócio e atenda a normas, como prazos e orçamentos.

O DAD não se concentra apenas na cozinha (a equipe de desenvolvimento). Ele também considera a interação com o garçom (a equipe de operações), o caixa (a equipe de negócios) e o gerente (a liderança da organização). Essa integração é essencial para garantir que toda a operação funcione harmoniosamente.

Depois de um dia de trabalho, o chef e a equipe avaliam: Quais pratos foram os mais pedidos? O que poderia ser feito de forma mais rápida? Como podemos melhorar a experiência do cliente? Da mesma forma, o DAD promove a melhoria contínua, garantindo que a equipe aprenda com cada iteração e ajuste suas práticas para entregar mais valor no futuro.

Assim como um chef que lidera uma cozinha versátil, o DAD é uma abordagem flexível e holística que permite adaptar as práticas às necessidades do momento. Ele garante que toda a equipe, do planejamento à entrega, trabalhe de forma coordenada, garantindo a satisfação do cliente final. No mundo da agilidade, o DAD é como o chef que não apenas cozinha, mas gerencia toda a operação do restaurante, garantindo que tudo funcione como um sistema integrado e eficiente!

Por que utilizá-lo?

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Existem diversas razões pelas quais você deve considerar adotar o DAD. Vejamos algumas delas na tabela a seguir:

RAZÕES	DESCRIÇÃO
O DAD CONTINUA DE ONDE O SCRUM PARA...	O DAD descreve como todas as técnicas ágeis se encaixam, indo muito além do Scrum, para definir um ciclo de vida completo de entrega de soluções ágeis. Assim como o Scrum, o DAD aborda liderança, papéis e responsabilidades e gerenciamento de mudanças nos requisitos. Mas, ao contrário do Scrum, o DAD não para por aí, também trata de outros aspectos importantes do desenvolvimento de software, como arquitetura, design, testes, programação, documentação, implantação e muitos outros. Em resumo, o DAD oferece uma compreensão muito mais ampla de como o desenvolvimento ágil funciona na prática, fazendo grande parte do "trabalho pesado do processo" que o Scrum deixa para você.
O DAD É PRAGMÁTICO...	O kit de ferramentas do DA oferece opções, não prescrições, permitindo que você personalize facilmente uma estratégia que reflita a situação em que sua equipe se encontra. Para fazer isso de forma eficaz, você precisa entender as escolhas orientadas a processos que tem e quais são os trade-offs. O DAD torna essas escolhas explícitas por meio de sua abordagem orientada a objetivos de processos.
O DAD SUPORTA FORMAS DE TRABALHO (WOW) ÁGEIS E LEAN...	O DAD suporta vários ciclos de vida de entrega, incluindo um ciclo de vida ágil baseado no Scrum, um ciclo de vida lean baseado no Kanban, dois ciclos de entrega contínua, um ciclo de vida exploratório baseado no Lean Startup e um ciclo de vida de Programas. As equipes se encontram em situações únicas e, como resultado, um único tamanho de processo não serve para todos. Mesmo em empresas pequenas, observamos situações em que algumas equipes adotam uma abordagem ágil, outras uma abordagem lean e algumas uma combinação de ambas.
O DAD É BASEADO NO EMPIRISMO...	Durante vários anos, Scott Ambler, Mark Lines e muitos outros colaboradores do DAD trabalharam ou visitaram centenas de empresas ao redor do mundo, em uma ampla variedade de indústrias e ambientes. O DAD, e o kit de ferramentas DA em geral, captura as estratégias comprovadas adotadas por essas organizações, descrevendo os pontos fortes e fracos de cada estratégia e fornecendo orientações sobre quando aplicá-las e quando não.
O DAD FORNECE UMA BASE SÓLIDA PARA ESCALAR O ÁGIL...	O DAD apoia a escalabilidade bem-sucedida de técnicas ágeis e lean de várias maneiras. Primeiro, seus ciclos de vida completos e a amplitude de conselhos sobre desenvolvimento de software respondem como aplicar o ágil na prática com sucesso. Segundo, sua abordagem orientada a objetivos fornece a flexibilidade necessária para personalizar seu processo ágil para enfrentar os desafios enfrentados por equipes ágeis que trabalham em escala. Terceiro, o DAD incorpora muitos conceitos fundamentais necessários em escala, incluindo DevOps, governança ágil explícita e consciência organizacional.
O DAD HABILITA E VAI ALÉM DO SAFE...	O SAFe deixa os detalhes da construção para você e, como resultado, pode se mostrar frágil em muitas organizações. O DAD fornece a base sólida de processos que falta no SAFe e, de fato, é complementar ao SAFe. O DAD descreve várias estratégias para organizar equipes grandes ou distribuídas geograficamente. Ele descreve uma variedade de opções para escalar sua abordagem de desenvolvimento de software ágil e lean, oferecendo opções sensíveis ao contexto que o SAFe não aborda.
AS EQUIPES DO DAD ENTREGAM SOLUÇÕES,	O DAD reconhece que o software que desenvolvemos roda em hardware, que pode precisar de atualizações, e é suportado por documentação. Nossos stakeholders também podem precisar evoluir seus processos de negócios e, às vezes, até suas estruturas organizacionais,

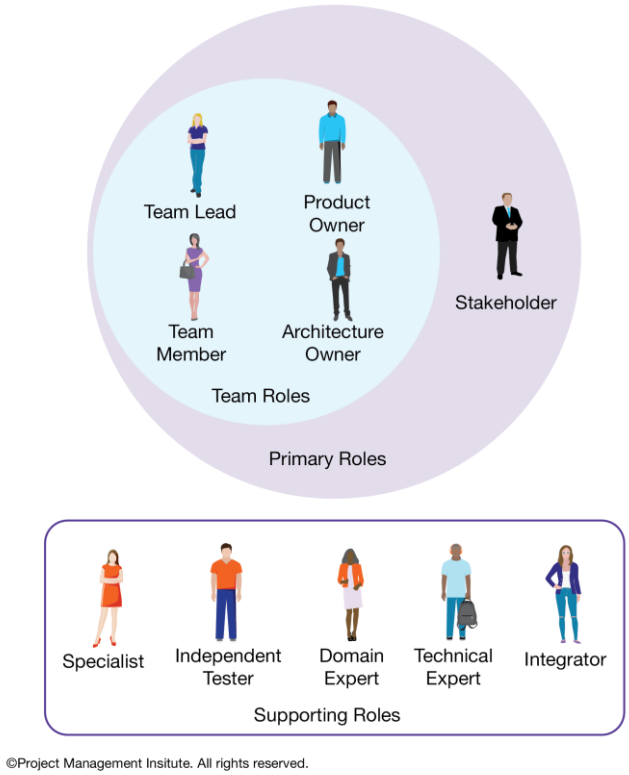
NÃO APENAS SOFTWARE...	para atender às novas necessidades da situação que enfrentam. Em resumo, as equipes do DAD entregam soluções consumíveis que incluem software, mudanças de hardware, documentação de suporte, processos de negócios aprimorados e até mesmo mudanças organizacionais.
O DAD ESTÁ EVOLUINDO...	Estamos constantemente aprendendo como praticantes, aprendendo sobre e experimentando novas estratégias ágeis e lean o tempo todo. Esses aprendizados estão sendo constantemente aplicados para evoluir o DAD.
O DAD É UMA PARTE CRÍTICA DE SUA ESTRATÉGIA DE DISCIPLINED DEVOPS...	O DAD é a parte "Dev" do Disciplined DevOps. Ele inclui conselhos explícitos para uma gama de estratégias de desenvolvimento que habilitam o DevOps.

Principais Papeis

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

O DAD inclui um robusto conjunto de papeis para entrega de soluções ágeis. Eles podem ser divididos da seguinte maneira:

- **Papeis Primários:** esses papeis são comumente encontrados em equipes independentemente do nível de escala da equipe.
- **Papeis de Suporte:** esses papeis são preenchidos, geralmente de forma temporária, para endereçar problemas de escala.



PAPEIS	DESCRIÇÃO
STAKEHOLDER	Um stakeholder é alguém que é materialmente impactado pelo resultado da solução. Nesse sentido, o stakeholder é claramente mais do que um usuário final: um stakeholder pode ser um usuário direto, usuário indireto, gerente de usuários, gerente sênior, membro da equipe de operações, o "dono do ouro" que financia o projeto, membro da equipe de suporte (help desk), auditor, gerente de programa/portfólio, desenvolvedores que trabalham em outros sistemas que se integram ou interagem com o que está em desenvolvimento, ou profissionais de manutenção potencialmente afetados pelo desenvolvimento e/ou implantação de um projeto de software. As equipes do DAD trabalharão idealmente junto com seus stakeholders diariamente ao longo do projeto.
MEMBRO DA EQUIPE	O papel do membro da equipe foca na produção da solução real para os stakeholders. Os membros da equipe realizarão atividades como testes, análise, arquitetura, design,

	programação, planejamento, estimativas e muitas outras, conforme apropriado durante o projeto. Nem todo membro da equipe terá todas essas habilidades, pelo menos não inicialmente, mas eles terão um subconjunto delas e se esforçarão para adquirir mais habilidades ao longo do tempo. Os membros da equipe são, às vezes, descritos pelos métodos ágeis principais como "desenvolvedores" ou simplesmente como programadores. No entanto, no DAD, reconhecemos que nem todo membro da equipe necessariamente escreve código. Os membros da equipe identificarão tarefas, estimarão tarefas, "se inscreverão" para tarefas, executarão as tarefas e acompanharão seu status até a conclusão.
LÍDER DA EQUIPE	Um aspecto importante das equipes auto-organizadas é que o líder da equipe facilita ou orienta o time na execução de atividades de gestão técnica, em vez de assumir essas responsabilidades sozinho. O líder da equipe é um líder-servidor do time, criando e mantendo as condições que permitem ao time ser bem-sucedido. O líder da equipe também atua como um coach ágil, ajudando a manter o time focado na entrega de itens de trabalho e no cumprimento de suas metas e compromissos de iteração feitos ao product owner. Ele ou ela age como um verdadeiro líder, facilitando a comunicação, capacitando o time a otimizar seus processos, garantindo que o time tenha os recursos necessários e removendo qualquer impedimento (resolução de problemas) de forma oportuna. Quando as equipes são auto-organizadas, uma liderança eficaz é crucial para o sucesso.
DONO DO PRODUTO	Em um sistema com centenas ou milhares de requisitos, muitas vezes é difícil obter respostas para perguntas relacionadas aos requisitos. O product owner é o único indivíduo na equipe que fala como a "voz do cliente". Ele ou ela representa as necessidades e desejos da comunidade de stakeholders para a equipe ágil de entrega. Dessa forma, esclarece quaisquer detalhes sobre a solução e é também responsável por manter uma lista priorizada de itens de trabalho que a equipe implementará para entregar a solução. Embora o product owner possa não ser capaz de responder a todas as perguntas, é sua responsabilidade buscar a resposta de forma oportuna para que a equipe possa se concentrar em suas tarefas. Trabalhar próximo à equipe para responder a qualquer pergunta sobre os itens de trabalho enquanto eles estão sendo implementados reduz substancialmente a necessidade de documentação de requisitos, testes e design. Ainda haverá necessidade de documentação como manuais de operações, manuais de suporte e guias do usuário. Cada equipe do DAD, ou subequipe no caso de grandes programas organizados em equipes interligadas, possui um único product owner. Um objetivo secundário do product owner é representar o trabalho da equipe ágil para a comunidade de stakeholders, incluindo a organização de demonstrações da solução à medida que ela evolui e a comunicação do status do projeto para stakeholders importantes.
DONO DA ARQUITETURA	A arquitetura é uma fonte importante de risco para o projeto, e alguém precisa ser responsável por garantir que a equipe mitigue esse risco. Como resultado, o DAD inclui explicitamente o papel do proprietário de arquitetura do Agile Modeling. O proprietário de arquitetura é a pessoa responsável pelas decisões de arquitetura da equipe e que facilita a criação e evolução do design geral da solução. O líder da equipe frequentemente também assume o papel de proprietário de arquitetura em equipes pequenas. Isso não é sempre o caso, especialmente em escala, mas é muito comum para equipes ágeis menores. Embora o proprietário de arquitetura seja tipicamente o desenvolvedor mais experiente da equipe - e às vezes seja conhecido como arquiteto técnico, arquiteto de software ou arquiteto de solução - vale ressaltar que esta não é uma posição hierárquica na qual outros membros da equipe se reportam. Ele ou ela é como qualquer outro membro da equipe e deve se inscrever e entregar trabalho relacionado a tarefas, como qualquer outro membro. Os proprietários de arquitetura devem ter um histórico técnico e um sólido entendimento do domínio de negócios.

Vantagens e Desvantagens

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

Dentre as vantagens de utilização dessa abordagem, podemos mencionar: oferece suporte para todas as fases de um projeto, garantindo uma visão integrada desde a concepção até a entrega; não é prescritivo; permite que equipes escolham práticas baseadas em seu contexto e necessidades; combina o melhor de diferentes abordagens ágeis (Scrum, XP, Kanban) e tradicionais, ajudando equipes a encontrar um equilíbrio ideal; ajuda a integrar equipes de desenvolvimento com outras áreas da organização, promovendo agilidade empresarial; inclui práticas para atender às necessidades de conformidade e rastreabilidade, muitas vezes exigidas em ambientes corporativos; incentiva o foco no que agrega valor real ao cliente, ajudando a priorizar as atividades de maneira mais eficaz.

Dentre as desvantagens, podemos citar: pode ser complexo para equipes iniciantes em agilidade devido à ampla variedade de práticas e opções; requer profissionais experientes para avaliar o contexto e escolher as práticas mais adequadas, o que pode ser desafiador para equipes menos maduras; a inclusão de práticas de governança e rastreabilidade pode parecer burocrática em organizações que já possuem alto nível de agilidade; para equipes que preferem uma abordagem estruturada, como Scrum, a flexibilidade do DAD pode ser confusa ou desafiadora; a personalização e configuração de práticas podem consumir mais tempo inicialmente, antes de atingir a velocidade de execução.

O DAD é uma abordagem poderosa para equipes e organizações que desejam adotar ou escalar práticas ágeis, mas que enfrentam requisitos complexos e variados. Com sua flexibilidade e foco no ciclo de vida completo, o DAD ajuda as equipes a encontrar a abordagem certa para o seu contexto. No entanto, é mais indicado para organizações com experiência em agilidade ou que têm a disposição para investir tempo e recursos na implementação e configuração. Quando usado corretamente, o DAD pode levar a entregas mais rápidas, melhor alinhamento organizacional e maior valor para os clientes.

AGILE-WATERFALL

Conceitos Básicos

INCIDÊNCIA EM PROVA: BAIXÍSSIMA

A metodologia híbrida Agile-Waterfall combina elementos da abordagem Waterfall (Cascata), tradicionalmente preditiva e sequencial, com práticas da metodologia Ágil, que é iterativa, incremental e adaptativa. Essa combinação visa aproveitar o melhor dos dois mundos, utilizando a estrutura e o controle do Waterfall para determinadas fases do projeto e a flexibilidade e colaboração do Agile para outras áreas.

Analogia:

Imagine que você está construindo uma casa. Esse projeto envolve partes que precisam ser bem planejadas e seguidas rigidamente (como fundação e estrutura), mas também áreas que podem ser ajustadas ao longo do processo (como decoração e acabamentos). A metodologia Agile-Waterfall funciona como a combinação desses dois estilos de trabalho.

No início do projeto, você precisa de um plano sólido e detalhado para garantir que a fundação seja construída corretamente. Isso inclui: criar plantas da casa; definir os materiais necessários; obter todas as aprovações (como licenças de construção).

Assim como no Waterfall, essa etapa segue um processo linear e sequencial. Você não pode começar a construir as paredes sem antes concluir a fundação. Esse planejamento é essencial para garantir que a casa tenha uma estrutura estável e segura.

Agora que a estrutura principal está pronta, entra a parte criativa e flexível, como: escolher os móveis; pintar as paredes; decidir onde colocar os quadros e plantas. Aqui, o Agile entra em ação. Você pode ajustar as cores das paredes ou a posição dos móveis à medida que avança. Por exemplo: durante a montagem da cozinha, você percebe que prefere uma bancada maior. Ao instalar as luzes, descobre que algumas áreas precisam de mais iluminação e ajusta o plano. Essas decisões são tomadas iterativamente, com base em como o espaço realmente funciona e no feedback da família ou dos moradores.

A combinação dos dois métodos garante que: as partes estruturais (fundação e paredes) sejam entregues dentro do prazo e orçamento, com alta previsibilidade, como no Waterfall. As partes personalizáveis (acabamentos e decoração) sejam flexíveis e ajustadas conforme necessário, como no ágil. Você não precisa decidir todos os detalhes de decoração no início do projeto, mas também não pode improvisar a construção da fundação. Isso reflete a essência do Agile-Waterfall: usar cascata onde é necessário controle rígido e ágil onde a flexibilidade agrega mais valor.

Nessa metodologia, as fases iniciais do projeto (como planejamento, requisitos, entre outros) seguem o formato sequencial do modelo em cascata – isso é útil para projetos que exigem forte documentação inicial ou aprovação antes de avançar. Durante o desenvolvimento e a entrega,

utiliza-se a metodologia ágil para criar ciclos iterativos e incrementais (como sprints ou iterações). Isso permite maior flexibilidade para lidar com mudanças de requisitos e feedback contínuo.

Algumas áreas do projeto, como gerenciamento de custos e cronogramas, podem ser gerenciadas com o modelo em cascata, enquanto outras, como desenvolvimento de software ou teste, utilizam práticas ágeis. O Waterfall oferece controle e documentação detalhada, enquanto o Agile minimiza a burocracia e promove a entrega contínua de valor. A abordagem ágil incentiva o envolvimento constante dos stakeholders e equipes multidisciplinares durante o ciclo de vida do projeto. Vejamos a seguir suas principais vantagens e desvantagens:

VANTAGENS	DESCRIÇÃO
FLEXIBILIDADE COM ESTRUTURA	Oferece a flexibilidade do Agile para lidar com mudanças, mantendo a estrutura e previsibilidade do Waterfall em fases críticas.
ATENDIMENTO A DEMANDAS HÍBRIDAS	Projetos que envolvem partes bem definidas (como infraestrutura) e outras com alta incerteza (como design e desenvolvimento) se beneficiam dessa abordagem.
MELHOR ALINHAMENTO COM O NEGÓCIO	A estrutura inicial do Waterfall permite alinhar expectativas com stakeholders, enquanto o Agile facilita a adaptação às necessidades em evolução.
FOCO NA ENTREGA DE VALOR	A entrega incremental do Agile garante que o cliente receba valor contínuo ao longo do projeto.
CONTROLE SOBRE CUSTOS E CRONOGRAMA	O planejamento detalhado do Waterfall ajuda a monitorar prazos e orçamentos com maior precisão.

DESVANTAGENS	DESCRIÇÃO
COMPLEXIDADE NA IMPLEMENTAÇÃO	Combinar duas abordagens tão distintas pode ser desafiador, especialmente para equipes inexperientes.
CONFLITOS DE CULTURA	Equipes que estão acostumadas com uma abordagem (Waterfall ou Agile) podem resistir à integração de elementos da outra.
SOBRECARGA DE PLANEJAMENTO	O planejamento detalhado do Waterfall pode criar atrasos, enquanto o Agile geralmente requer adaptação contínua.
DIVISÃO ARTIFICIAL DE FASES	Em alguns projetos, forçar uma separação clara entre fases sequenciais e iterativas pode não funcionar bem.
RISCO DE INEFICIÊNCIA	A falta de integração adequada entre os processos Agile e Waterfall pode levar à perda de eficiência e alinhamento.

A Metodologia Agile-Waterfall é ideal para projetos que possuem fases bem definidas (como engenharia ou infraestrutura) e partes incertas ou criativas (como desenvolvimento de software); para organizações em transição que estejam migrando de abordagens tradicionais para ágeis, mas ainda precisam manter elementos estruturados; para projetos de grande porte, em que há necessidade de documentação inicial para aprovação e flexibilidade nas fases de execução; e para setores regulados, como finanças, saúde ou governo, onde algumas partes do projeto exigem conformidade rigorosa e outras demandam inovação.

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.