

# DESENVOLVIMENTO DE SISTEMAS

Métodos e Técnicas Ágeis. DevOps e Outros



Livro Eletrônico



# SUMÁRIO

Apresentação .....	4
Métodos e Técnicas Ágeis. DevOps e Outros .....	5
DevOps .....	5
Principais Características .....	5
Benefícios Proporcionados pelo DevOps .....	5
Práticas Relacionadas ao DevOps .....	8
Ferramentas Utilizadas no DevOps .....	14
Metodologias Ágeis de Desenvolvimento .....	17
Lean Manufacturing ou Manufatura Enxuta (Pensamento Enxuto – PE ou Sistema Enxuto) .....	21
Scrum .....	23
Conceitos Básicos .....	23
Papéis do Scrum (ou Time Scrum) .....	24
Artefatos do Scrum .....	25
Fluxo de Trabalho no Scrum .....	27
Eventos .....	28
Kanban .....	33
Conceitos Básicos .....	33
Fluxo de Trabalho no Kanban .....	35
Scrumban .....	35
Conceitos Básicos .....	35
eXtreme Programming (XP) .....	37
Valores .....	38
Práticas .....	39
Test Driven Development (TDD) .....	44
Behaviour-Driven Development (BDD) .....	50
Domain-Driven Design (DDD) .....	53
Metodologias Ágeis com DevOps .....	58

O conteúdo deste livro eletrônico é licenciado para 61984693488 Martins Rodrigues - 00193743132, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

---

Resumo .....	59
Questões Comentadas em Aula.....	64
Questões de Concurso .....	71
Gabarito .....	77
Gabarito Comentado .....	78
Referências .....	97

## APRESENTAÇÃO

Olá, querido (a) amigo (a), meus cumprimentos!

**"Se seus sonhos estiverem nas nuvens, não se preocupe, pois eles estão no lugar certo; agora, construa os alicerces!"**

Os alicerces para uma excelente prova são: persistência, garra, força de vontade, estudo disciplinado e fé em Deus!

Nesta aula merecem destaque o **DevOps** e as **metodologias ágeis de desenvolvimento**, como **Lean, SCRUM, eXtreme Programming (XP), TDD, BDD, DDD, Kanban e Scrumban**.

Em caso de dúvidas, acesse o fórum do curso ou entre em contato.

Um abraço!

# MÉTODOS E TÉCNICAS ÁGEIS. DEVOPS E OUTROS

## DevOps

### PRINCIPAIS CARACTERÍSTICAS

- O termo **DevOps** vem da junção da abreviação dos termos **Developers**, em que se refere ao time de desenvolvimento, e **Operations**, que se refere ao time de infraestrutura de TI de uma organização.
- **DevOps** consiste em uma **filosofia de trabalho**, em que **diferentes equipes trabalham unidas** para **construir, testar, entregar, monitorar e aprimorar sistemas de informação** robustos de uma forma ágil, apoiando-se em modernas **técnicas de automação de fluxos de trabalho e de infraestrutura**.
- Em um **modelo de DevOps**, as **responsabilidades são divididas**, com equipes de desenvolvimento e infraestrutura trabalhando de forma conjunta, podendo até mesmo serem combinadas em uma única unidade organizacional ou dividindo o mesmo espaço físico.

### DIRETO DO CONCURSO

**001.** (CESPE/BNB/ESPECIALISTA TÉCNICO/ANALISTA DE SISTEMA/2018) Julgue o item seguinte, a respeito de *web services*

DevOps é um conjunto de ferramentas e práticas de trabalho para integração entre os colaboradores dos grupos de desenvolvimento de código, de operações e de apoio, e pode ser utilizado na produção rápida e segura de aplicações e serviços.



Apesar do termo DevOps se referir apenas ao time de desenvolvimento e infraestrutura, **outras equipes, como o time de qualidade, testes, segurança e análise de negócio podem participar do ciclo de vida de desenvolvimento**, trabalhando em conjunto para a identificação precoce de problemas e assim contribuindo para a entrega rápida de soluções.

**Certo.**

### BENEFÍCIOS PROPORCIONADOS PELO DevOps

- Entrega rápida de valor;
- Melhoria da capacidade de inovação;
- Entregas mais resilientes;
- Melhoria do clima organizacional; etc.



Figura. Benefícios DevOps

## DIRETO DO CONCURSO

**002.** (CESPE/STJ/ANALISTA JUDICIÁRIO/DESENVOLVIMENTO/2015) No que se refere à gestão de TI, julgue os itens a seguir.

O DevOps, movimento profissional emergente que defende uma colaboração maior entre desenvolvimento e operações de TI, resulta em um fluxo rápido do trabalho planejado, que aumenta a confiabilidade, a estabilidade e a segurança do ambiente de produção.



DevOps geralmente significa o **movimento profissional emergente** que defende uma **colaboração maior entre desenvolvimento e operações de TI**.

DevOps **resulta em um fluxo rápido do trabalho planejado** (por exemplo, altas taxas de implementação) **e ao mesmo tempo aumenta a confiabilidade, a estabilidade e a segurança do ambiente de produção**. Desenvolvimento e operações de TI representam geralmente o fluxo de valor entre os negócios (em que os requisitos são definidos) e o cliente (em que o valor é entregue).<sup>1</sup>

**Certo.**

- o DevOps **possui o foco no aumento da eficiência por meio da integração contínua e da automação de processos**.
- O primeiro passo para qualquer equipe adotar uma nova filosofia tal qual o DevOps consiste nas pessoas entenderem que precisam trabalhar juntas. A colaboração entre as diferentes áreas de TI é um dos pilares para a criação da mentalidade DevOps no time.

<sup>1</sup> Referência: <http://www.ibm.com/developerworks/br/library/se-devops/part1/>. 2014. Acesso: março de 2016.

- À medida em que as equipes vão se alinhando, naturalmente os silos existentes vão sendo desfeitos, com **desenvolvedores e analistas de infraestrutura repensando suas atribuições e dividindo a responsabilidade pelo sucesso do projeto.**
- De acordo com o Anexo I, do Guia de Projetos de Software com Práticas de Métodos Ágeis para o SISP, versão 1.0, **o conceito de DevOps reside na relação de interdependência entre os desenvolvedores de software e os profissionais de tecnologia da informação.**

O DevOps exige comunicação estreita entre os profissionais, colaboração contínua e cooperação máxima. Uma implementação

bem-sucedida da cultura DevOps em qualquer empresa passa, necessariamente, por uma mudança cultural. Além da cultura, existe outro aspecto essencial para que DevOps seja bem-sucedido: a automatização. Transparência e naturalidade ao lidar com adversidades dentro de um projeto de software são, a partir de DevOps, premissas que não devem, jamais, ser negligenciadas. Em qualquer time DevOps o objetivo comum de todos tem de ser apenas uma **entrega ágil** e com qualidade assegurada ao cliente.

- A **entrega rápida de valor para a área de negócio** é um dos primeiros benefícios da filosofia DevOps a ser percebido pelas áreas finalísticas. Dentre outros motivos, isso ocorre principalmente pela **adoção de ciclo de lançamento mais curtos**, ou seja: pequenas partes do projeto são gradativamente implantadas, de forma que o usuário final já possa ter algum benefício com o produto, ao mesmo tempo que gera feedback para aprimorar as próximas partes a serem implantadas.
- O DevOps aplica **abordagem ágil de desenvolvimento de software** ao permitir que um negócio maximize a velocidade de entrega de um produto ou serviço.



Figura. DevOps (ENAP, 2020)

## PRÁTICAS RELACIONADAS AO DEVOPS

- **Integração Contínua**

Consiste na combinação de cada trecho de código entregue ser automaticamente construído e testado, detectando falhas e vulnerabilidades o quanto antes.

Nessa prática observamos fácil transferência de conhecimento e experiências entre as áreas de Desenvolvimento, Operações e Apoio.

O fluxo apresentado na figura seguinte destaca um exemplo de utilização da integração contínua. **O processo se inicia com o programador realizando o commit do código para o repositório, ou seja, enviando o código para o repositório. Uma vez recebida a nova versão do código, uma ferramenta de integração contínua previamente configurada irá disparar um gatilho para construir a aplicação automaticamente (ENAP, 2020).**

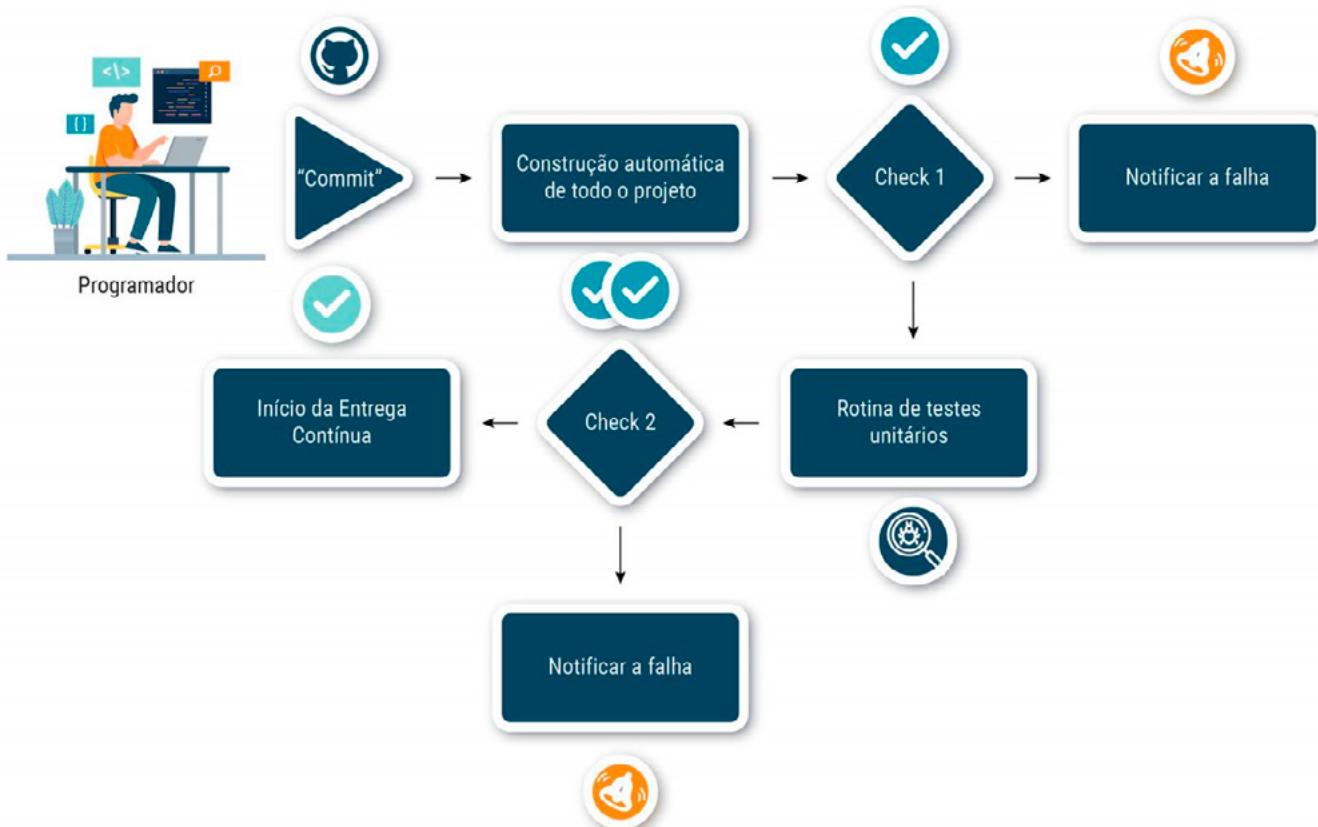


Figura. Exemplo de Uso da Integração Contínua na Prática (ENAP, 2020)

As possibilidades de personalização são variáveis, mas o conceito básico consiste em se ter **tarefas automatizadas sendo executadas de forma automática a partir da entrega de um trecho de código**.

- **Entrega Contínua**

Trata-se de uma prática fortemente dependente da integração contínua. Assim, muitas vezes são referenciadas de forma conjunta, por meio da sigla **CI/CD (Continuous Integration / Continuous Delivery)**.

O objetivo da **entrega contínua** é garantir que o código armazenado no repositório esteja sempre apto a ser implantado.

Conforme RedHat (2022):

CI/CD é um método para entregar apps com frequência aos clientes. Para isso, é aplicada a **automação** nas etapas do desenvolvimento de apps. Os principais conceitos atribuídos a esse método são **integração, entrega e implantação contínuas**. Com o CI/CD, é possível solucionar os problemas que a integração de novos códigos pode causar para as equipes de operação e desenvolvimento (o famoso “inferno de integração”).

Por meio do processo de entrega contínua, de acordo com ENAP (2020), é construída uma imagem da nova versão do software em um ambiente o mais próximo do ambiente de produção possível. Dessa forma, é importante o uso de soluções de **container**, tal como o **Docker**, que permitem criar uma instância do ambiente desejado de forma programática. Durante essa etapa, o sistema será publicado, podendo ocorrer uma nova rodada de testes, como testes de carga, para verificar que a aplicação está dentro dos parâmetros esperados para ser publicada. Ferramentas de entrega contínua, como o **OpenShift** e o **Gitlab** permitem controlar até mesmo a publicação da solução em ambiente de produção, reduzindo significativamente a complexidade da implantação de novas versões dos sistemas para a equipe de infraestrutura.

ENAP (2020) destaca que, durante a fase de integração contínua, todo novo código submetido ao repositório será automaticamente testado por meio de scripts automáticos. Os testes automatizados permitem identificar **erros** na aplicação de forma precoce, aumentando a velocidade das entregas e aprimorando a qualidade dos produtos.

Dentre as categorias de testes que podem ser incluídas em uma estratégia de testes para a aplicação merecem destaque (ENAP, 2020):

Testes Unitários	Checam uma funcionalidade especificamente, conferindo se a função se comporta como esperado ao se inserir dados válidos e inválidos.
Testes de Integração	Verificam se os módulos funcionam corretamente em conjunto, checando, por exemplo, se ocorrem conflitos de dependências ou se as interfaces conseguem se comunicar.
Testes de Regressão	Verificam se os novos módulos adicionados geram defeitos em outras partes do sistema.
Testes de Usabilidade	Verificam se uma pessoa consegue operar o sistema com facilidade, checando, por exemplo, se a interface do sistema está compatível com as recomendações do eMAG.
Testes de Desempenho	Verificam se o sistema consegue suportar a carga de usuários prevista e dentro de um tempo de resposta aceitável.

- **Microserviços**

São uma arquitetura derivada da arquitetura orientada a serviços SOA (*Service Oriented Architecture*). Trata-se de uma pequena funcionalidade independente e funcional, que se comunica com outros serviços ou aplicações por meio de uma interface REST (ENAP, 2020).

A adoção de microserviços, apesar de muito benéfica, **não** é recomendada para equipes com um baixo grau de maturidade em seus projetos, pois é bastante complexa.

- **Infraestrutura como Código**

**É a prática de tratar a infraestrutura de TI como se fosse código – exatamente como um software.**

Dentre os **benefícios** que a adoção da **Infraestrutura como código** pode trazer, pode-se citar (ENAP, 2020):

- **padronização**: uma vez definido o modelo para a criação de um determinado recurso (uma máquina virtual, por exemplo), todas as instâncias desse recurso seguirão o mesmo padrão;
- **agilidade**: o provisionamento e entrega de novos recursos é muito mais rápido, liberando analistas para outras atividades de maior valor agregado;
- **automação**: a utilização de scripts permite que o provisionamento de novos recursos seja automatizado;
- **reuso**: arquivos já criados para determinada demanda podem ser reaproveitados para demandas similares.
- **Monitoramento e Controle de Logs**

Permite a identificação precoce de problemas na aplicação.

Uma falha identificada precocemente evita o desperdício de recursos, permite corrigir falhas de segurança antes que elas se materializem e melhoram a experiência do usuário, aumentando o valor da solução entregue (ENAP, 2020).

Uma vez que a organização possui um ambiente de implantação automatizado, **ferramentas de monitoramento podem disparar gatinhos de problemas encontrados**, iniciando **automaticamente** o processo de implantação de uma versão limpa da aplicação (ENAP, 2020).

## DIRETO DO CONCURSO

**003.** (CESPE/MPE-PI/ANALISTA MINISTERIAL/TECNOLOGIA DA INFORMAÇÃO/2018) Acerca da gestão ágil de projetos com Scrum, de DevOps e da arquitetura corporativa (TOGAF), julgue o próximo item.

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.



**Infraestrutura como código (IaC)** é um processo em que o gerenciamento e o provisionamento de recursos de infraestrutura, tais como servidores, bancos de dados, espaços de armazenamento e afins são realizados por meio de arquivos de configuração e scripts, ao invés do acesso direto a máquinas físicas ou uso manual das ferramentas tradicionais. Isso permite a aplicação de práticas que incluem controle de versão, revisão por pares, testes automatizados, lançamento de releases por tags e entrega contínua, por exemplo.

**Certo.**

---

**004. (CESPE/SLU-DF/ANALISTA DE GESTÃO DE RESÍDUOS SÓLIDOS/INFORMÁTICA/2019)**

Com relação a *DevOps* e TOGAF, julgue o seguinte item.

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.



Trecho retirado do livro de Sharma e Coyne. De acordo com os autores, em **DevOps**, o princípio **monitorar e validar a qualidade operacional** transfere o monitoramento das características funcionais e não funcionais dos sistemas para o início do ciclo de vida do software, para que seja possível 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.

**Obs.:** Referência:

Sharma, Sanjeev; Coyne, B. **DevOps for Dummies: A Wiley Brand**. 2015. (<https://www.immagic.com/eLibrary/ARCHIVES/EBOOKS/W150421S.pdf>)

**Certo.**

---

**005. (CESPE/STF/TÉCNICO JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2013)** Quanto à gestão ágil de projetos com Scrum e às noções gerais de *DevOps*, julgue os itens subsecutivos. O *DevOps* aplica abordagem ágil de desenvolvimento de software ao permitir que um negócio maximize a velocidade de entrega de um produto ou serviço.



**Metodologias ágeis** são abordagens de desenvolvimento com **foco na entrega rápida e incremental de soluções para o usuário**. Metodologias ágeis tem um foco em manter o nível de produtividade do time de desenvolvimento elevado para a rápida entrega de valor para o cliente.

Já o **DevOps** possui o **foco no aumento da eficiência** por meio da integração contínua e da automação de processos (ENAP,2020).

Observando o foco das duas abordagens, é possível identificar que **ambas tratam da questão da eficiência para entrega rápida de valor**.

Logo, ao serem adotadas de forma conjunta, é possível capturar os pontos fortes de cada abordagem, resultando em um elevado nível de entrega de valor para o usuário (ENAP,2020), permitindo que um negócio maximize a velocidade de entrega de um produto ou serviço.

**Certo.**

---

**006.** (CESPE/STJ/TÉCNICO JUDICIÁRIO/SUPORTE TÉCNICO/2018) Julgue o item seguinte, a respeito de DevOps e das disposições constantes da NBR ISO/IEC 2700)

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.



O **DevOps** é compatível com a aplicação de métodos ágeis, tais como SCRUM, ou, ainda, com o uso de ferramentas que permitam visualizar os fluxos do processo.

**Errado.**

---

**007.** (FCC/SEFAZ-SC/AUDITOR-FISCAL DA RECEITA ESTADUAL/TECNOLOGIA DA INFORMAÇÃO/2018) No âmbito da infraestrutura, uma das vantagens *DevOps* é

- a) realizar *deploy* sob controle manual de especialistas.
- b) estabelecer uma divisão bem delimitada entre infraestrutura e desenvolvimento.
- c) possuir um ambiente aberto (não padrão) sem muitos controles.
- d) ter a infraestrutura como código.
- e) fazer com que cada etapa do processo seja aprovada formalmente pelos gerentes.



Conforme visto na figura seguinte, a **infraestrutura como código (IaC)** é uma das vantagens DevOps.

**Obs.:** | **IaC** é um processo em que o gerenciamento e o provisionamento de recursos de infraestrutura, tais como servidores, bancos de dados, espaços de armazenamento e afins são realizados por meio de arquivos de configuração e scripts, ao invés do acesso direto a máquinas físicas ou uso manual das ferramentas tradicionais.

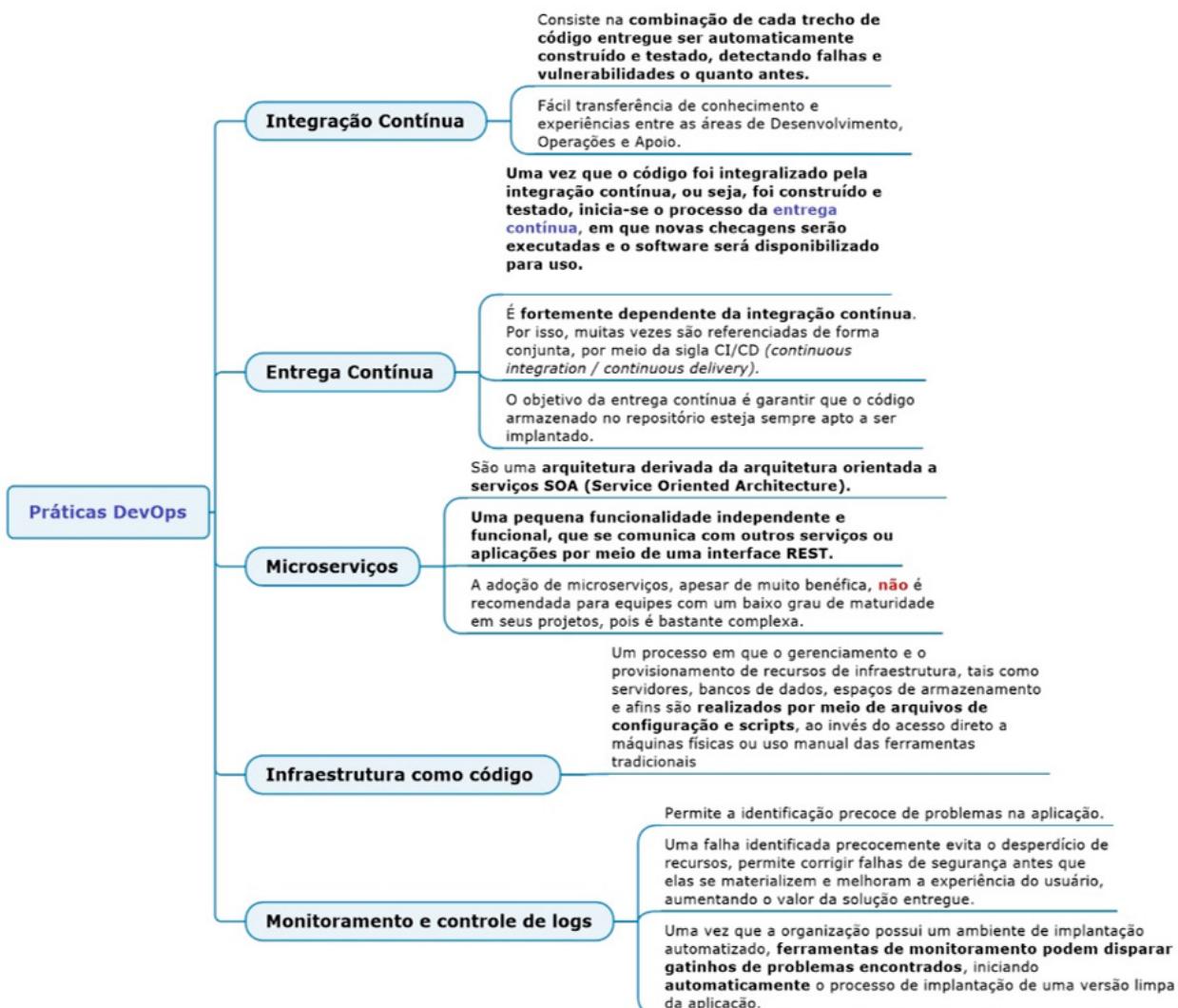


Figura. Práticas DevOps (QUINTÃO, 2023)

**Letra d.**

**008.** (FUMARC/TÉCNICO JUDICIÁRIO/TRT 3<sup>a</sup> REGIÃO/APOIO ESPECIALIZADO/TECNOLOGIA DA INFORMAÇÃO/2022) Analise as afirmativas a seguir referentes aos conceitos e ferramentas de *DevOps*:

- I – Um composto de *Dev* (desenvolvimento) e *Ops* (operações), *DevOps* é a união de pessoas, processos e tecnologias com foco na entrega rápida de serviços de TI.
- II – CI/CD pode ser definido como um método para entregar aplicações com frequência aos clientes, baseado em integração e entrega contínuas.
- III – Jenkins é uma ferramenta de integração contínua de código aberto, que pode ser usada para automatizar tarefas relacionadas à construção, teste, entrega e implantação de software. Está CORRETO o que se afirma em:

- a) I, apenas.
- b) I e II, apenas.
- c) I e III, apenas.
- d) II e III, apenas.
- e) I, II e III.



I – Certo. A palavra **DevOps** é a combinação dos termos “desenvolvimento” e “operações”. Mas, ela representa um conjunto de ideias e práticas que ultrapassam o significado desses dois termos. O *DevOps* inclui segurança, maneiras colaborativas de trabalhar, análise de dados e muitas outras práticas e conceitos. É a união de pessoas, processos e tecnologias com foco na entrega rápida de serviços de TI.

II – Certo. Conforme RedHat (2022):

**CI/CD é um método para entregar apps com frequência aos clientes.** Para isso, é aplicada a **automação** nas etapas do desenvolvimento de apps. Os **principais conceitos atribuídos a esse método são integração, entrega e implantação contínuas.**

III – Certo. Uma das ferramentas *open source* mais conhecidas para CI/CD é o servidor de automação Jenkins, que pode ser usada para automatizar tarefas relacionadas à construção, teste, entrega e implantação de software.

**Letra e.**

## FERRAMENTAS UTILIZADAS NO DEVOPS

Vamos então ao estudo das principais ferramentas utilizadas no DevOps.

### Orquestração de DevOps

Processo de **gestão, configuração e coordenação do conjunto de componentes de TI**, sejam eles, servidores, aplicações, serviços e afins.

Ferramentas relacionadas (ENAP, 2020):

<b>Kubernetes</b> 	Solução de orquestração de <i>containers</i> de código aberto. Conforme <a href="https://kubernetes.io/pt-br/">https://kubernetes.io/pt-br/</a> , trata-se de uma “plataforma de código aberto portável, extensível, para gestão de <i>workloads</i> e serviços baseados em <i>container</i> que facilita um ambiente de configuração declarativa e automatizada.
 <b>ANSIBLE</b>	Ferramenta de automação para configurar e implantar servidores, software e serviços a partir de um ponto central de gestão.

## Controle de Versão de Ambiente DevOps

Faz a gestão do ambiente em que os códigos ficarão armazenados, serão versionados e fornecidos para as diversas outras ferramentas de automação e gestão da arquitetura.

Ferramentas relacionadas (ENAP, 2020):

	<p>Utiliza um conceito de <i>branches</i>, que permitem testar variações da versão do código rapidamente. Caso o teste seja bem sucedido, é possível juntar as modificações introduzidas pela <i>branche</i> ao código principal por um processo de <i>merge</i>. Caso o teste seja mal sucedido, o código da <i>branche</i> pode ser removido do projeto.</p> <p>O git adota um modelo distribuído, no qual todos os usuários possuem uma cópia completa do repositório em que estão trabalhando.</p> <p>Dentro do repositório o git cria ainda uma área temporária - <i>Staging Area</i>, em que é possível adicionar apenas uma parte das modificações realizadas para serem salvas no repositório.</p> <p>Dessa forma, o fluxo básico de envio de um arquivo modificado para o git consiste em:</p> <ul style="list-style-type: none"><li>• <b>git add "nomearquivo"</b> – adiciona um arquivo para o Stage;</li><li>• <b>git commit -m "descrição do commit"</b> – adiciona a nova versão arquivo ao repositório local;</li><li>• <b>git push "repositório-remoto" "master"</b> envia os dados da <i>branch</i> "master" do repositório local para o repositório remoto.</li></ul>
	<p>Suite de ferramentas que apoia diversos processos de um ambiente DevOps, como o controle de versão. A ferramenta pode ser instalada no próprio centro de dados do órgão ou por meio da nuvem pública fornecida pela ferramenta. O controle de versão da ferramenta é baseado em GIT.</p>
	<p>Suite de ferramentas que apoia diversos processos de um ambiente DevOps. Dentre os seus diferenciais, a solução consta uma extensa lista de ferramentas de terceiros que podem ser conectadas ao repositório da sua aplicação, como ferramentas de produtividade e qualidade.</p>
	<p>Ferramenta de armazenamento e controle de versão da suite da Microsoft Azure DevOps Server, que substituiu o Team Foundation Server no final de 2018. Além das funcionalidades básicas, oferece um grande suporte a colaboração e integração com outras ferramentas da Microsoft. Oferece suporte a repositórios git e TFVC (Team Foundation Version Control).</p>

## Gerenciamento de Configuração

Permitem a implementação de um ambiente de Infraestrutura como código (IaC) fortemente baseado em automação. O uso desse tipo de ferramenta diminui a necessidade de recursos humanos, agiliza as atividades de provisionamento de ambiente e de *deploy*, além de contribuírem para garantir que a infraestrutura de TI esteja de acordo com o estado desejável.

**Ferramentas relacionadas (ENAP, 2020):**

 <b>CHEF</b>	Ferramenta de gestão de configuração de código aberto, com foco em automação em toda a infraestrutura de TI. O ambiente desejado é escrito utilizando Ruby em arquivos conhecidos como "receitas" ( <i>recipes</i> ). As receitas são compiladas dando origem aos livros <i>cookbooks</i> , que são a definição de como um nó será implantado.
 <b>puppet</b>	Ferramenta de gestão de configuração de código aberto, com suporte a implantações automáticas. Conta com um mecanismo de relatórios bastante avançado. Assim como o Chef, o Puppet utiliza Ruby como linguagem de programação para criação dos <i>Scripts</i> .

**Integração Contínua**

Atualmente, a maioria das ferramentas de repositório/controle de versão já contemplam algum tipo de mecanismo de integração contínua.

Existe ainda a opção de utilizar ferramentas de terceiros, como (ENAP, 2020):

 <b>Jenkins</b>	Ferramenta de integração contínua de código aberto, que pode ser usada para automatizar tarefas relacionadas à construção, teste, entrega e implantação de software.
 <b>go</b>	Destaca pela possibilidade de criação de fluxos de trabalho complexos, com atividades paralelas ou sequenciais, bem como o monitoramento da execução de uma forma visual.
 Team City	Ferramenta de CI/CD com suporte a múltiplas funcionalidades. Possui suporte para integração com diversos recursos, como servidores de controle de versão, plataformas de computação em nuvem e ferramentas de monitoramento.

**Monitoramento**

Monitorar ambientes de tecnologia de informação (TI) é uma tarefa importante, pois auxilia na identificação de problemas e gera gatilhos para ações preventivas capazes de evitar que a aplicação chegue a ficar indisponível.

Vejamos algumas **ferramentas** utilizadas para o **monitoramento de ambientes de TI** (ENAP, 2020):

 <b>Prometheus</b>	Ferramenta de código aberto. Conta com um modelo de dados multidimensional, descoberta de nós de forma automática ou manual, coleta de dados por HTTP, etc.
 <b>Nagios</b>	Sua versão básica, o Nagios core, pode ser baixada gratuitamente e oferece monitoramento de eventos e gestão de alertas. Possui uma ampla lista de <i>plugins</i> desenvolvidos pela comunidade que ampliam e melhoram a sua usabilidade.

## Controle de Logs

Uma ferramenta de controle de *logs* fornece um ponto único para monitoramento dos *logs*, facilita a localização dos registros desejados, permite controlar o acesso dos usuários apenas aos registros que ele realmente precisa e pode ainda cruzar registros, de forma a auxiliar na busca da causa raiz de um problema.

Vejamos algumas **ferramentas** de controle de *logs* (ENAP, 2020):

	Ferramenta de código aberto para coleta, armazenamento e análise de registros. Possui <i>dashboards</i> para facilitar a visualização dos dados, recursos de busca avançados, tolerância a falhas e pacotes de configuração para facilitar a adoção da ferramenta.
ELK Stack	<ul style="list-style-type: none"><li>Um conjunto de ferramentas abertas para monitoramento de ambientes. É composto por:</li><li>• <i>Elasticsearch</i>: engine de busca e análise de dados. Componente que irá <u>armazenar os dados</u> e processar as consultas aos mesmos.</li><li>• <i>Logstash</i>: componente que fará a consulta às diversas fontes de registros, realiza alguma transformação que seja necessária e armazena em um repositório (Elasticsearch, por exemplo).</li><li>• <i>Kibana</i>: componente visual do conjunto, em que os dados armazenados no Elasticsearch poderão ser consultados de uma forma visual.</li></ul>
	Ferramenta de código aberto para <b>coleta de dados</b> . Ponto único para coleta e consulta de dados de diferentes fontes.

## METODOLOGIAS ÁGEIS DE DESENVOLVIMENTO

A Engenharia de Software vem há anos criando técnicas de modelagem, projeto e desenvolvimento de sistemas. Dentre os desafios dos pesquisadores da área, pode-se citar a preocupação em desenvolver softwares com qualidade garantida, no prazo estabelecido e sem alocar recursos além do previsto. No entanto, muitas das metodologias criadas são consumidoras de recursos, aplicando-se principalmente a grandes sistemas.

Como a Engenharia de Software ainda está evoluindo, novos modelos estão sendo apresentados, como: **desenvolvimento ágil**; dentre outros.

Aqui merecem destaque as **metodologias ágeis de desenvolvimento**, como **Lean, SCRUM, eXtreme Programming (XP), TDD, BDD, DDD, Kanban e Scrumban**.

O desenvolvimento ágil de software fundamenta-se no **Manifesto Ágil** (<https://agilemanifesto.org/>). Por definição, os **12 princípios para a prática do desenvolvimento ágil** são:

Satisfação do cliente	A maior prioridade está em satisfazer o cliente por meio da entrega adiantada e contínua de software de valor.
Mudança em favor da vantagem competitiva	<b>Mudanças de requisitos são bem-vindas, mesmo em fases tardias do desenvolvimento.</b> Os processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente. As metodologias tradicionais são preditivas – esse cenário é ilusório.
Prazos curtos	<b>Entregar software em funcionamento com frequência</b> , desde a cada duas semanas até a cada dois meses, com uma preferência por prazos mais curtos.
Trabalho em conjunto	<b>Tanto pessoas relacionadas a negócios como desenvolvedores devem trabalhar em conjunto</b> , diariamente, durante todo o curso do projeto.
Ambientação e suporte	Para construir projetos ao redor de indivíduos motivados, <b>é preciso dar a eles o ambiente e o suporte necessários</b> , confiando que farão seu trabalho.
Falar na cara	O método mais eficiente de transmitir informações tanto externas como internas para um time de desenvolvimento é por <b>meio de uma conversa face a face</b> . As metodologias tradicionais utilizam documentos, diagramas, relatórios, telefonemas para promover a comunicação no projeto.
Funcionalidade	<b>Um software funcional é a medida primária de progresso.</b> As metodologias tradicionais propunham a entrega de artefatos (Exemplo: Documentação) que, em geral, não agregavam valor algum aos clientes, como também uma forma de medir o progresso do projeto.
Ambiente de sustentabilidade	Processos ágeis <b>promovem um ambiente sustentável</b> , com patrocinadores, desenvolvedores e usuários sendo capazes de manter passos constantes.
Padrões altos de tecnologia e design	A <b>contínua atenção à excelência técnica e ao bom design</b> aumenta a agilidade.
Simplicidade	<b>Fazer algo simples</b> é dominar a arte de maximizar a quantidade de trabalho que não precisou ser feito. As metodologias tradicionais, algumas vezes, recorriam a implementações desnecessariamente complexas, a planejamentos exageradamente detalhados, etc.
Autonomia	As <b>melhores arquiteturas, os requisitos e os designs emergem de times autoorganizáveis</b> . As metodologias tradicionais geralmente precisam de um gerente de projetos responsável por organizar o trabalho da equipe como um todo, sendo também responsável pela tomada de decisões.
Reflexões para otimizações	Em intervalos regulares, <b>o time reflete em como ficar mais efetivo</b> , então, se ajustam e otimizam seu comportamento de acordo. As metodologias tradicionais muitas vezes são engessadas, dessa forma não revisitam frequentemente seu comportamento.

De acordo com o Manifesto Ágil deve-se valorizar:

- os **indivíduos e interações** mais que processos e ferramentas;
- **software em funcionamento** mais que documentação abrangente;
- **colaboração com o cliente** mais que negociação de contratos;
- **responder a mudanças** mais que seguir um plano.

São **exemplos** de metodologias ágeis:

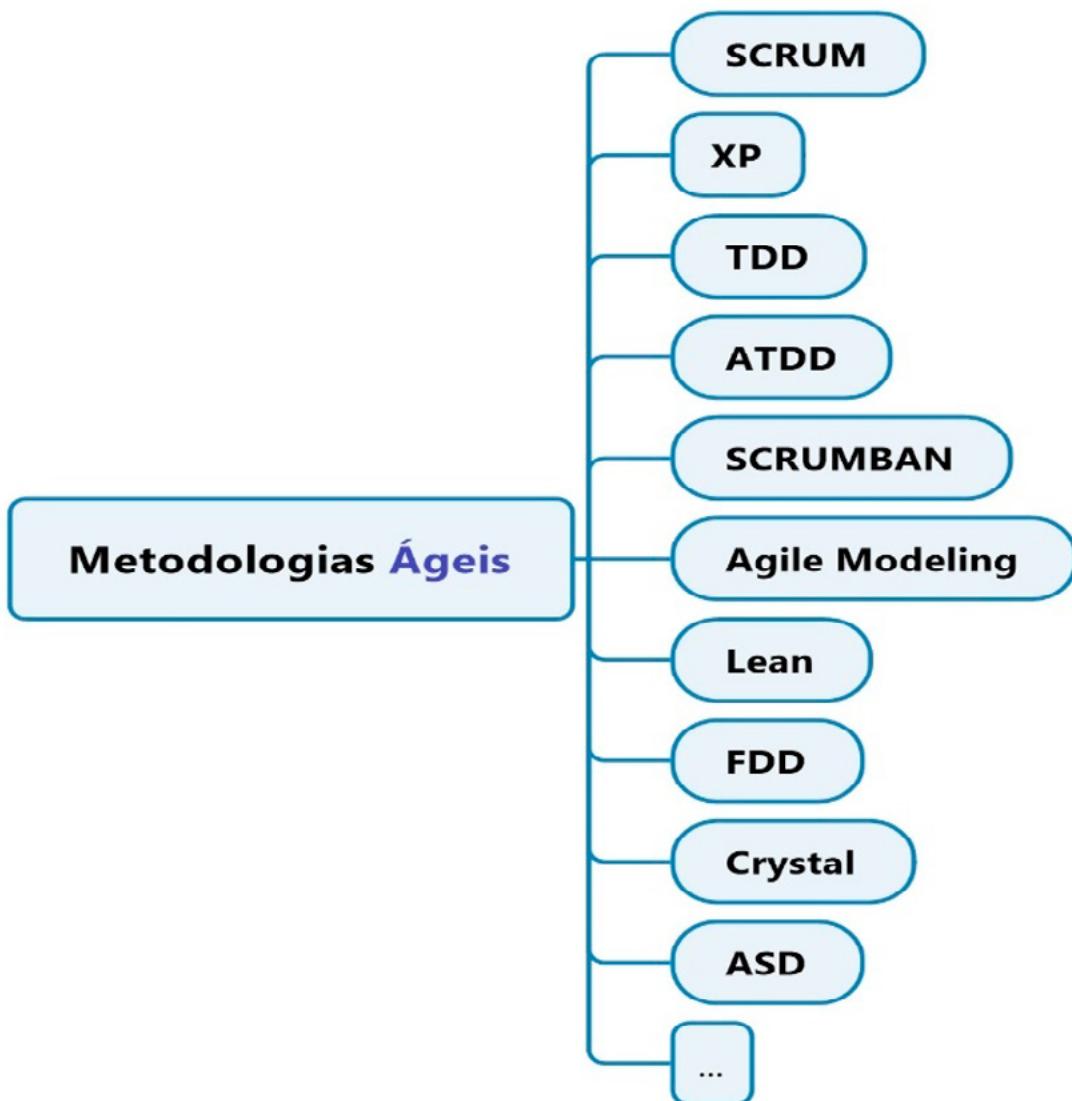


Figura. Metodologias Ágeis. Fonte: Quintão (2023)

Veja a seguir um esquema destacando as Metodologias Ágeis no âmbito dos principais modelos de processo de software.

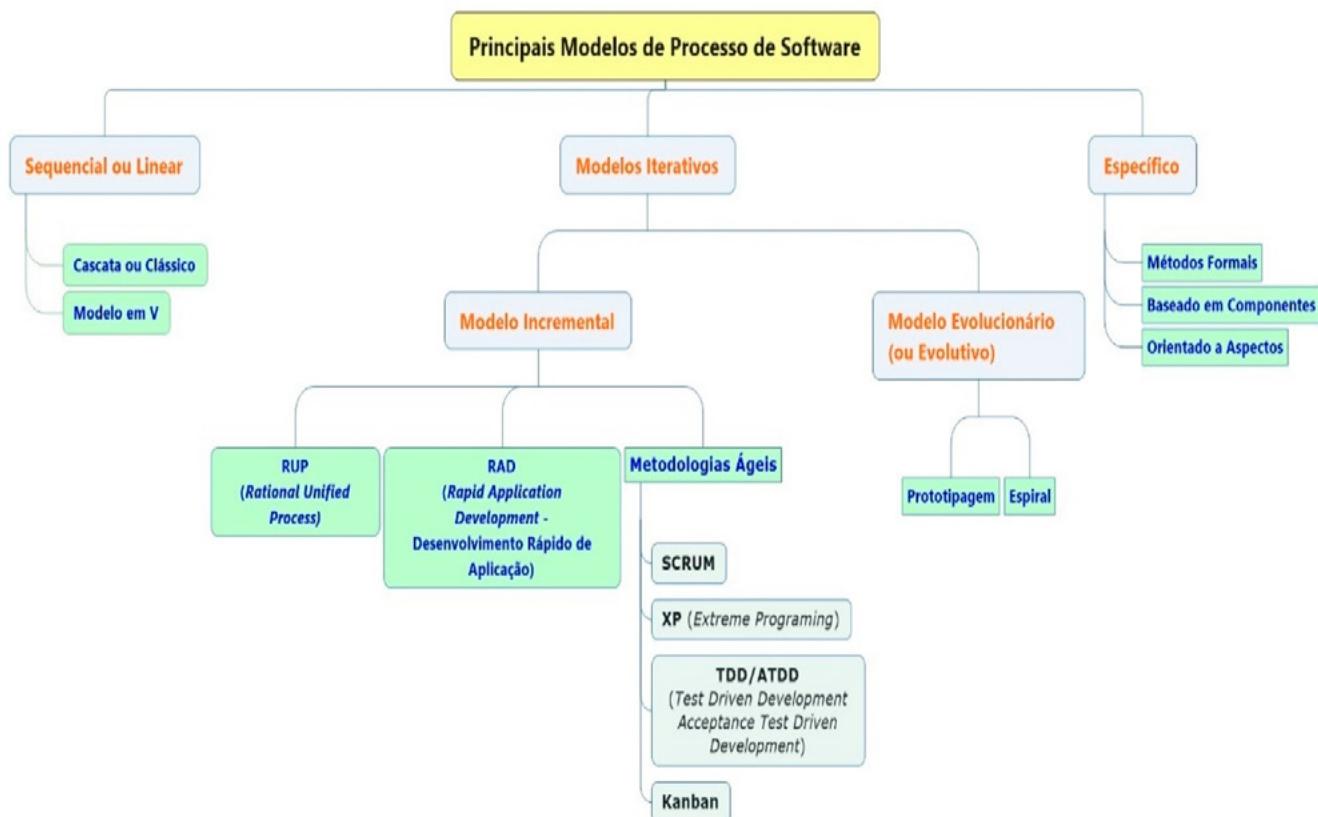


Figura. Principais Modelos de Processo de Software.  
Fonte: Quintão (2023)

Veja a seguir algumas características fundamentais entre as metodologias ágeis, segundo Sommerville:

- **não** há especificação detalhada do sistema, e a documentação do projeto é minimizada ou gerada automaticamente;
- o sistema é desenvolvido em uma série de versões;
- interfaces de usuário do sistema são geralmente desenvolvidas com um sistema **interativo** de desenvolvimento que permite a criação rápida do projeto.

## DIRETO DO CONCURSO

**009.** (ESAF/STN/ANALISTA DE FINANÇAS E CONTROLE/GOVERNANÇA E GESTÃO EM TI/2013) O desenvolvimento ágil de software fundamenta-se no Manifesto Ágil. Segundo ele deve-se valorizar:

- a) mudança de respostas em vez do seguimento de um plano.
- b) indivíduos e interações em vez de processos e ferramentas.
- c) documentação extensiva operacional em vez de software funcional.
- d) indivíduos e intenções junto a processos e ferramentas.
- e) seguimento de um plano em vez de resposta a mudança.



O Manifesto Ágil contém quatro valores fundamentais, que são:

- Os **indivíduos e suas interações** acima de procedimentos e ferramentas;
- **Software funcional** acima de documentação abrangente;
- A **colaboração dos clientes** acima da negociação de contratos;
- A **capacidade de resposta a mudanças** acima de um plano pré-estabelecido;<sup>2</sup>

Assim, tem:

- a) Errada. Temos aqui a capacidade de resposta às mudanças, ao invés de seguir um plano pré-estabelecido.
- b) Certa. Os **indivíduos e suas interações** acima de procedimentos e ferramentas.
- c) Errada. **Software funcional** acima de documentação abrangente.
- d) Errada. Os **indivíduos e suas interações** acima de procedimentos e ferramentas.
- e) Errada. A **capacidade de resposta a mudanças** acima de um plano pré-estabelecido.

**Letra b.**

## LEAN MANUFACTURING OU MANUFATURA ENXUTA (PENSAMENTO ENXUTO – PE OU SISTEMA ENXUTO)

Na década de 50, Taiichi Ohno implantou o **Sistema Toyota de Produção (Toyota Production System – TPS)** com o principal **objetivo de identificar e eliminar os possíveis desperdícios**, buscando reduzir custos e aumentar a qualidade e velocidade de entrega do produto aos clientes. Por representar uma maneira de produzir mais utilizando cada vez menos, este sistema foi denominado **Produção enxuta (Lean production ou Lean manufacturing)** por James P. Womack e Daniel T. Jones.

Assim, são vários as traduções e sinônimos para filosofia da **Manufatura Enxuta**, como **Lean Manufacturing, Lean, Pensamento Enxuto (PE)**, até porque não é aplicada apenas na manufatura, ou ainda **Sistema Enxuto**.

Segundo Cruz (2015), **Lean está ligado à eliminação de desperdício em qualquer etapa de um processo**. No desenvolvimento de software, isso está intrinsecamente ligado a funcionalidades que **não** agregam valor para o cliente, à perda de tempo em atividades que **não** entregam resultado, construção de funcionalidades que **não** serão utilizadas, retrabalho por **baixa qualidade**.

**Sua essência é a capacidade de eliminar desperdícios continuamente e resolver problemas de maneira sistemática.** Isso implica repensar a maneira como se lidera, gerencia e desenvolve pessoas. É por meio do pleno engajamento das pessoas envolvidas com o trabalho que se consegue vislumbrar oportunidades de melhoria e ganhos sustentáveis (LEAN, 2017).

<sup>2</sup> Fonte: <http://manifestoagil.com.br/>

Tyagi et al., (2015) destacaram que **as práticas de PE** podem ser aplicadas isoladamente, no entanto, **quando utilizadas de forma organizada e quando apoiada por um trabalho integrado da organização, aumentam potencialmente o desempenho da planta por meio da implementação de práticas do pensamento enxuto.** A principal razão para esses benefícios é a **sinergia**, o trabalho integrado de todos da organização para reduzir a variabilidade.

O conceito de PE possui aplicação em **toda a organização**, pois criam produtos ou serviços por meio de processos que ultrapassam as fronteiras funcionais com a intenção de criar valor para os clientes que podem ser externos ou internos.

De acordo com Womack e Jones (2004), **existem cinco princípios básicos fundamentais da mentalidade lean: valor, fluxo de valor, fluxo, produção puxada e perfeição.** Vamos à descrição desses princípios:

- **valor:** consiste em definir o que é Valor. Não é a empresa e sim o **cliente** que **define o que é valor;**
- **fluxo de Valor:** consiste em **identificar os processos** que geram valor, aqueles que não geram valor, mas são importantes para a manutenção dos processos e da qualidade e, por fim, aqueles que não agregam valor e que devem ser eliminados;
- **fluxo:** deve-se dar “**fluidez**” para os processos e atividades que restaram;
- **produção puxada:** conectam-se os processos por meio de sistemas puxados.
- **perfeição:** a **busca do aperfeiçoamento contínuo em direção a um estado ideal** deve nortear todos os esforços da empresa (WOMACK e JONES, 2004).

## DIRETO DO CONCURSO

**010. (FGV/IMBEL/ANALISTA ESPECIALIZADO/ANALISTA ADMINISTRATIVO/ REAPLICAÇÃO/2021)** A filosofia “Lean” é uma abordagem com foco em processos e que tem, como essência, a busca pela redução de desperdícios. Assinale a opção que apresenta um dos princípios basilares da filosofia “Lean”.

- a) Melhoria pontual.
- b) Maximização do uso de recursos.
- c) Processamento “empurrado”.
- d) Qualidade perfeita na segunda vez.
- e) Construção de relacionamento de curto prazo com os fornecedores.



O **Lean Manufacturing** é uma filosofia de gestão baseada na **eliminação de desperdícios** para consequente **diminuição de custos, melhoria de qualidade do produto final e maximização do uso de recursos** (WOMACK; JONES; ROOS, 1992).

**Letra b.**

## SCRUM

### CONCEITOS BÁSICOS

O **Scrum** é um *framework de processo ágil* utilizado para gerenciar e controlar o desenvolvimento de um produto de software por meio de práticas **iterativas** e **incrementais**.

A figura seguinte é bem mais interessante para ilustrar a diferença entre incremental e iterativo!

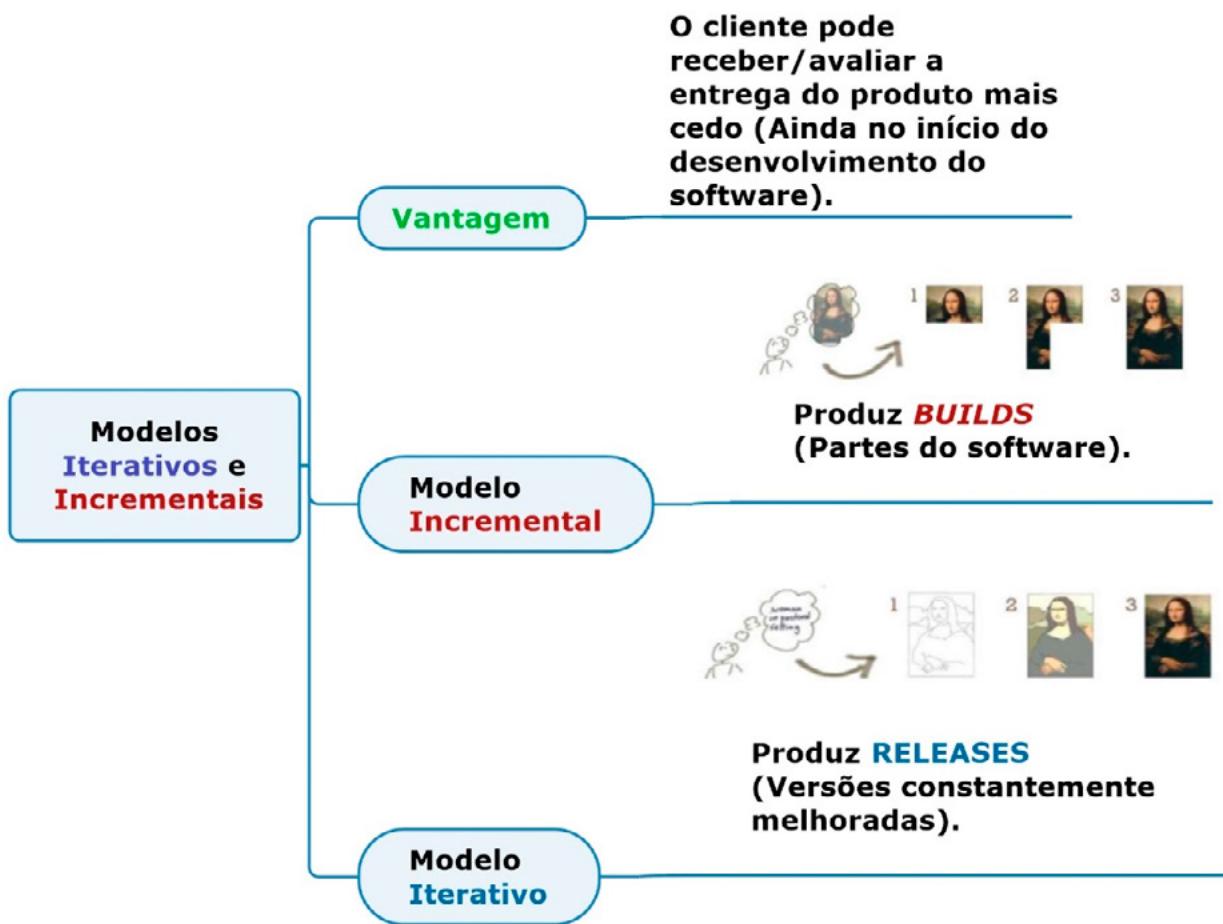


Figura. Modelos Iterativos e Incrementais. Fonte: Quintão (2022)

Scrum é composto por um conjunto de boas práticas de gestão que admite ajustes rápidos, acompanhamento e visibilidade constantes e planos realísticos.

**Obs.:** | Scrum é um processo bastante leve para gerenciar e controlar projetos de desenvolvimento de software e para a criação de produtos.

O Scrum é um **processo de desenvolvimento ágil de software** baseado em grupos de práticas e papéis predefinidos.

Ele é um processo **iterativo** e **incremental** para gerenciamento de projetos e desenvolvimento de sistemas, em que cada **sprint** é uma iteração que segue um ciclo PDCA (*Plan, Do, Check, Act*) e entrega um incremento de software pronto.

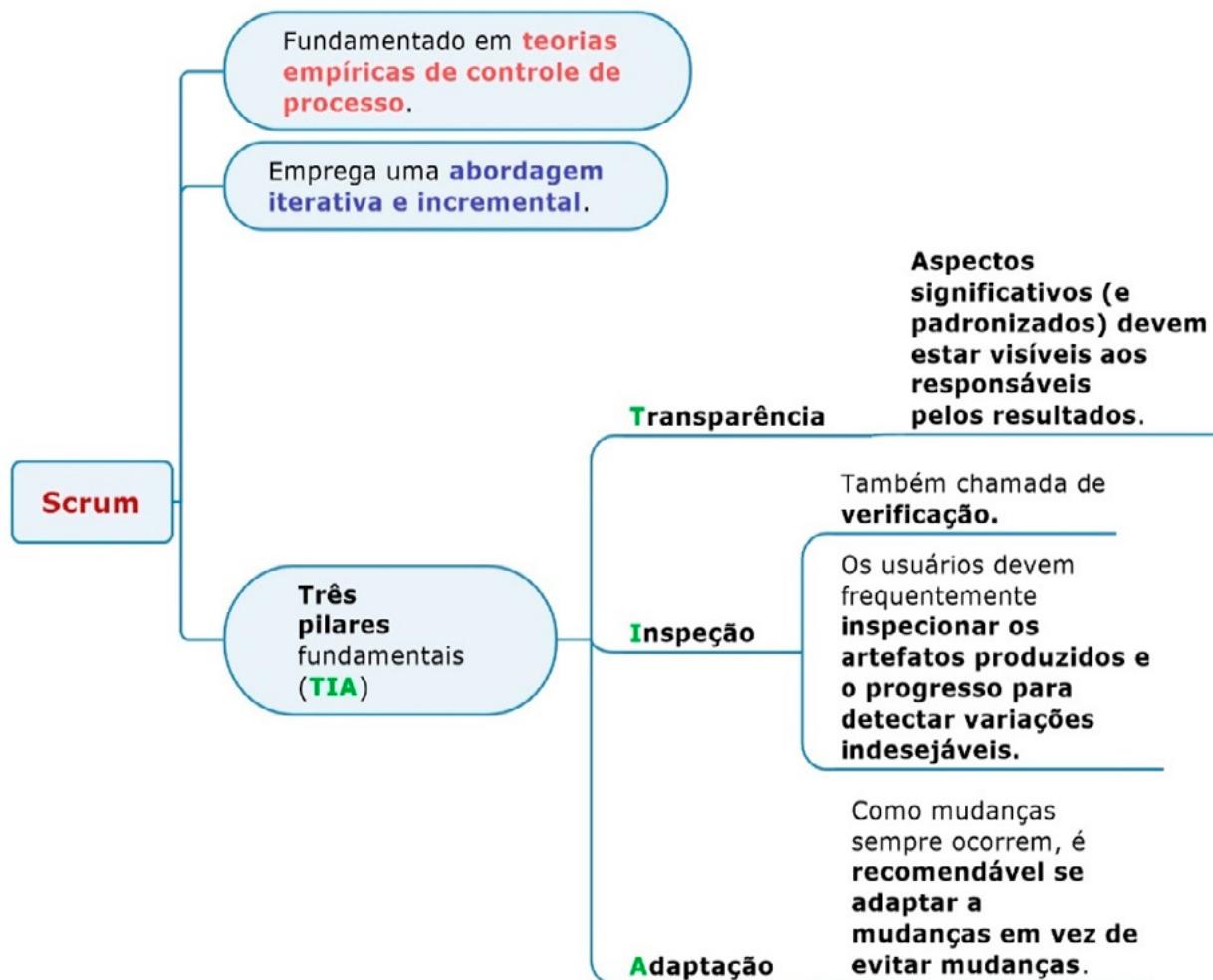


Figura. Scrum. Fonte: Quintão (2023)

## PAPÉIS DO SCRUM (OU TIME SCRUM)

Os principais papéis do Scrum são: **Product Owner (PO)**, **Scrum Master (SM)** e **Development Team (DT)**.

Papéis do Scrum	Descrição
Product Owner (PO)	<b>Dono do Produto.</b> É uma pessoa e <b>não</b> um comitê. Ele pode representar o desejo de um comitê no <i>Product Backlog</i> , mas aqueles que quiserem uma alteração nas prioridades dos itens de <i>backlog</i> devem convencer o <i>Product Owner</i> .
Development Team (Dt)	<b>Equipe de Desenvolvimento.</b> Profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto “pronto” ao final de cada <i>sprint</i> . O Time de Desenvolvimento só responde ao <i>Product Owner</i> . Responsável pela microgestão e pela criação do produto. <b>São auto-organizados.</b> Ninguém (nem mesmo o Scrum Master) diz ao Time de Desenvolvimento como transformar o <i>Product Backlog</i> em incrementos de funcionalidades potencialmente utilizáveis. Individualmente, os integrantes do Time de Desenvolvimento podem ter habilidades especializadas, mas a responsabilidade pertence aos desenvolvedores como um todo.
Scrum Master (SM)	<b>Mestre Scrum.</b> É responsável por garantir que o Scrum seja entendido e aplicado! Ele ajuda a treinar o time de desenvolvimento em autogerenciamento e interdisciplinaridade. <b>O Scrum Master</b> tem o papel de liderança muito importante para o processo. Ele deve remover todo e qualquer obstáculo que surgir durante o desenvolvimento, garantindo que o Scrum Team possa focar no real objetivo definido. Além disso, ele é responsável por fazer com que a equipe siga e pratique o processo e ainda por criar uma atmosfera de ajuda mútua entre a equipe (o resultado é sempre da equipe e não individual).

Não há como fazermos um mapeamento direto entre os papéis do Scrum e os papéis convencionais conhecidos. Não existe a figura única do Gerente de Projetos. Suas responsabilidades estão diluídas entre os papéis citados. Cada um conhece sua participação frente ao projeto e trabalha em conjunto para conseguir alcançar o *goal* definido.

## ARTEFATOS DO SCRUM

Segundo o Guia do Scrum, o *framework* possui apenas três artefatos oficiais, que são:

Artefatos Oficiais	Descrição
Product Backlog	Uma lista ordenada (por valor, risco, prioridade, etc.) de requisitos ou funcionalidades que o produto deve conter criada pela Equipe Scrum.
Sprint Backlog	Conjunto de itens selecionados para serem implementados durante a sprint mais o plano para transformá-los em um incremento.
Product Increment	Ao final de cada sprint, a equipe de desenvolvimento entrega um incremento do produto – resultado do que foi produzido ao final do trabalho realizado na sprint.

Inicialmente um requisito deve fazer parte do **Product Backlog (Produto Backlog)**, que possui descrições de necessidades de clientes de alto nível levantadas.

A tarefa de manter a descrição e refinamento destes requisitos é do **Product Owner**. Ele é responsável por definir, para cada nova *release* de um produto, o objetivo (**Release Goal** ou **Vision**). Ele interage a todo momento com seu cliente final e, dessa forma, conhece suas expectativas e as do mercado.

Esta lista de demandas nunca acaba e de tempos em tempos (a cada nova *release*), um item de **Product backlog** é promovido a **Release backlog** para integrar ao escopo de uma ou mais Releases. O **Product Owner** deve comunicar e discutir o novo *goal* com todos os envolvidos em reuniões chamadas de **Release Planning**.

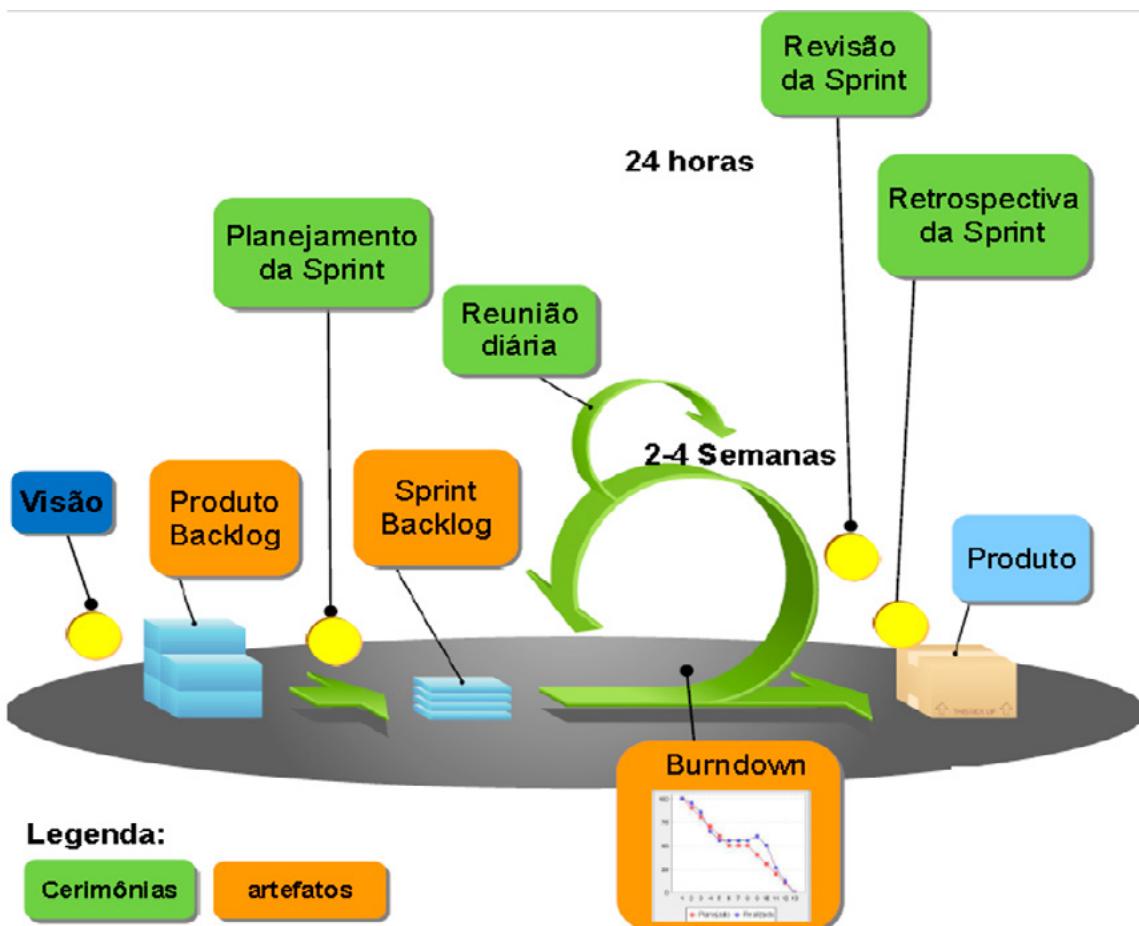


Figura – Estrutura do Scrum

Portanto, a primeira reunião oficial de *Release* que ocorre é a **Release Planning**. Nessa reunião, são discutidos os recursos envolvidos – tanto computacionais quanto humanos, riscos identificados, prazo acertado e requisitos previstos acordados junto ao cliente e coerentes com a demanda de mercado. Seu propósito principal é definir e comunicar as funcionalidades macro de uma *Release* do produto e fazer com que os envolvidos no projeto entendam e se comprometam com o *Release Goal* definido. As funcionalidades **não** devem ser discutidas em detalhes.

Isso será feito em cada iteração durante reuniões de **Sprint Planning** (Planejamento da Sprint). Ela pode ter a presença de diversos papéis: seja cliente externo, desenvolvedor, área comercial, etc. **Cada release é composta de iterações curtas, chamadas de Sprints.**

Conforme dito, a primeira reunião é a **Release Planning**. Cada Sprint tem reuniões de planejamento, Sprint Planning, que detalham o requisito apresentado durante o Release Planning e ainda possuem seu *Sprint Goal*.

Semelhante ao *goal* do Release, tem o mesmo cunho de comunicar a todos os envolvidos o objetivo maior de Sprint corrente. É realizado de forma **iterativa**, por meio de ciclos de “tempo fechado” (*Time-Boxed*) em **30 dias corridos**. Ainda sobre a reunião, são conhecidos os requisitos do Release Backlog que serão promovidos como *Selected Backlog*. Eles são discutidos em maiores detalhes e servem para elucidar **O QUE** se espera como resultado final.

**Em um Sprint, portanto, são executadas atividades de refinamento de requisitos, análise, projeto, desenvolvimento e testes pelo Scrum Team**, devendo resultar em um incremento de produtos funcionando e demonstrável para o *Product Owner* ao final do Sprint, na reunião de **Sprint Review**.

## FLUXO DE TRABALHO NO SCRUM

Conforme destaca Sotile (2018), “no Scrum, o trabalho que será feito no próximo sprint é previamente selecionado. A seguir, o sprint é bloqueado, o trabalho é feito e depois do sprint a fila está vazia”.

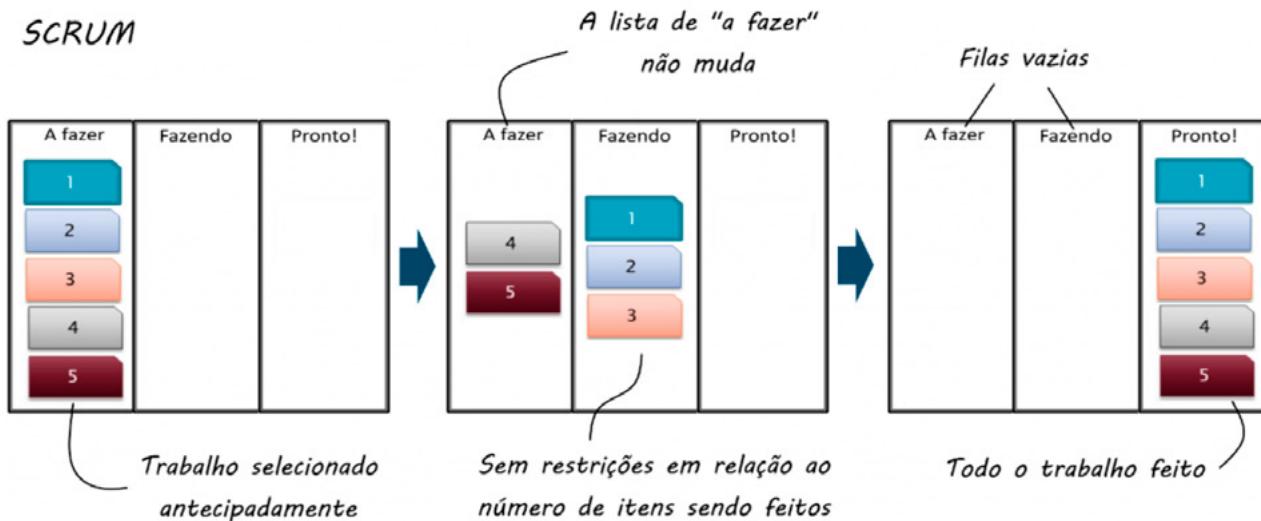


Figura. Trabalho no Scrum. Fonte: Sotile (2018)

## EVENTOS

Diariamente são executadas reuniões de acompanhamento e monitoramento do *Sprint* por meio da reunião de **Daily Scrum**. Esta reunião deve ter a presença do Scrum Team e Scrum Master obrigatoriamente.

**Deve ter a duração máxima de 15 minutos** (é interessante até utilizar um cronômetro para isso) e são permitidas somente 3 perguntas:

- **O que você fez hoje?**
- **O que fará amanhã?**
- **Que impedimentos surgiram e que atrapalharam sua produtividade?**

Outros assuntos deverão ser discutidos em outras reuniões, como **Follow-Up Meetings** para detalhamentos e elucidações relativas aos requisitos e **Reestimate Meetings** para aplicação de técnicas como o *Poker Planning* para estimativa de requisitos.

Há uma reunião de **Sprint Planning 2** que promove o desdobramento do *Selected Backlog* em *Sprint Backlog* nos quais aspectos técnicos são discutidos. É a hora do **COMO** as coisas devem ser feitas. Neste momento, pode-se discutir sobre a arquitetura do produto, reuso de componentes, mudança em interfaces, etc.

Ao final do Sprint, é feita uma reunião final de apresentação do trabalho executado ao *Product Owner* chamada **Sprint Review**. Ele será responsável por validar a apresentação e concluir se o Sprint Goal foi atingido. Após esta, reuniões de *debrief* são executadas, chamadas **Retrospective Meetings**.

Reuniões de retrospectiva levam a reflexão acerca do que passou e quais atitudes devem ser tomadas para o correto atingimento dos objetivos. São levantados **aspectos positivos (WWW – what went well)** e **negativos (WCBI - what can be improved)** tanto organizacionais quanto relativos a Release.

Todos os envolvidos devem estar presentes e são responsáveis por minimizar todo e qualquer risco ao projeto e também por manter os aspectos positivos levantados.

E o mesmo se repete por todos os **Sprints** planejados, com a diferença de que o último **Sprint** possui uma reunião de Release Review, que tem como guia o **Release Goal** definido. **Nunca é demais lembrar que o planejamento é feito durante toda a Release e, portanto, alterações podem ser feitas.** Constitui em uma boa prática poder refazer o planejamento durante cada **Sprint Planning**, mas durante o **Sprint**, o Scrum Master deve procurar “blindar” o Scrum Team de grandes modificações.

As estimativas de tamanho de cada funcionalidade/requisito são executadas obrigatoriamente pelo Scrum Team. Dessa forma, todos sabem para onde ir (**goal**) e é desta forma que se obtém o comprometimento da equipe para com o projeto.

A viabilidade de continuidade dos planos é constantemente verificada. A comunicação de um projeto que utiliza o Scrum é mais efetiva e direta, além de informações estarem sempre à tona com a utilização de **quadros brancos (Agile Radiator)**.

A figura seguinte exemplifica um *Agile Radiator* monitorando um projeto real. Eles garantem visibilidade do projeto a todos os envolvidos.

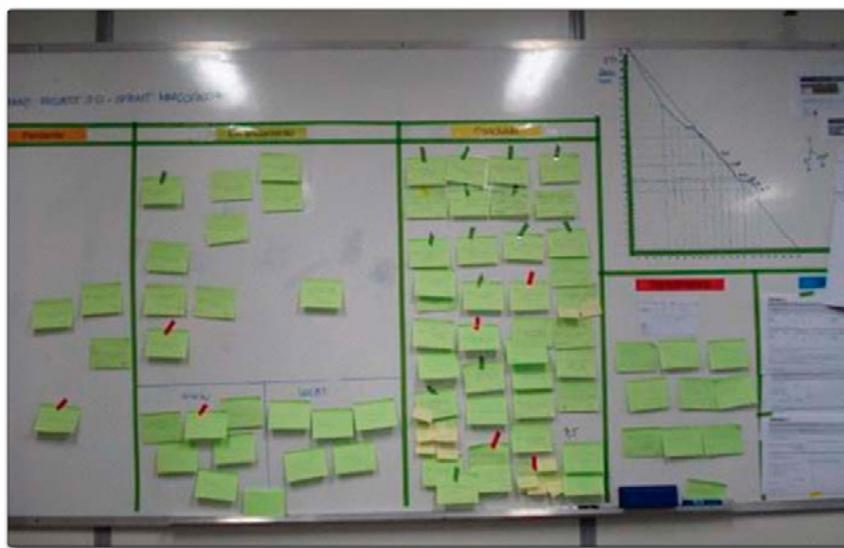


Figura. Agile Radiator

Não há como mascarar o real andamento. O **goal** fica afixado e os requisitos – **por meio de Post-its - (Selected backlogs)** e seus desdobramentos (**Sprint backlogs**) são posicionados na situação em que se encontram (se ainda não iniciados - **planejados**, se sendo executados no momento - **em andamento** e se terminados – 100% **concluídos**). Eles devem ser posicionados de acordo com a prioridade dos mesmos – *Business Value* declarado pelo *Product Owner*.

Post-its localizados no topo nos dizem ser de maior prioridade que os posicionados no rodapé do quadro branco. Impedimentos (**Impediment Backlog**) que ocorrem durante o Sprint também são evidenciados.

O Scrum tem um gráfico, de nome **BurnDown Chart**, que é a principal ferramenta de gerenciamento do processo de desenvolvimento de software. Ele representa o trabalho restante sobre tempo, ou seja, permite visualizar o progresso e/ou a evolução do trabalho executado pela equipe e a quantidade trabalho x tempo que ainda faltam para completar a Sprint. A atualização do **Burndown** deve ser diária, isto facilita a tomada de decisão, pois, poderemos decidir em melhorar a produtividade da equipe e/ou para mitigar risco da Sprint.

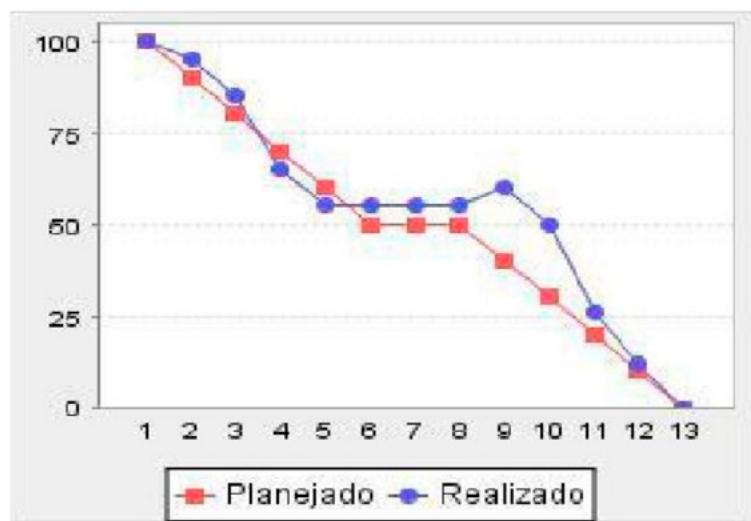


Figura. BurnDown Chart

O Scrum Team diariamente, por meio de reuniões chamadas *Daily Scrum*, atualiza o *Agile Radiator* com a situação atual, movendo **post-its** e indicando o trabalho executado durante o período.

Vale destacar que a granularidade de cada **Sprint Backlog** deve ser pequena.

Sua estimativa não deve ultrapassar 8 horas de trabalho e isso se constitui em uma boa prática no gerenciamento do andamento das atividades. Por meio desta reunião, o time consegue avaliar se é necessário mudar o plano para acertar o rumo, se deve priorizar alguma outra atividade, se muitos impedimentos estão ocorrendo e como minimizá-los, se há algum recurso que está necessitando de ajuda, etc. É real! Isso tudo sempre à luz do **goal** definido. Além de todas estas vantagens, ainda proporciona visibilidade a todos os envolvidos.



Durante cada reunião de **Daily Scrum**, *post-its* são “movimentados” pelo Scrum Team contemplando cada uma das perguntas associadas a esta reunião. Atividades executadas (o que foi feito hoje) são movidas para a seção “**Concluídos**”. Atividades planejadas (o que será feito amanhã) são colocadas na seção “**Em Andamento**” e impedimentos são registrados (que impedimentos surgiram? Repare a seção em vermelho no rodapé à direita).

Além destes artefatos, o *Burndown Chart* deve ser atualizado ao final da reunião contabilizando o tamanho total entregue pelo Scrum Team no dia a ser representado. O *Burndown* possui duas retas: uma que exibe o planejado e outra que reflete o realizado. Ele está registrado no topo à esquerda. Ele contém o total de trabalho restante e só de “batermos o olho” no gráfico conseguimos verificar a evolução do trabalho. Todo o *Agile Radiator* tem esta característica. Se estivermos vendo muitos impedimentos, temos que verificar o que está sendo feito para prevenir novos e se estes estão sendo corretamente removidos. Se atividades de maior prioridade estão sendo deixadas para trás, a própria equipe pode verificar o que está havendo. O que foi bom (WWW) e o que pode ser melhorado (WCBI) também ficam afixados no quadro branco.

A cada novo *Sprint*, o *Agile Radiator* deve ser modificado para retratar a situação atual novamente. E este ciclo se inicia: **planejamento, execução, controle e avaliação do que foi feito para cada nova Release necessária**.

É necessária muita disciplina para seguir esta abordagem. Seus princípios e valores são baseados em dedicação e bom senso de todos os envolvidos.

Suas **práticas** pregam inspeções constantes - para o **feedback rápido** – e aceitação das mudanças e adaptações que o processo devia passar. O comprometimento entre todos os envolvidos também se constitui como um grande diferencial do *framework*. O processo de gestão está permeado entre os diversos papéis existentes.

## DIRETO DO CONCURSO

- 011.** (FCC/TJ-AP/ANALISTA JUDICIÁRIO/ÁREA APOIO ESPECIALIZADO/TECNOLOGIA DA INFORMAÇÃO/DESENVOLVIMENTO DE SISTEMAS/2014) O Scrum é um framework ágil para suportar o desenvolvimento e manutenção de produtos complexos. Nesta abordagem,
- a)** a Equipe de Desenvolvimento é composta pelo *Product Owner*, o Time Scrum e o Scrum Master. A Equipe é auto-organizável e escolhe qual a melhor forma para completar seu trabalho.
  - b)** o Scrum Master é o responsável por maximizar o valor do produto e do trabalho do Time Scrum. É a única pessoa responsável por gerenciar o Backlog do Produto.
  - c)** o tamanho ideal do Time Scrum, responsável pela codificação dos testes, é de 3 componentes. Se houver mais de 5 integrantes é exigida muita coordenação e não vai funcionar.
  - d)** o trabalho a ser realizado na Sprint é planejado na reunião de planejamento da Sprint, que é um time-box de 8 horas para uma Sprint de 1 mês de duração. Para Sprints menores, este evento deve ser proporcionalmente menor.
  - e)** a Revisão da Sprint é uma oportunidade para o Time Scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima Sprint. Esta é uma reunião *time-boxed* de 5 horas para uma Sprint de 2 meses.



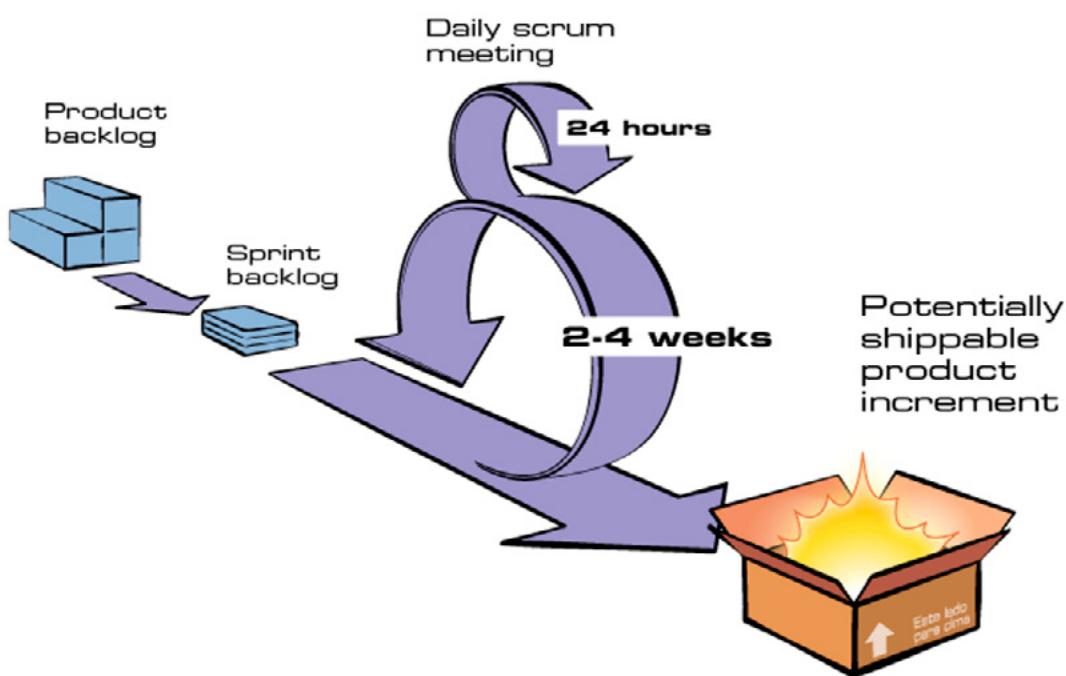
O Scrum é um processo de desenvolvimento de software iterativo e incremental. É um processo ágil que visa gerenciar projetos de difícil planejamento.

O Scrum não descreve as atividades que devem ser realizadas em cada situação. Ao invés disto, ele contém um conjunto de valores, princípios e práticas que fornecem a base para que uma empresa implemente suas atividades de engenharia e gestão.

Segundo <http://www.desenvolvimentoagil.com.br/scrum/>, no processo de desenvolvimento do Scrum, os projetos são organizados em iterações (Sprints). Um Sprint representa um intervalo no qual um conjunto de tarefas deve ser executado.

As funcionalidades a serem desenvolvidas no sistema são mantidas em uma lista chamada *Product Backlog*. Ao iniciar-se um Sprint, realiza-se uma reunião de planejamento (*Sprint Planning Meeting*), em que o *Product Owner* prioriza os itens do *Product Backlog* e a equipe de desenvolvimento identifica as atividades que ela será capaz de implementar durante o Sprint. Então, as tarefas alocadas no Sprint são transferidas do *Product Backlog* para o *Sprint Backlog*. Diariamente, em geral pela manhã, a equipe (Scrum Team) realiza uma reunião chamada Daily Scrum, visando relatar as tarefas realizadas no dia anterior, identificar impedimentos e priorizar o trabalho do dia que se inicia.

Ao término do Sprint, a equipe de desenvolvimento apresenta as funcionalidades implementadas em uma reunião chamada *Sprint Review Meeting*. Adicionalmente, realiza-se uma *Sprint Retrospective* em que se avalia o trabalho realizado e a equipe parte para o planejamento do próximo Sprint.



Na questão, a única alternativa correta é a letra D, que cita a reunião de planejamento do Scrum (*Sprint Planning Meeting*) em que se definem as tarefas do Sprint, conforme citado anteriormente.

**Letra d.**

**012.** (CESPE/ANATEL/ANALISTA ADMINISTRATIVO/ARQUITETURA DE SOLUÇÕES DE TIC/2014) Acerca dos processos de desenvolvimento de software, julgue o item seguinte: O Scrum é um conjunto simples e eficaz de regras e ferramentas que são utilizadas para maximizar resultados. O Scrum Master exerce o papel de facilitador e motivador da equipe, além de garantir que as regras e as ferramentas sejam utilizadas com vistas à criatividade do trabalho e ao retorno do investimento.



**Scrum** é um *framework* de processo dentro do qual podem ser empregados processos e técnicas variadas. Trata-se de um conjunto simples e eficaz de regras e ferramentas que são utilizadas para maximizar resultados. Pode ser aplicado em qualquer contexto no qual um grupo de pessoas trabalhe junto para atingir algum objetivo.

Scrum Master é o responsável pela aplicação dos valores e práticas do Scrum. Remove obstáculos, facilita resultados, garante a plena funcionalidade e produtividade da equipe. Considerado como um “escudo para interferência externas”.

**Certo.**

**013.** (FCC/TRE-CE/ANALISTA JUDICIÁRIO/ANALISTA DE SISTEMAS/2012) No SCRUM, sprint é

- a) um representante dos stakeholders e do negócio.
- b) uma lista de requisitos que tipicamente vêm do cliente.
- c) uma lista de itens priorizados a serem desenvolvidos para um software.
- d) uma iteração que segue um ciclo (PDCA) e entrega incremento de software pronto.
- e) um conjunto de requisitos, priorizado pelo *Product Owner*.



O Scrum é um processo de desenvolvimento ágil de software baseado em grupos de práticas e papéis predefinidos. Ele é um processo **iterativo e incremental para gerenciamento de projetos e desenvolvimento de sistemas**, em que cada **sprint** é uma iteração que segue um ciclo PDCA (*Plan, Do, Check, Act*) e entrega um incremento de software pronto.

**Letra d.**

## KANBAN

### CONCEITOS BÁSICOS

**Kanban** é um método de desenvolvimento de software com fortes bases em práticas enxutas, e tem como objetivo otimizar o processo de desenvolvimento de software pré-existente.

Henrik destaca as seguintes **práticas** do Kanban:

- 1 - **Visualizar o fluxo de trabalho (workflow);**
- 2 - **Limitar o trabalho em progresso;**
- 3 - **Gerenciar e medir o fluxo.**

Segundo o autor, embora poucas, quando bem utilizadas essas práticas podem levar um time de desenvolvimento de software a ótimos resultados. David Anderson recomenda que se acrescente mais duas condições a essa lista de Kniberg, para se ter um processo ainda mais otimizado:

- 1 - Tornar políticas do processo explícitas;
- 2 - Usar modelos para avaliar Oportunidades de Melhoria.

Entende-se por **fluxo** a progressão visual dos itens de trabalho por meio do sistema Kanban.

Assim, uma de suas práticas é o **Gerenciamento do Fluxo**. Nesse contexto:

- visualize o fluxo de trabalho: divida o trabalho em partes, e use colunas nomeadas para ilustrar onde cada item se encontra no fluxo de trabalho;

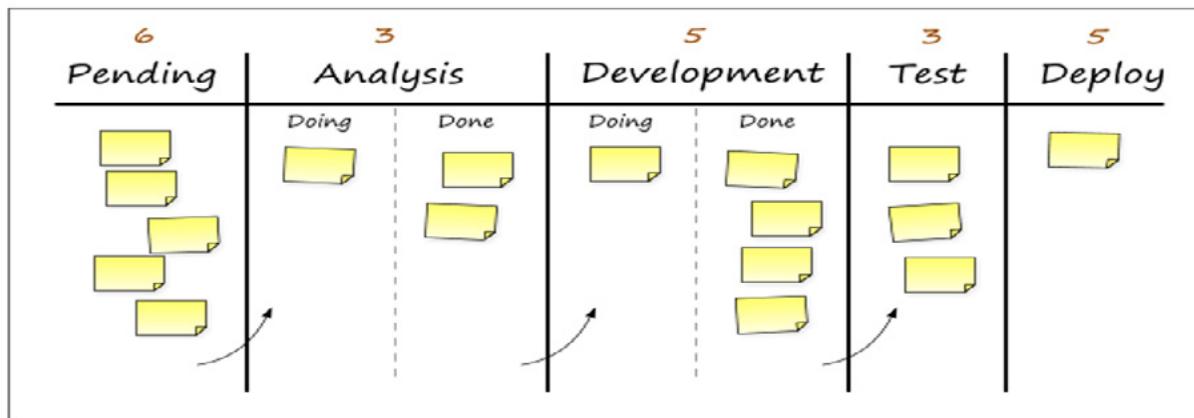


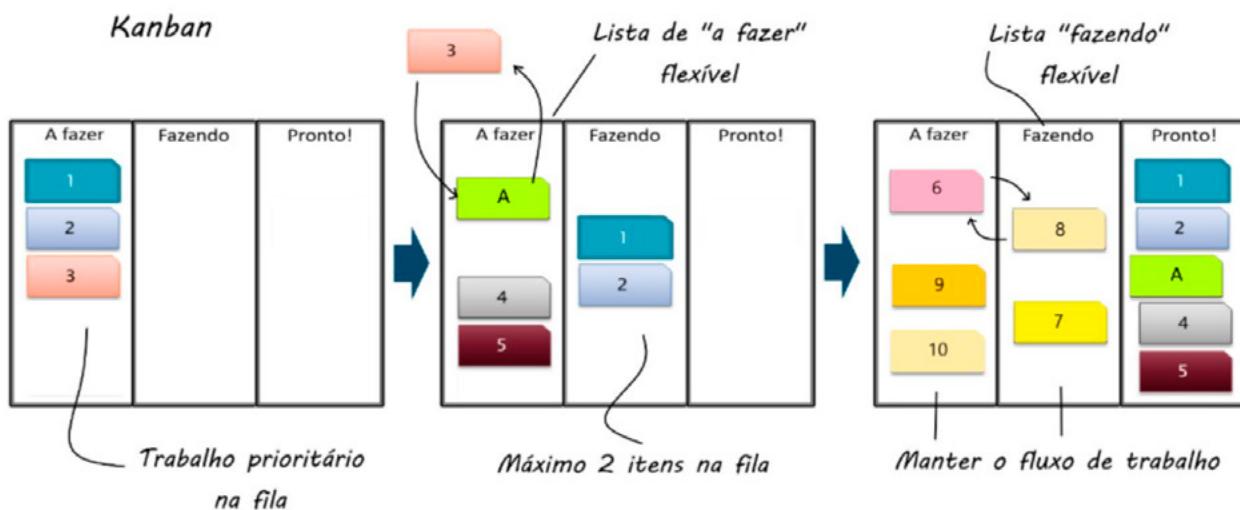
Figura. Quadro Kanban (Fonte: <http://www.kanbanblog.com/explained/>)

- limite o trabalho em progresso (*work in progress*), associando limites explícitos para quantos itens podem estar em progresso em cada estado do fluxo de trabalho;
- acompanhe o tempo de execução da tarefa e otimize o processo para tornar o tempo de execução o menor e mais previsível possível.

Segundo destaca Gomes (2013), “sempre que um item de trabalho parecer estar parado por muito tempo, sem fluir no processo, a equipe deve conversar a respeito para identificar a causa do problema, e então pensar em como melhorar a fluidez do processo. Entre as causas possíveis para **problemas** que podem levar um item de trabalho a não fluir no Kanban podemos citar: dificuldades técnicas, cliente ou analistas de negócio indisponíveis para esclarecer dúvidas de negócio da equipe, requisitos pouco claros, ambiente de homologação indisponível, longas esperas por informações, etc.”.

## FLUXO DE TRABALHO NO KANBAN

Conforme destaca Sotile (2018), “no Kanban, o **tamanho das filas**, chamado de limite de trabalho em processo, é **limitado**. Isso significa que os itens nas filas podem ser alterados a qualquer momento e que não há “final da *sprint*”. O trabalho continua fluindo.”



Fonte: Sotile (2018)

## SCRUMBAN

### CONCEITOS BÁSICOS

- **Scrumban (= Scrum + Kanban).**
- É baseado em sistema puxado, no qual a equipe já não planeja o trabalho durante a *sprint planning*, em vez disso, o refinamento é feito continuamente (SOTILE, 2018).
- Usa a natureza prescritiva do Scrum para ser ágil (SOTILE, 2018).
- Usa a melhoria de processo do Kanban para permitir que a equipe melhore continuamente seu processo (SOTILE, 2018).
- Usando o sistema puxado do Kanban o fluxo se torna mais suave à medida que a capacidade de processo melhora (SOTILE, 2018).
- O lead time médio e o tempo de ciclo se tornam o foco principal do desempenho. Se o tempo de ciclo estiver sob controle e a capacidade da equipe for equilibrada com a demanda, o tempo de espera também estará sob controle. Se o tempo de ciclo estiver sob controle, as burndowns são previsíveis e deixam de importar (SOTILE, 2018).
- Conforme destaca Sotile (2018), como a equipe agora coloca o trabalho em uma fila pequena e pronta antes de inseri-lo no trabalho em andamento, o *backlog* da iteração sempre contém algo para ser feito (fluxo ininterrupto).



Figura. Backlog (SOTILE, 2018)

- Em vez de se preocupar em estimar um escopo de trabalho para cada iteração, limita-se o tamanho do *backlog* de iteração **usando-se um tamanho fixo para o backlog**, o qual será executado até zero antes do término do intervalo de planejamento.
- Priorização sob demanda – o processo ideal de planejamento de trabalho deve sempre fornecer à equipe a melhor coisa para trabalhar a seguir, nem mais nem menos
- No Scrumban, podemos fazer o **planejamento de iteração** em intervalos regulares, sincronizados com revisão e retrospectiva, sendo que o objetivo do planejamento é preencher os espaços disponíveis e não preencher todos os espaços. Isso reduz muito a sobrecarga e a cerimônia do planejamento de iteração.

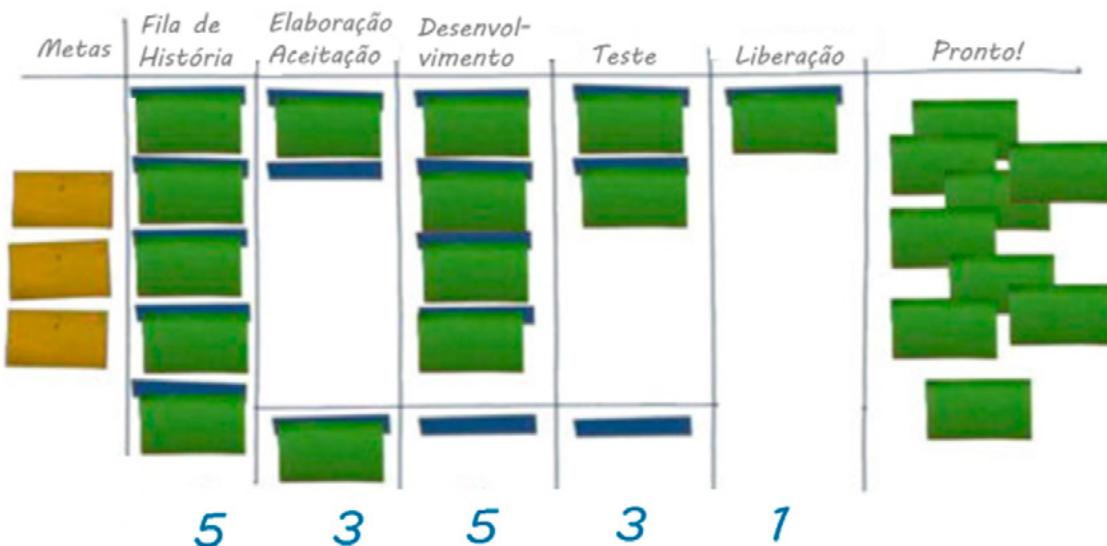


Figura. Exemplo de Scrumban Board (SOTILE, 2018)

- **Quando Usar Scrumban**, segundo Sotile (2018):
  - trabalho orientado a eventos, como *help desk* / suporte ou na fase de embalagem;
  - projetos com histórias de usuários frequentes e inesperadas ou erros de programação;
  - equipes focadas no desenvolvimento de novos produtos:
    - trabalho que antecede o desenvolvimento do *sprint* (*backlog*, pesquisa e desenvolvimento);
    - trabalho após o desenvolvimento do *sprint* (teste, empacotamento e implantação do sistema);
  - se o Scrum apresentar problemas de fluxo de trabalho, recursos e processos;
  - gerenciar comunidades de melhoria durante / após o início do uso do Scrum.
- **Algumas vantagens do Scrumban**, citadas em Sotile (2018):
  - *Just-in-time* (decisões e fatos justamente quando são necessários);
  - Prazo curto de entrega;
  - Kaizen (melhoria contínua);
  - Minimização de desperdício (tudo o que não agrega valor ao cliente);
  - Melhoria de processo, adicionando alguns valores do Scrum como e quando necessário.

## EXTREME PROGRAMMING (XP)

A **eXtreme Programming (XP)** faz parte também de uma série de métodos denominados **ágeis (agile)**. Estes métodos foram inicialmente considerados **leves (lightweight)** para diferenciá-los dos métodos tradicionais de desenvolvimento considerados pesados (*heavyweight*), os quais seriam baseados na produção de uma grande quantidade de documentação e modelos para guiar a programação.

Ao contrário dos métodos tradicionais que são orientados ao planejamento, previsibilidade e ao controle, os métodos ágeis são **orientados à construção, testes e principalmente, às pessoas**. As metodologias ágeis enfatizam os aspectos humanos, as relações pessoais, uma vez que buscam valorizar o pensamento criativo dos indivíduos envolvidos no projeto, em que o conhecimento é fator importante para o sucesso do projeto.

No desenvolvimento ágil a metodologia deve produzir conhecimento e não apenas documentação. Mas isto não significa que nos ambientes ágeis não exista documentação, apenas deixa de existir a filosofia de que “tudo tem que ser documentado” e sim documentar apenas o necessário uma vez que a documentação apenas auxilia e não guia o desenvolvimento.

Na próxima figura podemos comparar a XP ao modelo tradicional de desenvolvimento de software.

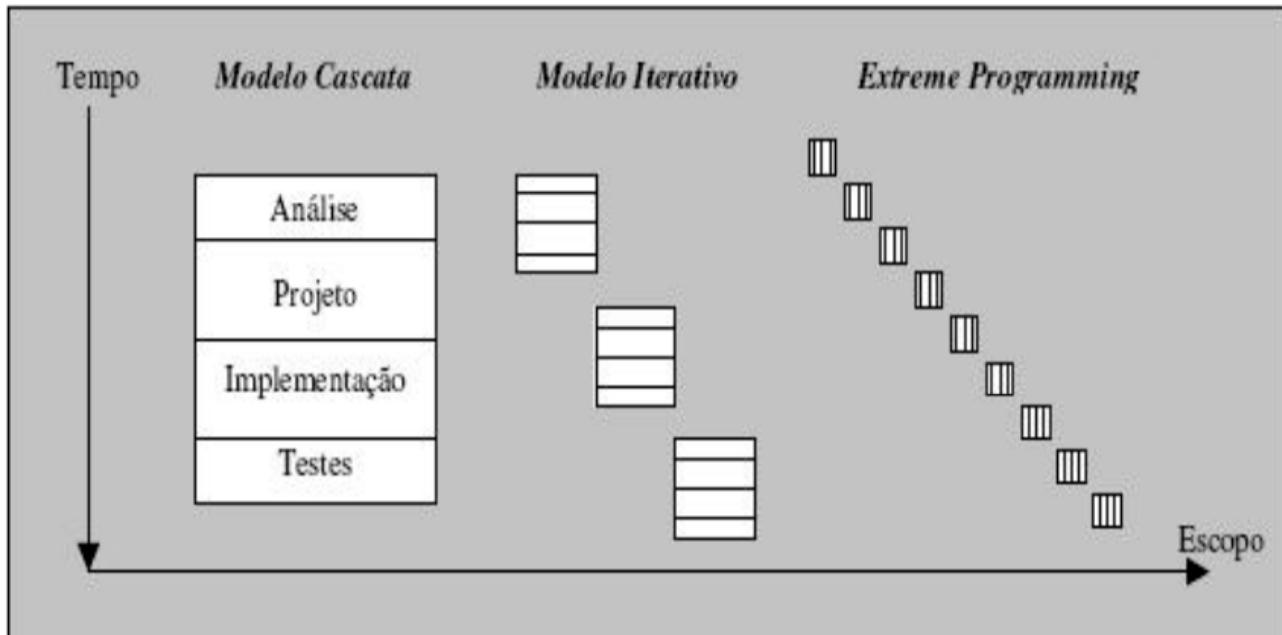


Figura. XP, Modelo Iterativo e Modelo Cascata

Como se vê na figura, o **modelo em cascata** estabelece uma ordenação linear no que diz respeito à realização das diferentes etapas, já o **modelo iterativo** é um modelo de processo de desenvolvimento que busca contornar algumas das limitações existentes no modelo cascata.

A XP trabalha com **iterações de menor tamanho possível, contendo os requisitos de maior valor para o negócio**, sendo assim, a equipe produz um conjunto reduzido de funcionalidades e coloca em produção rapidamente de modo que o cliente já possa utilizar o software no dia-a-dia e se beneficiar dele.

## VALORES

Para implantar a XP é necessário que se norteie o trabalho baseado em **cinco valores (Beck)**:

- **Comunicação:** é o principal valor da XP. Grande parte das técnicas da XP está baseada na comunicação, e se esta não for eficiente, pode causar problemas e atrasos no desenvolvimento do sistema.
- **Simplicidade:** a XP utiliza-se da simplicidade para implementar apenas aquilo que é suficiente para o cliente, não se procura fazer especulações sobre necessidades futuras, pois quase sempre são especulações errôneas, deixamos os problemas do futuro para o futuro.

- **Feedback:** o cliente deve receber o sistema o quanto antes, a fim de poder dar um *feedback* rápido, guiando assim o desenvolvimento do software.
- **Coragem:** é preciso muita **coragem (disciplina) para mudar a maneira pela qual desenvolve-se sistemas**. Colocar um sistema em produção assim que ele tenha valor para o cliente, fazer apenas o que se precisa para o momento e calcar o processo de análise principalmente na comunicação não é fácil, e precisa que a equipe esteja realmente decidida a mudar o seu processo de desenvolvimento.
- **Respeito:** a equipe ágil **busca respeito entre seus membros, outros envolvidos e os membros da equipe**, e, indiretamente, para o próprio software.

## PRÁTICAS

A XP é baseada em um conjunto de técnicas fundamentais (**práticas**), que podem ser aplicadas individualmente, cada uma delas gerando melhorias no processo de desenvolvimento, mas que funcionam melhor em conjunto, uma vez que uma técnica reforça o resultado da outra.

Apresenta-se a seguir uma descrição de cada prática.

- Cliente Presente (*On-site customer*)

**Na XP todas as decisões sobre o rumo do projeto devem ser tomadas pelo cliente.** Ele deve priorizar as tarefas, ser responsável pelos testes de aceitação, e, acima de tudo, orientar e tirar dúvidas dos desenvolvedores durante o processo de programação.

Se o cliente não puder estar junto dos desenvolvedores, algumas técnicas podem ser utilizadas, como, por exemplo, nomear um membro da equipe para representar o cliente. Também se podem agendar reuniões de acompanhamento com o cliente, pelo menos uma vez por semana. Nestas reuniões haverá muita discussão sobre prioridades, mas deve-se destinar uma parte significativa da mesma para que os programadores possam saber o que e como desenvolver.

### Jogo do Planejamento (The Planning Game)

**XP utiliza o planejamento para assegurar que a equipe de desenvolvimento esteja sempre trabalhando naquilo que irá gerar maior valor para o cliente a cada dia de trabalho.** Este planejamento é feito diversas vezes ao longo do projeto, para que todos tenham a oportunidade de revisar as prioridades.

Na XP o planejamento é um processo contínuo, e o mesmo é constantemente refinado pelo cliente e pela equipe de desenvolvimento, deixando assim a metodologia bastante flexível e entregando para o cliente sempre o máximo valor pelo investimento dele.

## Pequenos Lançamentos (Small Releases)

Para que o cliente possa fornecer constantemente *feedback* sobre o andamento do sistema, fazendo possível que o jogo do planejamento (*planning game*) seja executado, é importante que o sistema tenha *releases* pequenos, a fim de ajustar o desenvolvimento às necessidades que vão surgindo no decorrer do processo.

Normalmente, trabalha-se com pequenas iterações gerando *releases* mais curtos, mas algumas empresas vêm suprimindo a etapa das iterações, lançando direto um release por semana.

Quanto menor o intervalo de tempo entre cada versão que o cliente recebe, mais fácil será corrigir eventuais problemas, pois não terá sido entregue uma quantidade muito grande de funcionalidades, e mais fácil será fazer algum ajuste no software para atender a uma mudança de requisitos. Além disso, o XP recomenda que o cliente busque selecionar as funcionalidades que mais vão gerar valor para ele, com isso fazemos com que o cliente tenha um retorno de investimento o mais cedo possível.

## Desenvolvimento Guiado pelos testes (Test Driven Development- TDD)

O **desenvolvimento guiado pelos testes** define que qualquer método de um objeto que possa falhar deve ter um teste automatizado que garanta o seu funcionamento.

## Integração Contínua (Continuous Integration)

Quando uma equipe desenvolve um software é comum que o software seja “quebrado” em algumas partes a fim de que cada desenvolvedor implemente a sua parte, esta visão é interessante porque assim cada desenvolvedor tende a se preocupar só com a sua parte do sistema o que reduz a complexidade do problema. Neste caso são definidas interfaces para comunicação entre estas partes para posterior agrupamento e formação de um software coeso. Porém é muito comum acontecerem erros na integração, estes erros acontecem geralmente por que:

- as interfaces definidas não foram respeitadas;
- não houve compreensão por parte de um desenvolvedor da interface do outro;
- o sistema como um todo é pouco coeso.

Devido a estes erros e também da instabilidade das interfaces, por estarem sempre em constante mudança, a integração contínua se faz necessária para amenizar e até suprimir esses erros, uma vez que expõe todo o estágio atual de desenvolvimento, oferece *feedback* constante e estimula agilidade no desenvolvimento do software.

A técnica de integração contínua diz que o código desenvolvido por cada par de desenvolvedores deve ser integrado ao código base constantemente.

## Projeto Simples (Simple Design)

Kent Beck utiliza **quatro regras básicas** para definir **simplicidade**, são elas em ordem de prioridade:

1. O sistema (código e testes em conjunto) deve expressar tudo aquilo que você deseja comunicar.
2. O sistema não deve conter nenhum código duplicado.
3. O sistema deve ter o menor número de classes possível.
4. O sistema deve ter o menor número de métodos possível.

## Refatoração (Refactoring)

É o processo de **mudar um sistema de software de tal forma que não se altera o comportamento externo do código, mas melhora sua estrutura interna**. É um meio disciplinado de limpar o código e que reduz as possibilidades de introduzir erros. Em essência quando se refina o código, você melhora o projeto depois que este foi escrito. “Melhorar o projeto depois que foi escrito” é uma premissa do XP.

A técnica de refinamento do *design* é utilizada sempre com o intuito de simplificar o código. **Refactoring** significa **reescrever o código, melhorando e simplificando o mesmo, sempre que se tiver oportunidade**. O projeto do código vai sendo aos poucos melhorado por meio de modificações que o aprimorem. Estas modificações partem de um código fonte o qual passa em todos os testes, e devem levar a um código-fonte que continue passando em todos os testes.

## Programação em pares (Pair Programming)

Todo o código que vai para a produção é escrito por um par de programadores que utilizam uma mesma estação de trabalho ao mesmo tempo, ou seja, um computador, um teclado e dois desenvolvedores.

Na XP todo o código deve ser produzido por duas pessoas utilizando o mesmo computador. Enquanto um dos parceiros se preocupa com detalhes da implementação, ficando responsável pela digitação do código, o outro deve tentar ter uma visão mais ampla da rotina, imaginando as suas peculiaridades. Não apenas o código deve ser produzido por duas pessoas, como também todo o projeto da classe na qual vai se trabalhar.

## Propriedade Coletiva (Collective Ownership)

O código deve ser de propriedade de todos e todos devem ter permissão para alterar o que for necessário para que seu trabalho possa ser desenvolvido. Em estruturas onde determinadas rotinas são de “propriedade” de algumas pessoas, podem ocorrer atrasos no desenvolvimento devido à necessidade de que seja alterado algo nestas rotinas para que outras pessoas possam continuar com o seu trabalho.

## Padrões de Codificação (Coding Standards)

XP recomenda a utilização de padrões de codificação, que devem ser adotados no início do projeto com o consentimento de todos os desenvolvedores. Desta forma, qualquer desenvolvedor poderá ler o código com velocidade, sem se preocupar em compreender estilos de formatação especiais.

## Ritmo Sustentável (40 Hour Week)

Um dos grandes problemas que projetos de desenvolvimento de software enfrentam é o curto prazo de entrega do sistema. Com um prazo cada vez mais curto e sistemas cada vez mais complexos uma das alternativas dos desenvolvedores é o trabalho em excesso. As pessoas passam a trabalhar diariamente até mais tarde invadindo também finais de semana e feriados, porém trabalhar por longos períodos é contraproducente, e normalmente tende a piorar a situação.

## Metáfora (Metaphor)

Equipes XP mantêm uma visão compartilhada do funcionamento do sistema. Isto é feito por meio do uso de metáforas. Fazendo uso de metáforas podemos facilitar a explicação de qualquer aspecto dentro do projeto e ao mesmo tempo fixá-la com mais força na mente do ouvinte.

Utilizam-se metáforas para vários aspectos na XP, dentre eles procurase criar uma visão comum sobre todo o projeto de forma simples e objetiva facilitando assim a comunicação entre toda a equipe, uma vez que serve de base para os padrões de codificação e entre a equipe e o cliente, visto que o cliente, na maioria das vezes, não entende os termos técnicos relacionados ao desenvolvimento e usados diversas vezes pelos desenvolvedores e equipe.

## DIRETO DO CONCURSO

**014. (CESPE/ANATEL/ANALISTA ADMINISTRATIVO/ARQUITETURA DE SOLUÇÕES DE TIC/2014)** Acerca dos processos de desenvolvimento de software, julgue o item seguinte: A etapa de planejamento do Extreme Programming (XP) inicia-se com a escrita de UserStories (história do usuário). Por meio dessa ferramenta, aqueles que conhecem a técnica de construção de uma solução poderão guiar quem necessita dessa solução no exercício de descrevê-la de forma simples e concisa.



XP é considerada uma metodologia ágil para equipes pequenas e médias desenvolvendo software com requisitos vagos e em constante mudança” – Kent Beck

Segundo IBM, uma *user story* (História do usuário) é um requisito capturado normalmente em 1 parágrafo que descreve a necessidade de um usuário de forma breve utilizando uma linguagem comum ao negócio. A etapa de planejamento do XP inicia-se com a escrita de User Stories. Por meio dessa ferramenta, aqueles que conhecem a técnica de construção de uma solução poderão guiar quem necessita dessa solução no exercício de descrevê-la de forma simples e concisa.

**Certo.**

**015. (FCC/TRT 9ª REGIÃO/PR/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2013)**

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:

	<b>XP</b>	<b>Scrum</b>	<b>FDD</b>
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



A relação correta é a seguinte:

### eXtreme Programming – XP (Programação Extrema)

- I – Programação em pares, ou seja, a implementação do código é feita em dupla. Essa é a prática mais famosa do XP.  
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. O sistema é concebido a partir de uma *metáfora* e são descritos em *histórias do usuário*. Uma metáfora é a transposição de uma conceitualização do mundo real para o sistema a ser desenvolvido. Por exemplo, os programas de correio eletrônico foram construídos utilizando os conceitos de mensagem, caixa de entrada e caixa de saída. Cada mensagem possui remetente, destinatário, assunto e cópias carbono (cc). Este modelo conceitual reflete a forma como correspondências são enviadas nos escritórios e pelo sistema de correio dos Estados Unidos. A metáfora passa a ser fundamental para a elaboração das histórias de usuários.  
VI – Tem como único artefato de projeto os cartões CRC (Classes, Responsabilidades e Colaborações). O uso de cartões CRC é recomendado de forma a permitir o design em equipe. Cartões CRC permitem a descrição dos conceitos identificados na metáfora na forma de classes. Responsabilidades são identificadas para cada classe. As colaborações determinam as interações entre classes. Os cartões permitem que o todo o time possa colaborar com o design.

### Scrum

- II – Desenvolvimento dividido em **ciclos iterativos** de até 30 dias chamados de **sprints**.  
V – O ciclo de vida é baseado em três fases: pre-game phase, game-phase, post-game phase. Aqui estamos nos referindo às fases do Scrum.  
VII – Realiza reuniões diárias de acompanhamento (*Daily meetings*) de aproximadamente 15 minutos.  
IX – Os requisitos são descritos em um documento chamado **backlog** e são ordenados por prioridade.

### Desenvolvimento Guiado por Funcionalidades (do inglês, *Feature Driven Development*, ou FDD)

- VIII – Define **seis marcos** durante o projeto e a implementação de uma **funcionalidade (feature)**. Os marcos são: walkthroughs do projeto, projeto, inspeção do projeto, codificação, inspeção de código e progressão para construção.

**Letra b.**

## TEST DRIVEN DEVELOPMENT (TDD)

O **TDD** (**Test Driven Development** ou **Desenvolvimento Orientado por Testes**) consiste, basicamente, em teste e implementação, teste e implementação e teste e implementação. Desenvolvemos o software baseado em testes que são escritos antes do código de produção!

A primeira missão no TDD é identificar uma nova funcionalidade. Isso tendo sido feito, imediatamente é aplicado um teste. Em seguida, se estiver tudo ok, é implementada efetivamente a nova funcionalidade.

TDD é parte da metodologia XP e também utilizado em diversas outras metodologias, além de poder ser utilizada livremente.

Os testes, no TDD, devem seguir uma representação intitulada **modelo FI-R-S-T**.

- **F (Fast)** - **Rápidos**: devem ser rápidos, pois testam apenas uma unidade;
- **I – (Isolated)** - Testes unitários são **isolados**, testando individualmente as unidades e não sua integração;
- **R (Repeatable)** - **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.

O Ciclo de desenvolvimento orientado a Testes (TDD) é bem simples:

- cria-se um teste;
- executa-se o teste, que por sua vez irá falhar, pois a funcionalidade não foi implementada ainda;
- codifica-se de modo a fazer o teste passar;
- executa-se o teste novamente e, caso obtenha sucesso, prossiga para o próximo passo;
- refatore o código.



Figura. Ciclo de TDD. Fonte: Pícolo (2017)

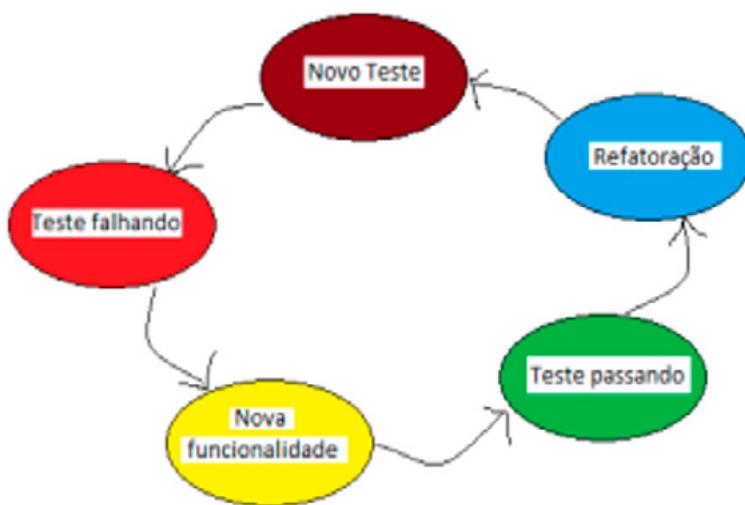
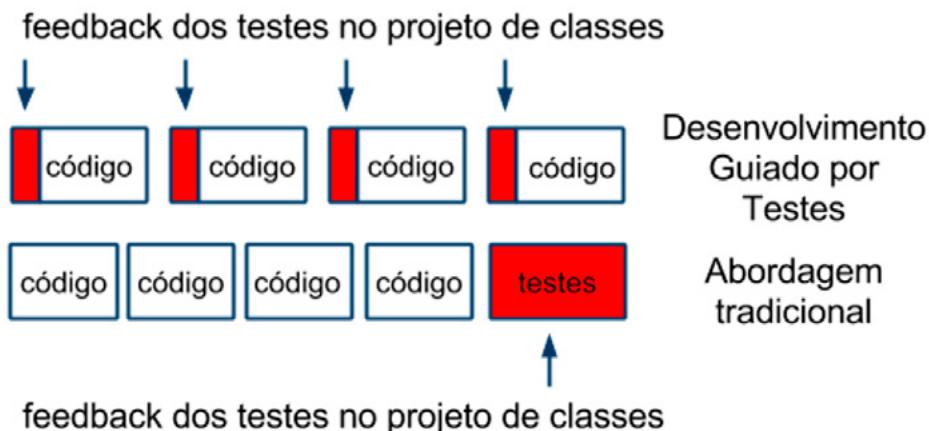


Figura. Ciclo de Desenvolvimento no TDD

Fonte: DevMedia (2018)

A próxima figura exemplifica a diferença entre a quantidade de *feedback* de um desenvolvedor que pratica TDD e de um desenvolvedor que escreve testes ao final.



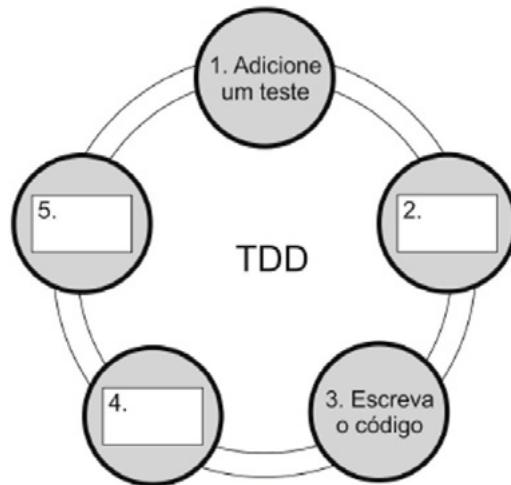
Fonte: Caelum (2020)

Temos diversos **ganhos** com o **TDD**, segundo DevMedia (2018):

- **feedback rápido sobre a nova funcionalidade** e sobre as outras funcionalidades existentes no sistema;
- **código mais limpo**, pois são escritos códigos simples para o teste passar;
- **segurança no refactoring** pois podemos ver o que estamos ou não afetando;
- **segurança na correção de bugs**;
- **maior produtividade**, pois o desenvolvedor encontra menos bugs e não desperdiça tempo com depuradores; etc.

## DIRETO DO CONCURSO

**016.** (FCC/ANALISTA JUDICIÁRIO/TRE-PR/APOIO ESPECIALIZADO/ANÁLISE DE SISTEMAS/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”.



A figura seguinte foi utilizada como referência nessa questão.



*Figura. Ciclo de TDD. Fonte: Pícolo (2017)*

O conteúdo deste livro eletrônico é licenciado para 61984693488 Martins Rodrigues - 00193743132, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

Observe que o **ciclo de desenvolvimento orientado a Testes** (TDD) é bem simples:

- cria-se um teste;
- executa-se o teste, que por sua vez irá falhar, pois a funcionalidade não foi implementada ainda;
- codifica-se de modo a fazer o teste passar;
- executa-se o teste novamente e, caso obtenha sucesso, prossiga para o próximo passo;
- refatore o código.

Vamos às assertivas:

- a) Errada. O passo 2 corresponde a “*Execute o teste e observe o resultado*”
- b) Errada. O passo 4 está descrito como “*Execute os testes automatizados*”.
- c) Errada. O passo 5 corresponde a “*Refatore os códigos*”.
- d) Certa. O passo 4 corresponde a “*Execute os testes automatizados*”.
- e) Errada. O passo 5 corresponde a “*Refatore os códigos*”.

**Letra d.**

---

**017.** (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



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.

Conforme visto, a letra D é a resposta.

**Letra d.**

**018.** (CESPE/MPOG/TECNOLOGIA DA INFORMAÇÃO/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.



**TDD** procura entender as regras de negócio e como elas devem estar refletidas no código e no modelo de domínio. **Cria-se com o TDD o mínimo de acoplamento.** Segundo destaca <http://www.agileandart.com/2010/07/16/ddd-introducao-a-domain-driven-design/>, com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos.

**Errado.**

**019.** (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.



A rotina dos desenvolvedores, ao empregar os processos baseados no *Test-Driven Development* (TDD) é concentrada na elaboração de **testes unitários**. Nesse caso, <https://klauslaube.com.br/2011/02/02/bdd-desenvolvimento-orientado-comportamento.html> destaca 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**”, onde partimos escrevendo testes específicos (unitários) e depois vamos abrangendo outras regiões do sistema (integração, funcional, aceitação, etc.)”.

**Obs.:** **Teste unitário:** utilizado para **validar as classes básicas e os componentes do sistema** que são considerados os **menores** elementos testáveis. Consiste em verificar se o fluxo de controle e dados estão corretos. Deve ser realizado no **início da iteração**.

- São escritos pelos desenvolvedores enquanto codificam o sistema.
- Devem ser feitos de modo que sejam fáceis de executar e re-executar várias e várias vezes para validar o sistema.
- Devem ser criados para todas as classes do sistema.
- São implementados para todos os métodos do sistema.
- São escritos antes e ao decorrer da produção do sistema.
- Devem ser o mais simples possível.

Referência: Pícolo (2017)

**Errado.**

## BEHAVIOUR-DRIVEN DEVELOPMENT (BDD)

BDD é a sigla utilizada para ***Behaviour-Driven Development*** (um modelo de desenvolvimento baseado no comportamento do software como um todo, de acordo com o cenário abordado).

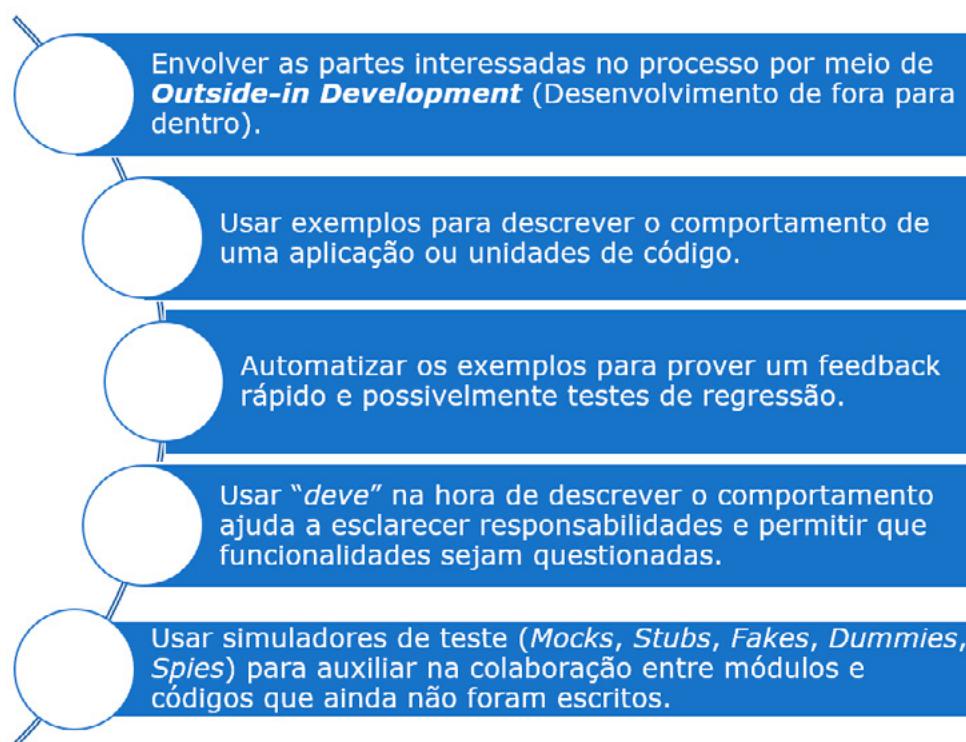
Para isso, é fácil criar testes que simulam esses cenários e verificar o seu comportamento. Trata-se de uma evolução do *Test Driven Development* (TDD) e do *Acceptance Test Driven Development* (ATDD). Essa metodologia de desenvolvimento ágil busca tornar as práticas destas outras metodologias mais acessíveis e intuitivas para os novatos e especialistas.

BDD colabora para que o desenvolvimento foque na **entrega de valor**, por meio da formação de um vocabulário comum, reduzindo a distância entre negócio e Tecnologia.

BDD está fundamentada em **três princípios simples** (ELEMARJR,2012):

1	2	3
<ul style="list-style-type: none"> <li>Negócio e Tecnologia deveriam “falar” sobre um sistema da mesma forma.</li> </ul>	<ul style="list-style-type: none"> <li>Qualquer sistema deveria ter um valor identificável e verificável para o “negócio”.</li> </ul>	<ul style="list-style-type: none"> <li>Análise, <i>design</i> e planejamento precoce tem, sempre, retorno questionável.</li> </ul>

**Principais práticas relacionadas ao BDD:**



Fonte: (<https://www.goconqr.com/pt/mindmap/7882459/behaviour-driven-development-bdd->)

**Obs.:** 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.

BDD se apoia no uso de um vocabulário pequeno e bem específico. Dessa forma, minimiza “ruídos” na comunicação de modo que todos os interessados – tanto de TI, quanto do negócio – estejam alinhados (ELEMARJR,2012).

Por apoiar todos os envolvidos e reduzir riscos de desenvolvimento inadequado, a “venda da ideia” para BDD costuma ser mais fácil do que para TDD. Embora, por envolver mais gente, seja mais “trabalhoso” de implantar.

**Obs.:** **Esquematizando...**

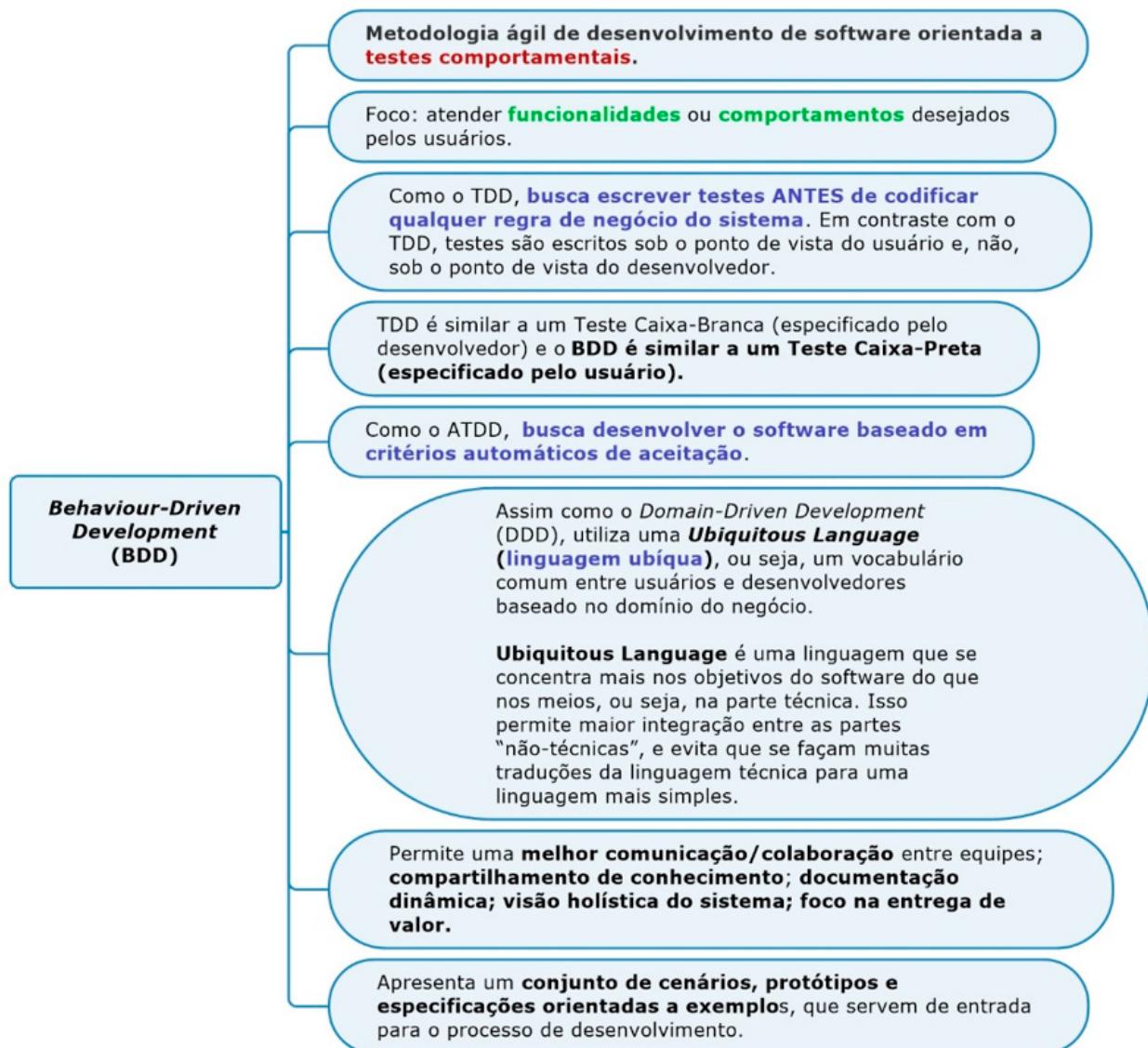


Figura. BDD. Fonte: Quintão (2021)

## DIRETO DO CONCURSO

**020.** (CESPE/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*).



**Ubiquitous Language** (ou **Linguagem Ubíqua**) é uma linguagem que se concentra mais nos objetivos do software do que nos meios, ou seja, na parte técnica. 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 **BDD** (*Behaviour-Driven Development*).

**Letra a.**

**021.** (IESES/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, portanto só pode ser feito quando o software estiver concluído.



- a) Certa. **BDD** (*Behaviour-Driven Development*) **utiliza uma linguagem ubíqua**, isto é, um vocabulário comum entre usuários e desenvolvedores baseado no domínio do negócio.
- b) Errada. **BDD** não é uma técnica de desenvolvimento de testes. É a sigla utilizada para **Behaviour-Driven Development** (um modelo de desenvolvimento **baseado no comportamento do software** como um todo, de acordo com o cenário abordado).
- c) Errada. **BDD** não é uma técnica de desenvolvimento de testes. Trata-se de uma **metodologia ágil de desenvolvimento de software orientada a testes comportamentais!** Os testadores, por exemplo, são muito importantes nos testes. Também participam os demais membros da equipe, e até os usuários.

d) Errada. Como o TDD, **BDD busca escrever testes ANTES de codificar qualquer regra de negócio do sistema.** Os testes são feitos dentro das iterações no desenvolvimento, e não deixados para o final. Em contraste com o TDD, **testes são escritos sob o ponto de vista do usuário** e, não, sob o ponto de vista do desenvolvedor.

**Letra a.**

---

**022. (FCC/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



Como vimos, o diferencial é que **BDD – Behaviour-Driven Development** - aborda o **comportamento** e o **resultado** como um todo, não se preocupando, necessariamente, com as classes, métodos e atributos de forma isolada.

**Letra c.**

---

## DOMAIN-DRIVEN DESIGN (DDD)

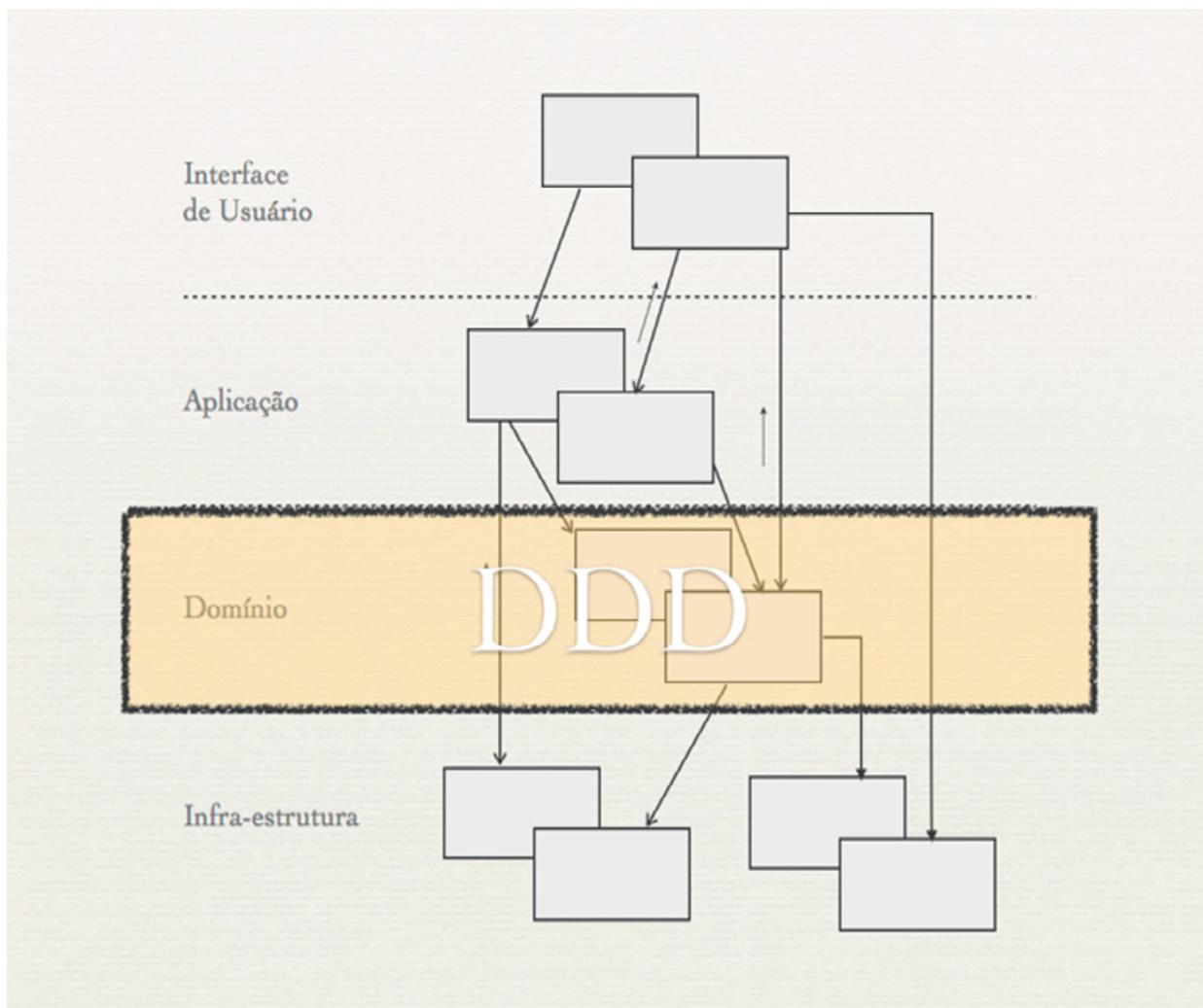
**Domain-Driven Design (DDD)** ou **Projeto Orientado a Domínio** é um padrão de modelagem de software orientado a objetos que procura **reforçar conceitos e boas práticas relacionadas à Orientação a Objetos (OO)**, como:

- alinhamento do código com o negócio;
- favorecer a **reutilização** (os blocos de construção facilitam aproveitar um mesmo conceito de domínio ou um mesmo código em vários lugares);

- mínimo de **acoplamento** (com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos);
- **independência** da tecnologia.

Além dos conceitos de OO, **DDD** (ou **Projeto Orientado a Domínio**) baseia-se em **duas premissas** principais:

- o foco principal deve ser o **domínio**;
- domínios complexos devem estar baseados em um **modelo**.



*Figura. Arquitetura em camadas, utilizada para separar o domínio do resto da aplicação. Fonte: Cukier (2010)*

Conforme destaca Cukier (2010), 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, intitulada **camada**

**anti-corrupção.** 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.

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

---

## DIRETO DO CONCURSO

**023.** (CESPE/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.



a) Errada. Esse item descreve o princípio de encapsulamento da Programação Orientada a Objetos. Esse princípio não está relacionado à **Domain-Driven Design (DDD)** ou **Projeto Orientado a Domínio**. A DDD procura reforçar conceitos e boas práticas relacionadas à Orientação a Objetos (OO), como:

- alinhamento do código com o negócio;
- favorecer a **reutilização** (os blocos de construção facilitam aproveitar um mesmo conceito de domínio ou um mesmo código em vários lugares);
- mínimo de **acoplamento** (com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos);
- **independência** da tecnologia.

b) Certa. Isso mesmo! DDD 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) Errada. O *Domain-Driven Design* (DDD) utiliza uma **Ubiquitous Language** (**linguagem ubíqua**), ou seja, um vocabulário comum entre usuários e desenvolvedores baseado no domínio do negócio.

**Obs.:** **Ubiquitous Language** é uma linguagem que se concentra mais nos objetivos do software do que nos meios, ou seja, na parte técnica. Isso permite maior integração entre as partes “não-técnicas”, e evita que se façam muitas traduções da linguagem técnica para uma linguagem mais simples.

d) Errada. Esse item está relacionado à Programação Orientada a Eventos e, não, a Domínio.

e) Errada. Esse item está relacionado à Programação Orientada a Objetos.

**Letra b.**

**024.** (CESPE/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.



Conforme visto, tem-se nesse contexto a **camada anti-corrupção**, responsável por traduzir e adaptar as chamadas para o sistema legado, usando uma fachada interna, ou seja, ela funciona como um adaptador.

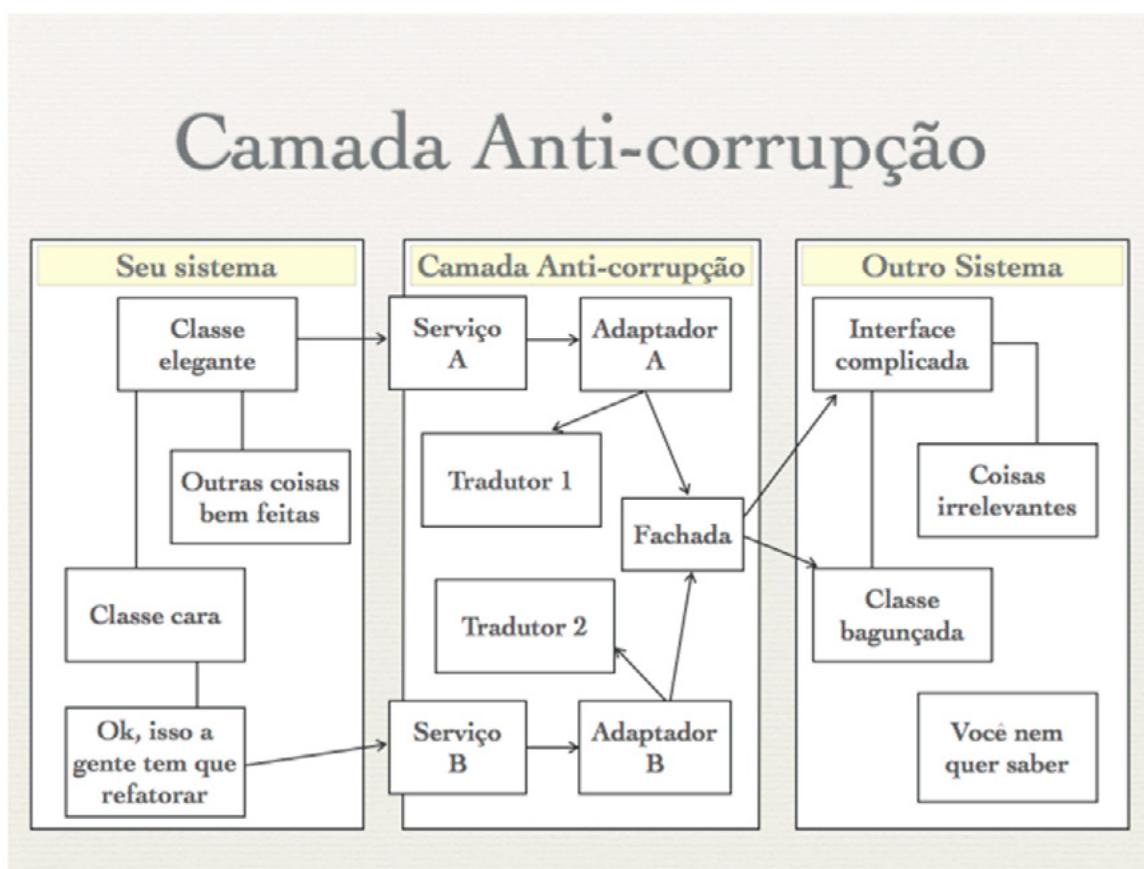


Figura. Camada Anti-corrupção

Fonte: Cukier (2010)

**Certo.**

**025.** (CESPE/ 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.



O *Domain-Driven Design* (DDD) utiliza uma **Ubiquitous Language** ([linguagem ubíqua](#)), ou seja, um vocabulário comum entre usuários e desenvolvedores baseado no domínio do negócio.

**Certo.**

**026.** (CESPE/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.



**Domain-Driven Design** ou **Projeto Orientado a Domínio** é um padrão de modelagem de software orientado a objetos que procura reforçar conceitos e boas práticas relacionadas à OO. Além dos conceitos de OO, DDD baseia-se em duas premissas principais:

- o foco principal deve ser o **domínio** (a lógica de negócios do software, seu campo de ação/atuação);
- domínios complexos devem estar baseados em um **modelo**.

**Certo.**

**027.** (CESPE/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.



O **Domain-Driven Design (DDD)** utiliza uma **Ubiquitous Language** ([linguagem ubíqua](#)), ou seja, um vocabulário comum entre usuários e desenvolvedores baseado no domínio do negócio.

**Certo.**

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



Isso mesmo! O *Domain-Driven Design (DDD)* 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.

**Certo.**

## METODOLOGIAS ÁGEIS COM DEVOPS

**Metodologias ágeis** são abordagens de desenvolvimento com foco na **entrega rápida** e **incremental de soluções** para o usuário. Possuem o foco em manter o nível de produtividade do time de desenvolvimento elevado para a rápida entrega de valor para o cliente. Já o **DevOps** possui o foco no aumento da eficiência por meio da **integração contínua** e da **automação de processos** (ENAP,2020).

Ao observar o foco das duas abordagens, é possível identificar que **ambas tratam da questão da eficiência para entrega rápida de valor** (ENAP,2020). Portanto, **ao serem adotadas de forma conjunta**, é possível **capturar os pontos fortes de cada abordagem, resultando em um elevado nível de entrega de valor para o usuário** (ENAP,2020).

## RESUMO

### DevOps

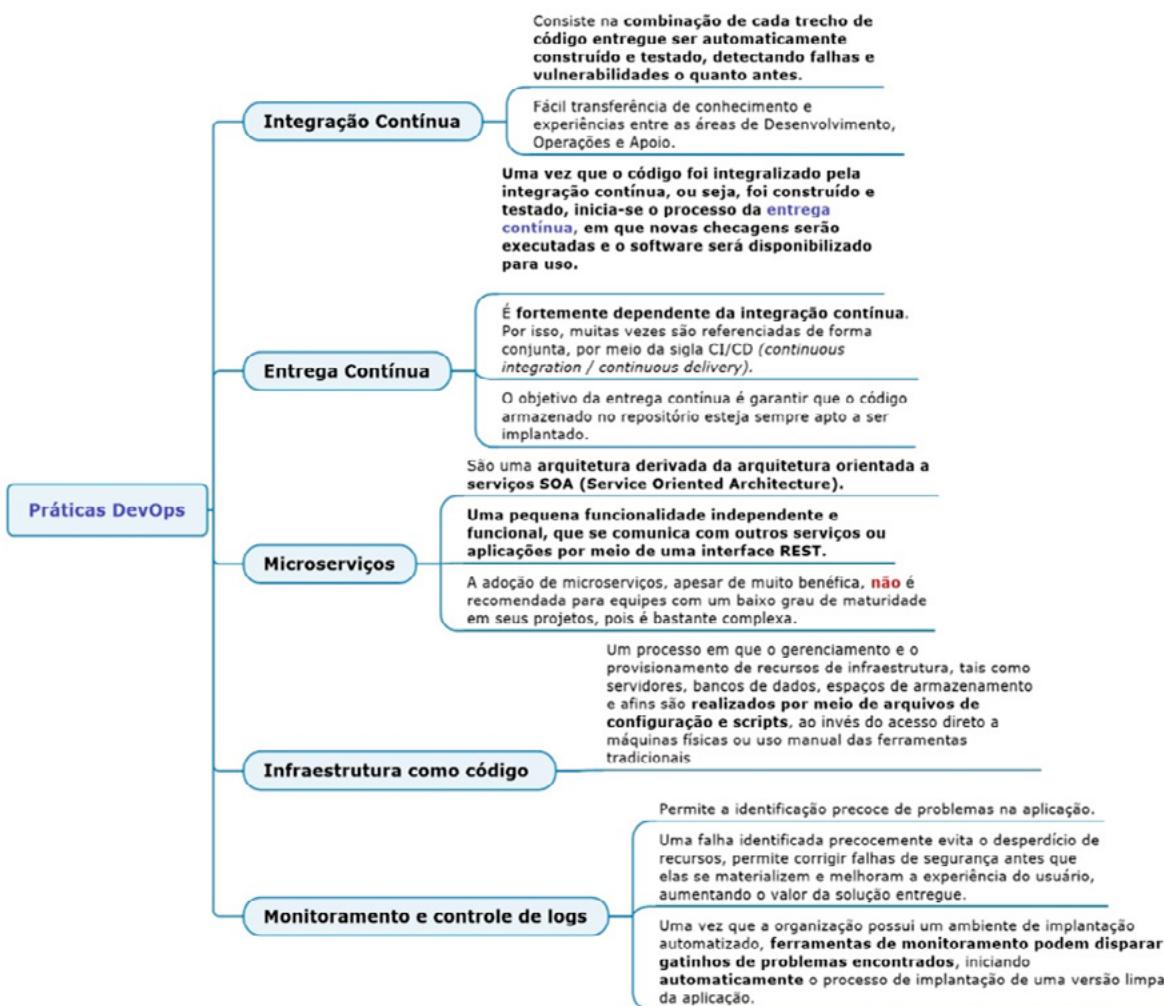


Figura. Práticas DevOps (QUINTÃO, 2023)

### Lean



Figura. Lean (QUINTÃO, 2023)

O conteúdo deste livro eletrônico é licenciado para 61984693488 Martins Rodrigues - 00193743132, vedada, por quaisquer meios e a qualquer título, a sua reprodução, cópia, divulgação ou distribuição, sujeitando-se aos infratores à responsabilização civil e criminal.

## eXtreme Programming

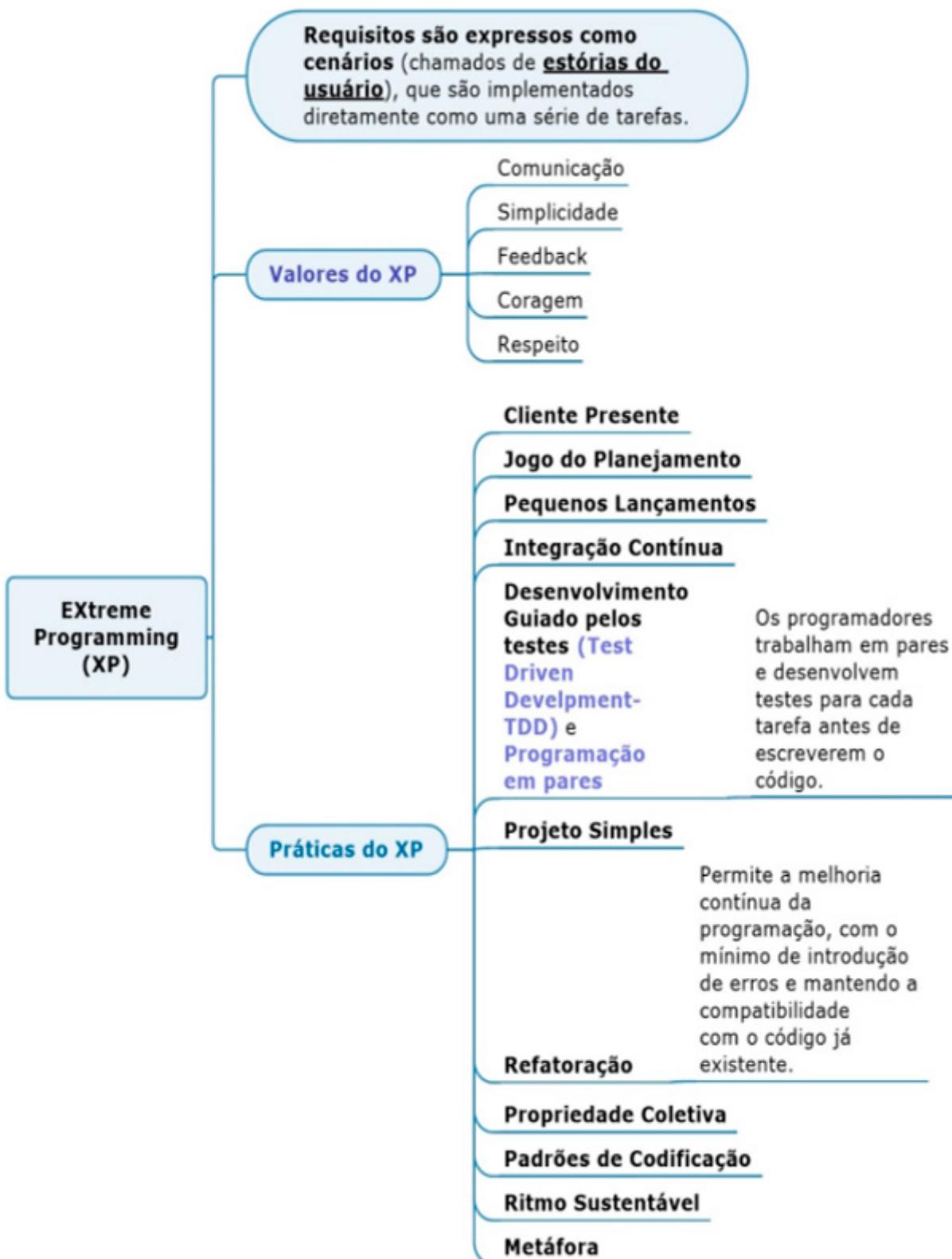


Figura. eXtreme Programming (XP)

## Scrum



Figura. Scrum (QUINTÃO, 2023)

Papéis do Scrum	
Product Owner (PO)	Dono do Produto. É uma pessoa e não um comitê. Ele pode representar o desejo de um comitê no Product Backlog, mas aqueles que quiserem uma alteração nas prioridades dos itens de backlog devem convencer o Product Owner.
Scrum Master (SM)	Mestre Scrum. É responsável por garantir que o Scrum seja entendido e aplicado! Ele ajuda a treinar o time de desenvolvimento em autogerenciamento e interdisciplinaridade.
Development Team (Dt)	Equipe de Desenvolvimento. Profissionais que realizam o trabalho de entregar uma versão usável que potencialmente incrementa o produto “pronto” ao final de cada sprint.

O framework possui apenas três artefatos oficiais, que são:

Artefatos Oficiais	Descrição
Product Backlog	Uma lista ordenada (por valor, risco, prioridade, etc.) de requisitos ou funcionalidades que o produto deve conter criada pela Equipe Scrum.
Sprint Backlog	Conjunto de itens selecionados para serem implementados durante a sprint mais o plano para transformálos em um incremento.
Product Increment	Ao final de cada sprint, a equipe de desenvolvimento entrega um incremento do produto – resultado do que foi produzido ao final do trabalho realizado na sprint.

## Scrum x Scrumban

	Scrum	Scrumban
Artefatos	Conselho, backlogs, burndowns	Somente conselho
Cerimônias	Scrum diário, planejamento da sprint, revisão da sprint, retrospectiva da sprint	Scrum diário (planejamento, revisão e retrospectiva conforme necessário)
Iterações	Sim (sprints)	Não (fluxo contínuo)
Estimativa	Sim	Não (tamanho similar)
Equipes	Deve ser multifuncional	Pode ser especializado
Papéis	Product Owner, Scrum Master, Equipe	Equipe + papéis necessários
Trabalho de equipe	Colaborativo, conforme as tarefas necessitem	Todos juntos para atingir objetivos
Trabalho em andamento	Controlado pelo conteúdo da sprint	Controlado pelo estado do fluxo de trabalho
Mudanças	Devem esperar pela próxima sprint	Adicionadas ao quadro conforme necessário
Backlog do produto	Lista de histórias priorizadas e estimadas	Cartões just-in-time
Impedimentos	Tratados imediatamente	Evitados

Fonte: Sotile (2018)

## Kanban x Scrumban

	Kanban	Scrumban
Papéis	Nenhum papel prescrito	Equipe + papéis necessários
Reunião diária de Scrum	Sem reuniões	Garantir trabalho contínuo nos requisitos e reduzir tempo ocioso dos membros da equipe
Reunião de Revisão e Retrospectiva	Não prescrito	Podem ser feitas conforme necessário para melhorar o processo e compartilhar aprendizados
Fluxo de Trabalho	Contínuo	Mesmo que no Kanban, porém limita os espaços para que o processo de “puxar” se torne mais confortável

Fonte: Sotile (2018)

## BDD

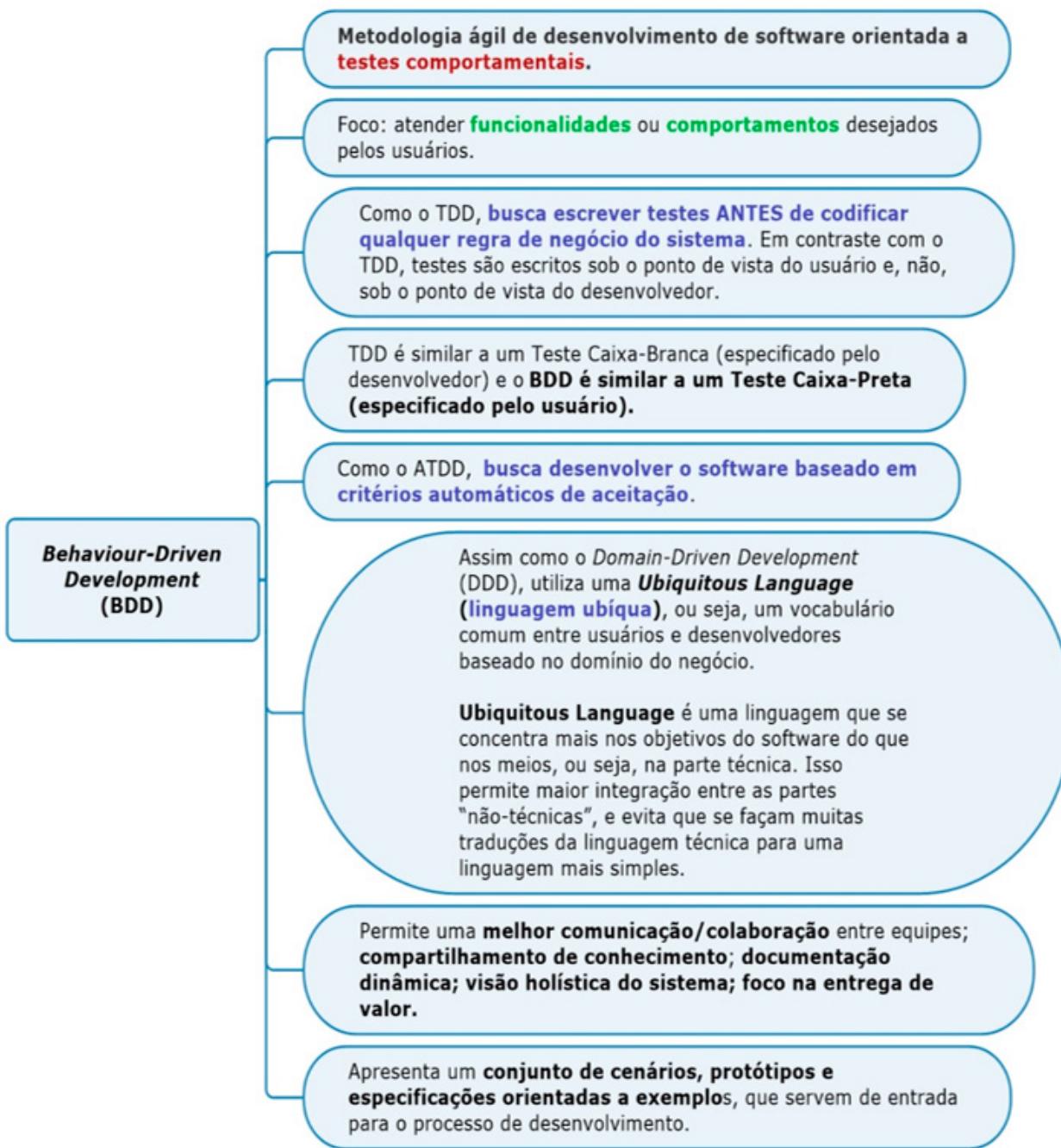


Figura. BDD. Fonte: Quintão (2021)

## QUESTÕES COMENTADAS EM AULA

**001.** (CESPE/BNB/ESPECIALISTA TÉCNICO/ANALISTA DE SISTEMA/2018) Julgue o item seguinte, a respeito de *web services*

*DevOps* é um conjunto de ferramentas e práticas de trabalho para integração entre os colaboradores dos grupos de desenvolvimento de código, de operações e de apoio, e pode ser utilizado na produção rápida e segura de aplicações e serviços.

**002.** (CESPE/2015/STJ/ANALISTA JUDICIÁRIO - DESENVOLVIMENTO) No que se refere à gestão de TI, julgue os itens a seguir.

O *DevOps*, movimento profissional emergente que defende uma colaboração maior entre desenvolvimento e operações de TI, resulta em um fluxo rápido do trabalho planejado, que aumenta a confiabilidade, a estabilidade e a segurança do ambiente de produção.

**003.** (CESPE/MPE-PI/ANALISTA MINISTERIAL/TECNOLOGIA DA INFORMAÇÃO/2018)

Acerca da gestão ágil de projetos com Scrum, de *DevOps* e da arquitetura corporativa (TOGAF), julgue o próximo item.

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.

**004.** (CESPE/SLU-DF/ANALISTA DE GESTÃO DE RESÍDUOS SÓLIDOS/INFORMÁTICA/2019)

Com relação a *DevOps* e TOGAF, julgue o seguinte item.

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.

**005.** (CESPE/STF/TÉCNICO JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2013) Quanto à gestão ágil de projetos com Scrum e às noções gerais de *DevOps*, julgue os itens subsecutivos.

O *DevOps* aplica abordagem ágil de desenvolvimento de software ao permitir que um negócio maximize a velocidade de entrega de um produto ou serviço.

**006.** (CESPE/STJ/TÉCNICO JUDICIÁRIO/SUPORTE TÉCNICO/2018) Julgue o item seguinte, a respeito de *DevOps* e das disposições constantes da NBR ISO/IEC 2700

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 permitem visualizar os fluxos do processo.

**007.** (FCC/SEFAZ-SC/AUDITOR-FISCAL DA RECEITA ESTADUAL/TECNOLOGIA DA INFORMAÇÃO/2018) No âmbito da infraestrutura, uma das vantagens *DevOps* é

- a)** realizar *deploy* sob controle manual de especialistas.
- b)** estabelecer uma divisão bem delimitada entre infraestrutura e desenvolvimento.
- c)** possuir um ambiente aberto (não padrão) sem muitos controles.
- d)** ter a infraestrutura como código.
- e)** fazer com que cada etapa do processo seja aprovada formalmente pelos gerentes.

**008.** (FUMARC/TÉCNICO JUDICIÁRIO/TRT 3<sup>a</sup> REGIÃO/APOIO ESPECIALIZADO/TECNOLOGIA DA INFORMAÇÃO/2022) Analise as afirmativas a seguir referentes aos conceitos e ferramentas de DevOps:

I – Um composto de Dev (desenvolvimento) e Ops (operações), DevOps é a união de pessoas, processos e tecnologias com foco na entrega rápida de serviços de TI.

II – CI/CD pode ser definido como um método para entregar aplicações com frequência aos clientes, baseado em integração e entrega contínuas.

III – Jenkins é uma ferramenta de integração contínua de código aberto, que pode ser usada para automatizar tarefas relacionadas à construção, teste, entrega e implantação de software.

Está CORRETO o que se afirma em:

- a)** I, apenas.
- b)** I e II, apenas.
- c)** I e III, apenas.
- d)** II e III, apenas.
- e)** I, II e III.

**009.** (ESAF/STN/ANALISTA DE FINANÇAS E CONTROLE/GOVERNANÇA E GESTÃO EM TI/2013) O desenvolvimento ágil de software fundamenta-se no Manifesto Ágil. Segundo ele deve-se valorizar:

- a)** mudança de respostas em vez do seguimento de um plano.
- b)** indivíduos e interações em vez de processos e ferramentas.
- c)** documentação extensiva operacional em vez de software funcional.
- d)** indivíduos e intenções junto a processos e ferramentas.
- e)** seguimento de um plano em vez de resposta a mudança.

**010.** (FGV/IMBEL/ANALISTA ESPECIALIZADO/ANALISTA ADMINISTRATIVO/ REAPLICAÇÃO/2021) A filosofia “Lean” é uma abordagem com foco em processos e que tem, como essência, a busca pela redução de desperdícios. Assinale a opção que apresenta um dos princípios basilares da filosofia “Lean”.

- a)** Melhoria pontual.
- b)** Maximização do uso de recursos.
- c)** Processamento “empurrado”.
- d)** Qualidade perfeita na segunda vez.
- e)** Construção de relacionamento de curto prazo com os fornecedores.

**011.** (FCC/TJ-AP/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/DESENVOLVIMENTO DE SISTEMAS/2014) O Scrum é um framework ágil para suportar o desenvolvimento e manutenção de produtos complexos. Nesta abordagem,

- a)** a Equipe de Desenvolvimento é composta pelo *Product Owner*, o Time Scrum e o *Scrum Master*. A Equipe é auto-organizável e escolhe qual a melhor forma para completar seu trabalho.
- b)** o *Scrum Master* é o responsável por maximizar o valor do produto e do trabalho do Time Scrum. É a única pessoa responsável por gerenciar o Backlog do Produto.
- c)** o tamanho ideal do Time Scrum, responsável pela codificação dos testes, é de 3 componentes. Se houver mais de 5 integrantes é exigida muita coordenação e não vai funcionar.
- d)** o trabalho a ser realizado na Sprint é planejado na reunião de planejamento da Sprint, que é um time-box de 8 horas para uma Sprint de 1 mês de duração. Para Sprints menores, este evento deve ser proporcionalmente menor.
- e)** a Revisão da Sprint é uma oportunidade para o Time Scrum inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima Sprint. Esta é uma reunião *time-boxed* de 5 horas para uma Sprint de 2 meses.

**012.** (CESPE/ANATEL/ANALISTA ADMINISTRATIVO/ARQUITETURA DE SOLUÇÕES DE TIC/2014) Acerca dos processos de desenvolvimento de software, julgue o item seguinte: O Scrum é um conjunto simples e eficaz de regras e ferramentas que são utilizadas para maximizar resultados. O *Scrum Master* exerce o papel de facilitador e motivador da equipe, além de garantir que as regras e as ferramentas sejam utilizadas com vistas à criatividade do trabalho e ao retorno do investimento.

**013.** (FCC/TRE-CE/ANALISTA JUDICIÁRIO/ANALISTA DE SISTEMAS/2012) No SCRUM, sprint é

- a)** um representante dos stakeholders e do negócio.
- b)** uma lista de requisitos que tipicamente vêm do cliente.
- c)** uma lista de itens priorizados a serem desenvolvidos para um software.
- d)** uma iteração que segue um ciclo (PDCA) e entrega incremento de software pronto.
- e)** um conjunto de requisitos, priorizado pelo *Product Owner*.

**014.** (CESPE/ANATEL/ANALISTA ADMINISTRATIVO/ARQUITETURA DE SOLUÇÕES DE TIC/2014) Acerca dos processos de desenvolvimento de software, julgue o item seguinte: A etapa de planejamento do *Extreme Programming* (XP) inicia-se com a escrita de *UserStories* (história do usuário). Por meio dessa ferramenta, aqueles que conhecem a técnica de construção de uma solução poderão guiar quem necessita dessa solução no exercício de descrevê-la de forma simples e concisa.

**015.** (FCC/TRT 9ª REGIÃO/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2013) 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)*.

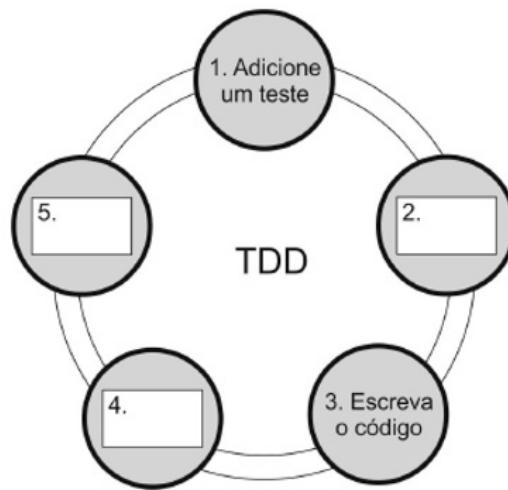
Algumas das práticas e características desses modelos de processo são descritas a seguir:

- 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:

	<b>XP</b>	<b>Scrum</b>	<b>FDD</b>
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

**016.** (FCC/ANALISTA JUDICIÁRIO (TRE-PR)/APOIO ESPECIALIZADO/ANÁLISE DE SISTEMAS/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”.

**017.** (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

**018.** (CESPE/MPOG/TECNOLOGIA DA INFORMAÇÃO/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.

**019.** (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.

**020.** (CESPE/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*).

**021.** (IESES/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, portanto só pode ser feito quando o software estiver concluído.

**022.** (FCC/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*

**023.** (CESPE/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.

**024.** (CESPE/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.

**025.** (CESPE/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.

**026.** (CESPE/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.

**027.** (CESPE/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.

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

## QUESTÕES DE CONCURSO

**029.** (CESPE/STJ/TÉCNICO JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/NÍVEL: MÉDIO/2015) No que concerne a DevOps, julgue o item que se segue. 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.

**030.** (CESPE/STJ/TÉCNICO JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2015) No que concerne a DevOps, julgue o item que se segue. DevOps é um conceito pelo qual se busca entregar sistemas melhores, com menor custo, em menor tempo e com menor risco.

**031.** (CESPE/STF/ANALISTA JUDICIÁRIO/SUPORTE EM TECNOLOGIA DA INFORMAÇÃO/2013) Acerca de DevOps e da gestão ágil de projetos com Scrum, julgue os itens subsequentes. 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.

**032.** (FGV/TJ-DFT/ANALISTA JUDICIÁRIO/ANÁLISE DE SISTEMAS/2022) Uma equipe de analista de sistemas está desenvolvendo o software ProgramaTJ aplicando a metodologia Lean. A equipe decidiu implementar apenas as funcionalidades formalmente requisitadas pelo cliente, evitando adicionar qualquer funcionalidade extra à ProgramaTJ por conta própria. Essa decisão da equipe remete, de forma direta, ao princípio da metodologia Lean para o desenvolvimento de software de:

- a) otimização do todo;
- b) adiar comprometimento;
- c) eliminação de desperdícios;
- d) respeitar as pessoas;
- e) criação de conhecimento.

**033.** (UFES/UFES/ENGENHEIRO/PRODUÇÃO/2017) A Produção Enxuta (Lean Production)

- a) é direcionada a sistemas automatizados que eliminam ou diminuem a necessidade de intensidade de mão de obra.
- b) objetiva eliminar os desperdícios.
- c) serve de referência para a produção mensal, tornando enxuto o estoque necessário.
- d) reduz o número de máquinas em uma linha de produção, recorrendo, para isso, à produção máxima de cada máquina em uso.
- e) objetiva reduzir o número de pessoas envolvidas no processo produtivo.

**034.** (FGV/ANALISTA/IBGE/ANÁLISE DE PROJETOS/2016) Uma organização adotou a filosofia Lean para a análise de seus processos e para promover a sua melhoria. Ao mapear sua cadeia de valor, ela identificou desperdícios básicos do Lean, como:

- a) produção menor que a necessária;
- b) trabalho em itens prioritários;
- c) tempo para mover coisas dentro de um processo ou entre eles;
- d) movimentação sem planejamento e em layout organizacional ruim;
- e) processamento de grande valor.

**035.** (CESPE/STJ/TÉCNICO JUDICIÁRIO/DESENVOLVIMENTO DE SISTEMAS/2018) Julgue o próximo item, relativo à governança de TI, à NBR ISO/IEC 38500:2009, ao COBIT 5 e ao DevOps. 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.

**036.** (CEBRASPE/CESPE/ANALISTA DE PREVIDÊNCIA COMPLEMENTAR/FUNPRESPE-EXE/TECNOLOGIA/2022) Acerca da engenharia de software, julgue o item que se segue. Na fase de desenvolvimento do Scrum, os requisitos são escritos no product backlog.

**037.** (UFMT/PERITO OFICIAL/POLITEC MT/CRIMINAL/CIÊNCIA DA COMPUTAÇÃO OU INFORMÁTICA/2022) No SCRUM, o papel responsável por garantir o máximo de valor agregado ao produto é:

- a) do Cliente.
- b) do Scrum Master.
- c) da Equipe.
- d) do Product Owner.
- e) do Desenvolvedor.

**038.** (FCC/ANALISTA JUDICIÁRIO/TST/APOIO ESPECIALIZADO/ANÁLISE DE SISTEMAS/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.

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

**040.** (FCC/AL-PE/ANALISTA LEGISLATIVO/SISTEMAS/ENGENHARIA DE SOFTWARE/2014)

Scrum e XP são duas metodologias ágeis que provêm práticas e regras que apresentam diferenças e também pontos em comum. Comparando-se estas metodologias, é correto afirmar:

- a)** A XP enfatiza a proximidade física do cliente com a equipe de desenvolvimento para facilitar a comunicação. No Scrum existem diversos eventos formais, tais como sprint backlog meeting e product backlog review, que incentivam a comunicação entre todos os profissionais envolvidos no projeto.
- b)** As duas metodologias utilizam iterações curtas de desenvolvimento (sprints), mas divergem no tempo de duração das mesmas. Enquanto no Scrum uma sprint dura de 15 minutos a 8 horas, na XP costuma durar de 1 a 24 horas.
- c)** Tanto o Scrum quanto a XP explicitamente não permitem que ocorram mudanças de escopo ou definição dentro de uma sprint. Por isso o cliente deve validar todos os requisitos no início do projeto, isso vai contribuir para evitar atrasos e até mesmo construções erradas.
- d)** A XP enfatiza que não se deve fazer horas extras constantemente e, se isso ocorrer, existem problemas no projeto que devem ser resolvidos não com aumento de horas, mas com melhor planejamento. O Scrum enfatiza que equipes auto-organizáveis escolhem qual a melhor forma para completarem seu trabalho.
- e)** O Scrum estabelece que os testes devem ocorrer o tempo todo durante o desenvolvimento, principalmente usando técnicas automatizadas. Na XP os testes podem ser realizados apenas na parte final de cada sprint, usando a técnica de refatoração, que busca validar todas as funcionalidades, pensando estrategicamente em como refatorar o código que está sendo implementado.

**041.** (FCC/TRT 13<sup>a</sup> REGIÃO/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2014)

No Scrum, um projeto se inicia com uma visão simples do produto que será desenvolvido. A visão pode ser vaga a princípio e ir tornando-se clara aos poucos. O ...<sup>I</sup>... então, transforma essa visão em uma lista de requisitos funcionais e não-funcionais para que, quando forem desenvolvidos, reflitam essa visão. Essa lista, chamada de ...<sup>II</sup>..., é priorizada pelo ...<sup>III</sup>... de forma que os itens que gerem maior valor ao produto tenham maior prioridade.

Completa, correta e respectivamente, as lacunas I, II e III:

- a)** Daily Scrum - Scrum Team - Sprint
- b)** Daily Scrum - Product Backlog - Sprint Planning Meeting
- c)** Product Owner - Sprint - Product Backlog
- d)** Scrum Team - Sprint Planning Meeting - Product Owner
- e)** Product Owner - Product Backlog - Product Owner

**042.** (FCC/TRF 3<sup>a</sup> REGIÃO/ANALISTA JUDICIÁRIO/INFORMÁTICA/ENGENHARIA DE SOFTWARE/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

**043.** (CESPE/MPOG/TECNOLOGIA DA INFORMAÇÃO/2013) Com relação às metodologias ágeis de desenvolvimento, julgue os itens a seguir. Do ponto de vista metodológico, o software livre é considerado uma abordagem similar aos métodos ágeis.

**044.** (CESPE/SERPRO/ANALISTA/DESENVOLVIMENTO DE SISTEMAS/ENGENHARIA DE SOFTWARE/2013) Julgue os itens a seguir, acerca de metodologias ágeis de desenvolvimento. 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.

**045.** (FCC/ASSEMBLEIA LEGISLATIVA DO ESTADO DE SÃO PAULO/AGENTE TÉCNICO LEGISLATIVO/TECNOLOGIA DA INFORMAÇÃO/2010) São consideradas metodologias ágeis de desenvolvimento de software:

- a)** XP e UP.
- b)** SCRUM e DSDM.
- c)** SCRUM e RUP.
- d)** DSDM e Cascata.
- e)** Cascata e PRINCE2.

**046.** (FCC/TRF 4<sup>a</sup> REGIÃO/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2010) Na fase de desenvolvimento do Scrum, o software é desenvolvido em processos iterativos denominados

- a)** *Building Products.*
- b)** *Product Backlog.*
- c)** *Sprint.*
- d)** *Product Owner.*
- e)** *Product Backlog Cycle.*

**047.** (CESPE/BANCO DA AMAZÔNIA/TÉCNICO CIENTÍFICO/ÁREA: TECNOLOGIA DA INFORMAÇÃO/ARQUITETURA DE TECNOLOGIA/2010) O Scrum é utilizado, como função primária, para o gerenciamento de projetos de desenvolvimento de software, mas também tem sido usado como *extreme programming* e outras metodologias de desenvolvimento. Teoricamente, o Scrum pode ser aplicado em qualquer contexto no qual um grupo de pessoas necessite trabalhar juntas para atingir um objetivo comum.

**048.** (CESGRANRIO/BNDES/DESENVOLVIMENTO/2008) Que situação favorece a escolha do uso de XP para um projeto de desenvolvimento de software, em oposição à escolha do RUP ou do modelo Cascata?

- a) Equipe do projeto localizada em diferentes cidades e com poucos recursos de colaboração.
- b) Equipe do projeto formada por pessoas com alto grau de competitividade.
- c) Cliente do projeto trabalhando em parceria com a equipe do projeto e sempre disponível para retirar dúvidas.
- d) Requisitos do software com pequena probabilidade de mudanças.
- e) Presença de um processo organizacional que exige a elaboração de vários documentos específicos para cada projeto.

**049.** (CESPE/ANAC/2009) A técnica conhecida como *refactoring* é constantemente aplicada no desenvolvimento baseado no método ágil *extreme programming*.

**050.** (CESGRANRIO/2010/IBGE/ANÁLISE DE SISTEMAS/DESENVOLVIMENTO DE APLICAÇÕES) O XP (Extreme Programming) usa uma abordagem orientada a objetos como seu paradigma de desenvolvimento predileto. Nessa perspectiva, analise as afirmativas abaixo.  
I – A atividade de Codificação começa com a criação de um conjunto de histórias que descreve as características e as funcionalidades requeridas para o software a ser construído.  
II – O XP encoraja o uso de cartões CRC (Class- Responsibility-Colaborator) como um mecanismo efetivo para raciocinar sobre o software no contexto orientado a objetos.  
III – O XP emprega a técnica de *refactoring* na codificação, mas desaconselha a utilização da programação por pares.  
IV – A criação de testes unitários antes da codificação começar é uma prática do XP.  
V – Se um difícil problema de projeto é encontrado como parte do projeto de uma história, o XP recomenda a criação imediata de um protótipo operacional daquela parte do projeto.

Estão corretas **APENAS** as afirmativas

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

**051.** (CESGRANRIO/BR DISTRIBUIDORA/INFRAESTRUTURA/ 2008) Considere a sequência de atividades e eventos apresentada a seguir.

- Projeto de concepção de um novo software de prateleira
- Projeto de desenvolvimento do novo software
- Projeto de marketing de lançamento do software
- Distribuição do software por dois anos
- Projeto de alteração das funcionalidades do software
- Projeto de marketing do software ampliado para atingir novos segmentos de mercado

Esta sequência de atividades é conhecida como

- a)** ciclo de vida do produto.
- b)** ciclo de vida de projetos.
- c)** ciclo de projetos.
- d)** projetos de ciclo de vida.
- e)** projetos de produtos.

**052.** (FUMARC/CEMIG/ANALISTA DE TI JUNIOR/2010) Sobre modelos de processo de desenvolvimento de software, assinale a alternativa INCORRETA:

- a)** O Scrum é um processo de desenvolvimento ágil de software baseado em grupos de práticas e papéis pré-definidos. Ele é um processo iterativo e incremental para gerenciamento de projetos e desenvolvimento de sistemas, onde cada *sprint* é uma iteração que segue um ciclo PDCA (*Plan, Do, Check, Act*) e entrega um incremento de software pronto.
- b)** O design centrado no usuário (UCD) é uma abordagem do processo de desenvolvimento de software baseada no entendimento explícito dos usuários, tarefas, e tem como objetivo principal o casamento entre o modelo conceitual embutido no sistema pelo projetista e o modelo mental do usuário.
- c)** Programação extrema (XP – *extreme programming*) é um processo de desenvolvimento ágil baseado em *feedback* rápido, e simplicidade; com enfoque explícito em tempo, custo e qualidade no desenvolvimento, que são alcançados através de uma definição rígida do escopo das funcionalidades da aplicação.
- d)** O modelo em espiral é um processo de desenvolvimento de software que intercala etapas de projeto e prototipação, combinando conceitos de desenvolvimento *top-down* e *bottom-up*, e permitindo, desta forma, análise de riscos e estimativas do progresso do trabalho mais realistas.

## GABARITO

- |        |       |
|--------|-------|
| 1. C   | 36. E |
| 2. C   | 37. d |
| 3. C   | 38. e |
| 4. C   | 39. C |
| 5. C   | 40. d |
| 6. E   | 41. e |
| 7. d   | 42. e |
| 8. e   | 43. C |
| 9. b   | 44. C |
| 10. b  | 45. b |
| 11. d  | 46. c |
| 12. C  | 47. C |
| 13. d  | 48. c |
| 14. C  | 49. C |
| 15. b  | 50. e |
| 16. d  | 51. a |
| 17. d  | 52. c |
| 18. E  |       |
| 19. E  |       |
| 20. a  |       |
| 21. a  |       |
| 22. c  |       |
| 23. b  |       |
| 24. C  |       |
| 25. C  |       |
| 26. C  |       |
| 27. C  |       |
| 28. C  |       |
| 29. C  |       |
| 30. C  |       |
| 31. C  |       |
| 32. c  |       |
| 33. b  |       |
| 34. d. |       |
| 35. C  |       |

## GABARITO COMENTADO

**029.** (CESPE/STJ/TÉCNICO JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/NÍVEL: MÉDIO/2015) No que concerne a DevOps, julgue o item que se segue. 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.



O termo **DevOps** (**mescla de desenvolvedor e operações**) é uma **metodologia de desenvolvimento de software baseada no alinhamento entre os times de desenvolvimento e de operações de TI em relação aos processos, ferramentas e responsabilidades**. O objetivo é aumentar a velocidade de desenvolvimento de um produto sem perder a qualidade. O alinhamento entre esses dois times consiste na automatização da maioria dos processos de desenvolvimento de software para dar agilidade, tornando possível a realização de vários **deploys** em um curto período de tempo. Para isso, o profissional especializado 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.

**Certo.**

**030.** (CESPE/STJ/TÉCNICO JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2015) No que concerne a DevOps, julgue o item que se segue. DevOps é um conceito pelo qual se busca entregar sistemas melhores, com menor custo, em menor tempo e com menor risco.



O objetivo do DevOps é aumentar a velocidade de desenvolvimento de um produto **sem perder a qualidade, entregando sistemas melhores, com menor custo, em menor tempo e com menor risco**.  
**Certo.**

**031.** (CESPE/STF/ANALISTA JUDICIÁRIO/SUPORTE EM TECNOLOGIA DA INFORMAÇÃO/2013) Acerca de DevOps e da gestão ágil de projetos com Scrum, julgue os itens subsequentes. 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.



DevOps inclui desenvolvimento colaborativo rápido sem contratemplos, **com testes, entrega e monitoração contínuo** e não funciona sem que haja uma mentalidade adequada, que tenha como objetivo permitir que todos colaborem para entregar valor mais rapidamente.

**Certo.**

**032.** (FGV/TJ-DFT/ANALISTA JUDICIÁRIO/ANÁLISE DE SISTEMAS/2022) Uma equipe de analista de sistemas está desenvolvendo o software ProgramaTJ aplicando a metodologia Lean. A equipe decidiu implementar apenas as funcionalidades formalmente requisitadas pelo cliente, evitando adicionar qualquer funcionalidade extra à ProgramaTJ por conta própria. Essa decisão da equipe remete, de forma direta, ao princípio da metodologia Lean para o desenvolvimento de software de:

- a) otimização do todo;
- b) adiar comprometimento;
- c) eliminação de desperdícios;
- d) respeitar as pessoas;
- e) criação de conhecimento.



O *Lean Manufacturing* é uma filosofia de gestão baseada na eliminação de desperdícios, eliminando do processo as atividades desnecessárias e que não agregam valor, para consequente diminuição de custos e melhoria de qualidade do produto final (WOMACK; JONES; ROOS, 1992).

**Letra c.**

**033.** (UFES/UFES/ENGENHEIRO/PRODUÇÃO/2017) A Produção Enxuta (Lean Production)

- a) é direcionada a sistemas automatizados que eliminam ou diminuem a necessidade de intensidade de mão de obra.
- b) objetiva eliminar os desperdícios.
- c) serve de referência para a produção mensal, tornando enxuto o estoque necessário.
- d) reduz o número de máquinas em uma linha de produção, recorrendo, para isso, à produção máxima de cada máquina em uso.
- e) objetiva reduzir o número de pessoas envolvidas no processo produtivo.



Figura. Lean (QUINTÃO, 2022)

**Letra b.**

**034.** (FGV/ANALISTA/IBGE/ANÁLISE DE PROJETOS/2016) Uma organização adotou a filosofia Lean para a análise de seus processos e para promover a sua melhoria. Ao mapear sua cadeia de valor, ela identificou desperdícios básicos do Lean, como:

- a) produção menor que a necessária;
- b) trabalho em itens prioritários;
- c) tempo para mover coisas dentro de um processo ou entre eles;
- d) movimentação sem planejamento e em layout organizacional ruim;
- e) processamento de grande valor.



A filosofia “Lean” é uma abordagem com foco em processos e que tem, como essência, a busca pela **redução de desperdícios, eliminando do processo as atividades desnecessárias e que não agregam valor!**

Ao mapear sua cadeia de valor, a organização identificou que a movimentação **sem** planejamento e em *layout* organizacional **ruim** pode gerar processos desnecessários e, consequentemente, desperdícios.

**Letra d.**

**035.** (CESPE/STJ/TÉCNICO JUDICIÁRIO/DESENVOLVIMENTO DE SISTEMAS/2018) Julgue o próximo item, relativo à governança de TI, à NBR ISO/IEC 38500:2009, ao COBIT 5 e ao DevOps. 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.



**Obs.:** O **Scrum** é um **framework para gerenciamento de projetos** bastante popular atualmente. A ideia básica do **Scrum** é um ciclo de desenvolvimento curto (*Sprints*), de poucas semanas, que ao final realizará a entrega de algo que tenha valor para o usuário. Em um processo típico do Scrum, temos a figura do **Product Owner**, que é um especialista da área de negócio, do **Scrum Master**, que é um facilitador, e do **time de desenvolvimento**, que são os técnicos que irão efetivamente produzir a solução. Tais figuras estão sempre em contato, seja definindo quais serão as próximas entregas, esclarecendo os requisitos de negócio ou debatendo os resultados obtidos até então (ENAP, 2020).

**Ao se adotar o Scrum conjuntamente com o DevOps**, a equipe de infraestrutura irá compor o time de desenvolvimento. Com isso, **problemas** de ambiente são **minimizados** e, consequentemente, a chance de sucesso da Sprint ampliada.

**Certo.**

**036.** (CEBRASPE/CESPE/ANALISTA DE PREVIDÊNCIA COMPLEMENTAR/FUNPRESPE-EXE/TECNOLOGIA/2022) Acerca da engenharia de software, julgue o item que se segue.  
 Na fase de desenvolvimento do Scrum, os requisitos são escritos no product backlog.



Artefatos Oficiais	Descrição
<i>Product Backlog</i>	Uma lista ordenada (por valor, risco, prioridade, etc.) de requisitos ou funcionalidades que o produto deve conter criada pela Equipe Scrum.

Conforme visto, **os requisitos são escritos no Product Backlog**. O erro da questão está evidente ao listar que os requisitos são escritos na fase de desenvolvimento, o que não é verdade. Na fase de desenvolvimento da Scrum já será utilizado o levantamento de requisitos!

**Errado.**

**037.** (UFMT/PERITO OFICIAL/POLITEC MT/CRIMINAL/CIÊNCIA DA COMPUTAÇÃO OU INFORMÁTICA/2022) No SCRUM, o papel responsável por garantir o máximo de valor agregado ao produto é:

- a) do Cliente.
- b) do Scrum Master.
- c) da Equipe.
- d) do Product Owner.
- e) do Desenvolvedor.



Os **principais papéis do Scrum** são: **Product Owner (PO)**, **Scrum Master (SM)** e **Development Team (DT)**.

Papéis do Scrum	Descrição
<i>Product Owner (PO)</i>	<b>Dono do Produto.</b> É uma pessoa e <b>não</b> um comitê. Ele pode representar o desejo de um comitê no <i>Product Backlog</i> , mas aqueles que quiserem uma alteração nas prioridades dos itens de <i>backlog</i> devem convencer o <i>Product Owner</i> . <b>É o responsável por maximizar o valor do produto</b> e do trabalho do Time de Desenvolvimento. Como isso é feito pode variar amplamente de acordo com as organizações, Times Scrum e indivíduos.

**Letra d.**

**038.** (FCC/ANALISTA JUDICIÁRIO/TST/APOIO ESPECIALIZADO/ANÁLISE DE SISTEMAS/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.



Atividade	Modelos
I – Dividir o cronograma em iterações time-box ou ciclos (sprints).	No <u>Scrum</u> , os projetos progridem por intermédio de uma série sequencial de sprints. Cada sprint possui um período fixo que normalmente varia de duas a quatro semanas.
II – Orientar o trabalho a eventos ao invés de limite de tempo.	Diferentemente do Scrum que possui <i>sprints</i> regulares de tamanho fixo, o <u>Kanban</u> possui fluxo contínuo, entrega contínua e mudanças podem ocorrer a qualquer momento.
III – Aplicar a programação em pares, integração contínua, orientação a testes (TDD.), revisão de código e todas as demais prescrições antes da implantação.	Esse item destaca práticas da XP. Na programação em pares duas pessoas trabalham juntas em uma mesma estação de trabalho para codificar. Conforme o trabalho é completado, o código é integrado ao trabalho dos outros (eis a integração contínua!). Ainda, cabe destacar que os programadores só partem para codificação depois que testes de unidade foram criados (TDD).

**Letra e.**

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



O Kanban pode ser considerado 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 acontece, por exemplo, no 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.

**Obs.:** 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.

**Certo.**

**040. (FCC/AL-PE/ANALISTA LEGISLATIVO/SISTEMAS/ENGENHARIA DE SOFTWARE/2014)**

Scrum e XP são duas metodologias ágeis que provêm práticas e regras que apresentam diferenças e também pontos em comum. Comparando-se estas metodologias, é correto afirmar:

- a)** A XP enfatiza a proximidade física do cliente com a equipe de desenvolvimento para facilitar a comunicação. No Scrum existem diversos eventos formais, tais como sprint backlog meeting e product backlog review, que incentivam a comunicação entre todos os profissionais envolvidos no projeto.
- b)** As duas metodologias utilizam iterações curtas de desenvolvimento (sprints), mas divergem no tempo de duração das mesmas. Enquanto no Scrum uma sprint dura de 15 minutos a 8 horas, na XP costuma durar de 1 a 24 horas.
- c)** Tanto o Scrum quanto a XP explicitamente não permitem que ocorram mudanças de escopo ou definição dentro de uma sprint. Por isso o cliente deve validar todos os requisitos no início do projeto, isso vai contribuir para evitar atrasos e até mesmo construções erradas.
- d)** A XP enfatiza que não se deve fazer horas extras constantemente e, se isso ocorrer, existem problemas no projeto que devem ser resolvidos não com aumento de horas, mas com melhor planejamento. O Scrum enfatiza que equipes auto-organizáveis escolhem qual a melhor forma para completarem seu trabalho.
- e)** O Scrum estabelece que os testes devem ocorrer o tempo todo durante o desenvolvimento, principalmente usando técnicas automatizadas. Na XP os testes podem ser realizados apenas na parte final de cada sprint, usando a técnica de refatoração, que busca validar todas as funcionalidades, pensando estrategicamente em como refatorar o código que está sendo implementado.



A XP é baseada numa série de práticas, a saber:

- Cliente Presente (*On-site customer*)
- Jogo do Planejamento (*The Planning Game*)
- Pequenos Lançamentos (*Small Releases*)
- Desenvolvimento Guiado pelos testes (*Test First Design*)
- Integração Continua (*Continuous Integration*)
- Projeto Simples (*Simple Design*)
- Refatoração (*Refactoring*)
- Programação em pares (*Pair Programming*)
- Propriedade Coletiva (*Collective Ownership*)
- Padrões de Codificação (*Coding Standards*)
- Ritmo Sustentável (*40 Hour Week*)
- Metáfora (*Metaphor*)

De acordo com a prática de **Ritmo Sustentável**, a equipe não deve trabalhar mais do que 40 horas por semana, evitando-se horas extras.

Já no Scrum, a equipe deve determinar a forma de organizar seu horário de maneira autônoma.

**Letra d.**

---

**041.** (FCC/TRT 13<sup>a</sup> REGIÃO/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2014)

No Scrum, um projeto se inicia com uma visão simples do produto que será desenvolvido. A visão pode ser vaga a princípio e ir tornando-se clara aos poucos. O <sup>I</sup>... então, transforma essa visão em uma lista de requisitos funcionais e não-funcionais para que, quando <sup>forem</sup> desenvolvidos, reflitam essa visão. Essa lista, chamada de <sup>II</sup>..., é priorizada pelo <sup>III</sup>... de forma que os itens que gerem maior valor ao produto tenham maior prioridade.

Completa, correta e respectivamente, as lacunas I, II e III:

- a) Daily Scrum - Scrum Team - Sprint
- b) Daily Scrum - Product Backlog - Sprint Planning Meeting
- c) Product Owner - Sprint - Product Backlog
- d) Scrum Team - Sprint Planning Meeting - Product Owner
- e) Product Owner - Product Backlog - Product Owner



O Product Owner é responsável pela identificação dos requisitos (funcionais e não-funcionais) que serão desenvolvidos. Estes requisitos compõem o Product Backlog, e serão desenvolvidos de acordo com a prioridade definida pelo Product Owner nas Sprint Planning Meetings.

**Letra e.**

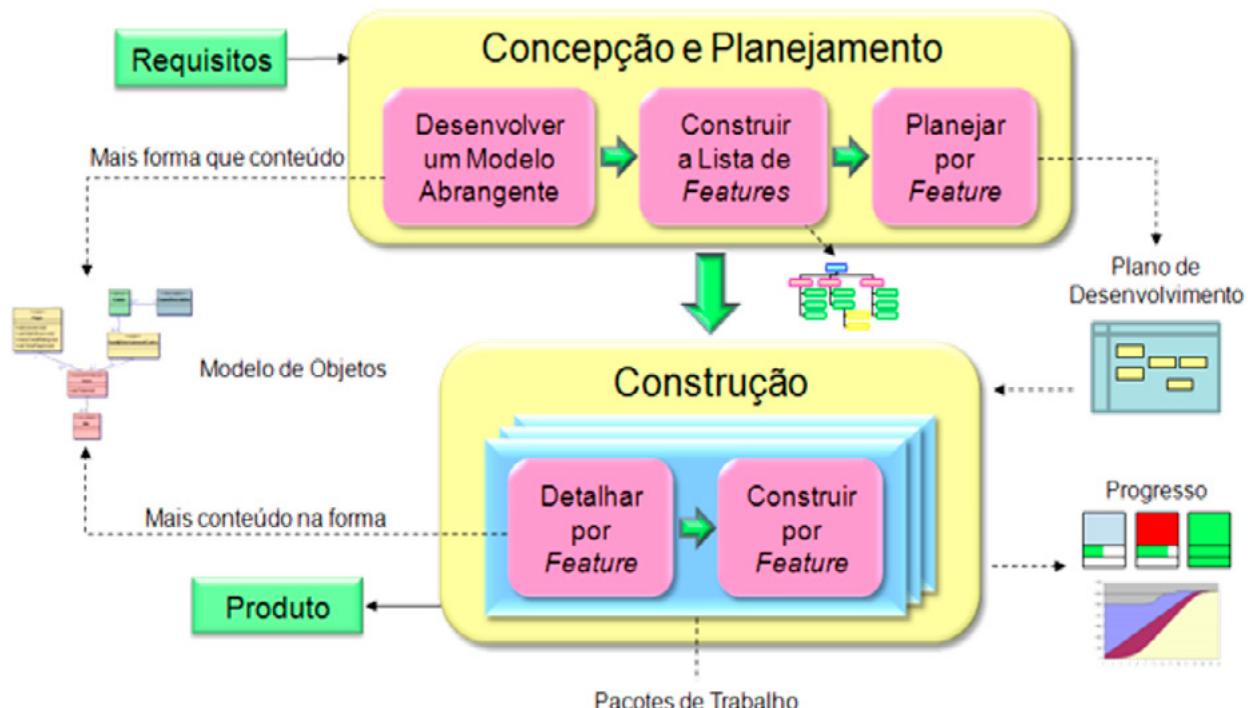
**042. (FCC/TRF 3<sup>a</sup> REGIÃO/ANALISTA JUDICIÁRIO/INFORMÁTICA/ENGENHARIA DE SOFTWARE/2014)** Os modelos ágeis de desenvolvimento de software têm menos ênfase nas definições de atividades e mais ênfase na pragmática e nos fatores humanos do desenvolvimento. Um destes modelos enfatiza o uso de orientação 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**



No enunciado da questão é apresentada justamente a definição de **FDD (Feature Driven Development)**, em português **Desenvolvimento Guiado por Funcionalidades**) que é uma das seis metodologias ágeis originais do desenvolvimento de software.

A estrutura básica do FDD é apresentada na figura abaixo, obtida em [http://pt.wikipedia.org/wiki/Feature\\_Driven\\_Development](http://pt.wikipedia.org/wiki/Feature_Driven_Development)



Conforme a figura acima, a fase de Concepção e Planejamento divide-se em 3 processos:

- **Desenvolver um Modelo Abrangente:** visa o entendimento das regras de negócio do projeto para gerar um modelo de objetos;
- **Construir uma Lista de Funcionalidades (Features):** divisão do modelo desenvolvido em três camadas: áreas de negócio, atividades de negócio e passos automatizados da atividade (funcionalidades). O resultado é uma hierarquia de funcionalidades do produto (*product backlog*); e
- **Planejar por Funcionalidade:** a estimativa da complexidade e dependência das funcionalidades, gerando um plano de desenvolvimento.

Já a Construção, divide-se em duas disciplinas, a saber: Detalhar por Funcionalidade: detalhar os requisitos e outros artefatos para a codificação; e Construir por Funcionalidade: o código e desenvolvido, incrementando-se o projeto.

**Letra e.**

---

**043.** (CESPE/MPOG/TECNOLOGIA DA INFORMAÇÃO/2013) Com relação às metodologias ágeis de desenvolvimento, julgue os itens a seguir. Do ponto de vista metodológico, o software livre é considerado uma abordagem similar aos métodos ágeis.



A banca considerou a assertiva como verdadeira no gabarito definitivo, provavelmente por considerar a “propriedade coletiva do código” em ambos os casos.

**Certo.**

---

**044.** (CESPE/SERPRO/ANALISTA/DESENVOLVIMENTO DE SISTEMAS/ENGENHARIA DE SOFTWARE/2013) Julgue os itens a seguir, acerca de metodologias ágeis de desenvolvimento. 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.



Kanban é um **método de desenvolvimento de software** com fortes bases em práticas enxutas, e tem como objetivo otimizar o processo de desenvolvimento de software pré-existente.

Henrik destaca as seguintes práticas do Kanban:

- 1 - **Visualizar o fluxo de trabalho (workflow);**
- 2 - **Limitar o trabalho em progresso;**
- 3 - **Gerenciar e medir o fluxo.**

Segundo o autor, embora poucas, quando bem utilizadas essas práticas podem levar um time de desenvolvimento de software a ótimos resultados. David Anderson recomenda que se acrescente mais duas condições a essa lista de Kniberg, para se ter um processo ainda mais otimizado:

- 1 - Tornar políticas do processo explícitas;
- 2 - Usar modelos para avaliar Oportunidades de Melhoria.

Entende-se por **fluxo** a progressão visual dos itens de trabalho por meio do sistema Kanban.

Assim, uma de suas práticas é o **Gerenciamento do Fluxo**. Nesse contexto:

- visualize o fluxo de trabalho: divida o trabalho em partes, e use colunas nomeadas para ilustrar onde cada item se encontra no fluxo de trabalho;

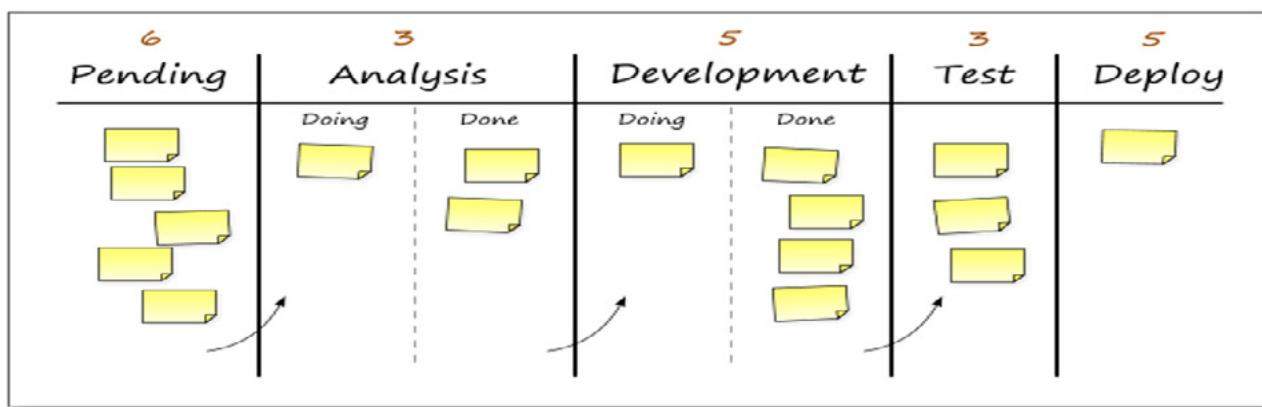


Figura. Quadro Kanban (Fonte: <http://www.kanbanblog.com/explained/>)

- limite o trabalho em progresso (*work in progress*), associando limites explícitos para quantos itens podem estar em progresso em cada estado do fluxo de trabalho;
- acompanhe o tempo de execução da tarefa e otimize o processo para tornar o tempo de execução o menor e mais previsível possível.

Segundo destaca Gomes (2013)

sempre que um item de trabalho parecer estar parado por muito tempo, sem fluir no processo, a equipe deve conversar a respeito para identificar a causa do problema, e então pensar em como melhorar a fluidez do processo.

Entre as causas possíveis para problemas que podem levar um item de trabalho a não fluir no Kanban podemos citar: dificuldades técnicas, cliente ou analistas de negócio indisponíveis para esclarecer dúvidas de negócio da equipe, requisitos pouco claros, ambiente de homologação indisponível, longas esperas por informações, etc.<sup>3</sup>

## Certo.

<sup>3</sup> Fonte:

Gomes, André Faria. Desenvolvimento Ágil com kanban, disponível em: [http://www.devmedia.com.br/websys.5/webreader.asp?cat=6&artigo=2955&revista=javamagazine\\_84#a-2955](http://www.devmedia.com.br/websys.5/webreader.asp?cat=6&artigo=2955&revista=javamagazine_84#a-2955).

Fonte: What is kanban? Em: <http://www.kanbanblog.com/explained>

**045.** (FCC/ASSEMBLEIA LEGISLATIVA DO ESTADO DE SÃO PAULO/AGENTE TÉCNICO LEGISLATIVO/TECNOLOGIA DA INFORMAÇÃO/2010) São consideradas metodologias ágeis de desenvolvimento de software:

- a) XP e UP.
- b) SCRUM e DSDM.
- c) SCRUM e RUP.
- d) DSDM e Cascata.
- e) Cascata e PRINCE2.



O jeito aqui é eliminarmos o que temos certeza de que não é metodologia ágil, então vamos lá! O modelo em Cascata (*Waterfall*), também chamado de Clássico, é o mais tradicional processo de desenvolvimento de software, e então podemos descartar as letras d) e e).

O **Rational Unified Process (RUP)** é um exemplo de modelo de processo de desenvolvimento baseado no *Unified Process* (Processo Unificado) desenvolvido pela Rational. Então já eliminados RUP e UP (Processo Unificado) que não tem relação com a metodologia ágil, o que excluirá as assertivas a)e d).

Por eliminação, chegamos à letra D.

O Scrum é um processo de desenvolvimento ágil de software baseado em grupos de práticas e papéis pré-definidos. Ele é um processo **iterativo e incremental para gerenciamento de projetos e desenvolvimento de sistemas**, em que cada *sprint* é uma iteração que segue um ciclo PDCA (*Plan, Do, Check, Act*) e entrega um incremento de software pronto.

Por fim, o **DSDM (Dynamic Systems Development Method - Método de Desenvolvimento de Sistemas Dinâmicos)**, segundo Pressman (2011, p. 96) é uma abordagem de desenvolvimento de software ágil que “oferece uma metodologia para construir e manter sistemas que atendem restrições de prazo apertado por meio do uso da prototipagem incremental em um ambiente de projeto controlado.

**Letra b.**

**046.** (FCC/TRF 4ª REGIÃO/ANALISTA JUDICIÁRIO/TECNOLOGIA DA INFORMAÇÃO/2010)

Na fase de desenvolvimento do Scrum, o software é desenvolvido em processos iterativos denominados

- a) Building Products.
- b) Product Backlog.
- c) Sprint.
- d) Product Owner.
- e) Product Backlog Cycle.



O **Scrum** é um processo de desenvolvimento ágil de software baseado em grupos de práticas e papéis predefinidos.

Ele é um processo **iterativo** e **incremental** para gerenciamento de projetos e desenvolvimento de sistemas, em que cada **sprint** é uma iteração que segue um ciclo PDCA (*Plan, Do, Check, Act*) e entrega um incremento de software pronto.

**Letra c.**

---

**047.** (CESPE/BANCO DA AMAZÔNIA/TÉCNICO CIENTÍFICO/ÁREA: TECNOLOGIA DA INFORMAÇÃO/ARQUITETURA DE TECNOLOGIA/2010) O Scrum é utilizado, como função primária, para o gerenciamento de projetos de desenvolvimento de software, mas também tem sido usado como *extreme programming* e outras metodologias de desenvolvimento. Teoricamente, o Scrum pode ser aplicado em qualquer contexto no qual um grupo de pessoas necessite trabalhar juntas para atingir um objetivo comum.



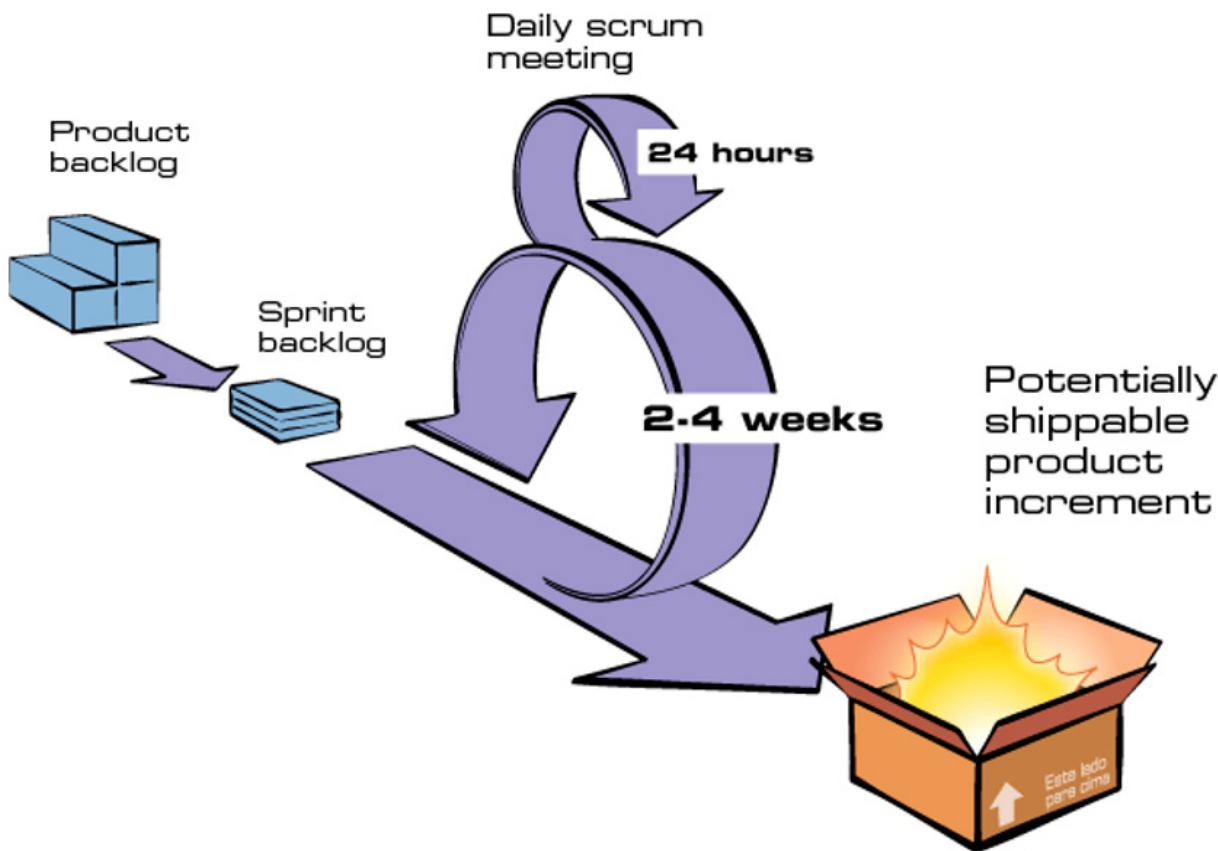
Define-se **Scrum** como uma **metodologia para desenvolvimento ágil e Gerenciamento de Projetos**. Esta metodologia foi concebida como uma forma de gerência de projetos em empresas automobilísticas e de produtos de consumo, a partir da observação de que projetos usando equipes pequenas e multidisciplinares produziram os melhores resultados, e associaram estas equipes altamente eficazes à formação Scrum do Rugby.

De acordo com o artigo apresentado no site <http://improveit.com.br/scrum>, na metodologia Scrum, os projetos são divididos em ciclos (tipicamente mensais) chamados de Sprints. O Sprint representa um intervalo de tempo no qual uma série de atividades deve ser executada. Como toda metodologia ágil de desenvolvimento de software, ele é iterativo, e cada iteração, é chamada de Sprint.

A utilização do Scrum ocorre da seguinte forma:

- As funcionalidades de um projeto são mantidas em uma lista chamada *Product Backlog*;
- Ao iniciar-se um Sprint, faz-se uma reunião de planejamento (*Sprint Planning Meeting*) na qual o *Product Owner* prioriza as funcionalidades do *Product Backlog* e a equipe seleciona as atividades que ela será capaz de implementar;
- As tarefas do respectivo Sprint são transferidas do *Product Backlog* para o *Sprint Backlog*.
- Diariamente, a equipe faz uma breve reunião (*Daily Scrum*), em geral pela manhã, visando disseminar conhecimento sobre o trabalho realizado no dia anterior, identificar impedimentos e priorizar o trabalho do dia; e

- No término de um *Sprint*, a equipe apresenta as funcionalidades implementadas em uma Sprint Review Meeting. Após isso, faz-se uma *Sprint Retrospective* e a equipe parte para o planejamento do próximo Sprint.
- Reinicia-se assim o ciclo.



**Certo.**

**048.** (CESGRANRIO/BNDES/DESENVOLVIMENTO/2008) Que situação favorece a escolha do uso de XP para um projeto de desenvolvimento de software, em oposição à escolha do RUP ou do modelo Cascata?

- Equipe do projeto localizada em diferentes cidades e com poucos recursos de colaboração.
- Equipe do projeto formada por pessoas com alto grau de competitividade.
- Cliente do projeto trabalhando em parceria com a equipe do projeto e sempre disponível para retirar dúvidas.
- Requisitos do software com pequena probabilidade de mudanças.
- Presença de um processo organizacional que exige a elaboração de vários documentos específicos para cada projeto.



Na XP todas as decisões sobre o rumo do projeto devem ser tomadas pelo cliente. Com o cliente sempre presente reforçamos a necessidade de comunicação entre as partes.

**Letra c.**

**049.** (CESPE/ANAC/2009) A técnica conhecida como *refactoring* é constantemente aplicada no desenvolvimento baseado no método ágil *extreme programming*.



**Refatoração** é uma prática do XP (*Extreme Programming*) constantemente aplicada! Ela destaca que o código deve ser constantemente melhorado, tornando-o mais simples e mais genérico, removendo redundâncias e duplicidades.

**Certo.**

**050.** (CESGRANRIO/2010/IBGE/ANÁLISE DE SISTEMAS/DESENVOLVIMENTO DE APLICAÇÕES) O XP (*Extreme Programming*) usa uma abordagem orientada a objetos como seu paradigma de desenvolvimento predileto. Nessa perspectiva, analise as afirmativas abaixo.

- I – A atividade de Codificação começa com a criação de um conjunto de histórias que descreve as características e as funcionalidades requeridas para o software a ser construído.
- II – O XP encoraja o uso de cartões CRC (*Class- Responsibility-Colaborator*) como um mecanismo efetivo para raciocinar sobre o software no contexto orientado a objetos.
- III – O XP emprega a técnica de *refactoring* na codificação, mas desaconselha a utilização da programação por pares.
- IV – A criação de testes unitários antes da codificação começar é uma prática do XP.
- V – Se um difícil problema de projeto é encontrado como parte do projeto de uma história, o XP recomenda a criação imediata de um protótipo operacional daquela parte do projeto.

Estão corretas **APENAS** as afirmativas

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



I – Errado. Segundo Beck e Fowler (2001), o XP procura assegurar que o cliente tome todas as decisões de negócio, enquanto a equipe de desenvolvimento cuida das questões técnicas. Teles (2004) destaca que todas as funcionalidades do sistema são levantadas por meio de estórias, que são registradas em pequenos cartões. Essas estórias devem ser escritas sempre pelo próprio cliente, com suas próprias palavras. A equipe de desenvolvimento utiliza os cartões para saber quais são as funcionalidades desejadas pelo cliente, e, então, os desenvolvedores escolhem as estórias que irão implementar em cada dia de trabalho.

II – Certo. A **Extreme Programming** é orientada a objetos, sendo muito usada com sucesso por grandes empresas do mercado de software. Para implantar a XP é necessário que se norteie o trabalho baseado em valores, podendo ser destacado o valor Coragem, em que é necessário mudanças na maneira pela qual se desenvolvem sistemas. Colocar um sistema em produção assim que ele tenha valor para o cliente, fazer apenas o que se precisa para o momento e calcar o processo de análise principalmente na comunicação não é fácil, e precisa que a equipe esteja realmente decidida a mudar o seu processo de desenvolvimento.

Na XP todas as decisões sobre o rumo do projeto devem ser tomadas pelo cliente. Ele deve priorizar as tarefas, ser responsável pelos testes de aceitação, e, acima de tudo, orientar e tirar dúvidas dos desenvolvedores durante o processo de programação. É por isto que as estórias do usuário, os cartões nos quais os requisitos do sistema são passados pelo cliente, têm apenas tópicos sobre o que deve ser feito, e não uma descrição detalhada das tarefas, algo que será feito pelo cliente durante o andamento do projeto (SOMMERVILLE, 2007).

**Cartões Classe-Responsabilidade-Colaboração (Class-Responsibility-Collaboration - CRC)** é uma técnica de modelagem popular orientada a texto, criada por Kent Beck e Ward Cunningham, inicialmente, para apoiar o ensino da modelagem orientada a objetos. Entretanto se mostrou uma técnica bastante útil para, de uma forma dinâmica e interativa, identificar as abstrações chave do sistema, realizar a modelagem OO conceitual e mesmo a modelagem de design OO. Podendo esta técnica ser empregada por qualquer processo de desenvolvimento OO, a mesma vem sendo adotada principalmente pelos processos ágeis, pela grande convergência existente entre suas características (p. ex., interatividade).

Larman (2008) destaca que os Cartões CRC são cartões de indexação em papel sobre os quais se escrevem as responsabilidades e os colaboradores de classes. Cada cartão representa uma classe. Uma sessão de modelagem CRC envolve um grupo sentado em volta de uma mesa, discutindo e escrevendo nos cartões enquanto montam cenários (“o que acontece se”) com os objetos, considerando o que eles precisam fazer e com que outros objetos eles precisam colaborar.

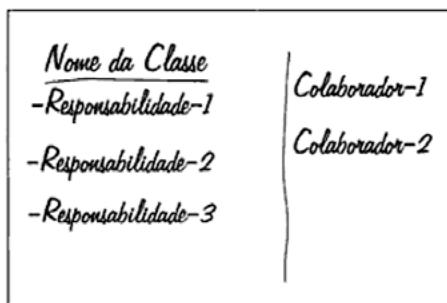


Figura. Gabarito de um cartão CRC, fonte: Larman (2008)

<u>Figura de Grupo</u> Contém mais figuras (não desenhadas) Encaminha transformações Imagem vazia em Cache ou atualização de membro	<u>Figuras</u> Figuras	<u>Desenhos</u> Contém figuras Acumula atualizações restaura a pedido	<u>Figura</u> <u>Visão de desenho</u> <u>Controlador de desenho</u>
<u>Ferramenta de Seleção</u> Seleciona figura (adicionar tratador para visão de desenho) Invoca tratadores	<u>Cache de desenho</u> <u>Visão de desenho</u> <u>Figuras</u> <u>Tratadores</u>	<u>Ferramenta de Rolagem</u> Ajusta as janelas de visão	<u>Visão de desenho</u>

Figura. Ilustra quatro exemplos de cartões CRC, Fonte: Larman (2008)

III – Errado. O XP emprega a técnica de *refactoring* na codificação. **Refatoração** é o processo de mudar um sistema de software de tal forma que não se altera o comportamento externo do código, mas melhora sua estrutura interna. É um meio disciplinado de limpar o código e que reduz as possibilidades de introduzir erros. Em essência quando você refina o código, você melhora o projeto depois que este foi escrito. Todo o código que vai para a produção é escrito por um par de programadores que utilizam uma mesma estação de trabalho ao mesmo tempo, ou seja, um computador, um teclado e dois desenvolvedores (SOMMERVILLE, 2007).

IV – Certo. Segundo Teles (2004), teste é a parte do desenvolvimento de software que todo mundo sabe que precisa ser feita, mas ninguém quer fazer!! Para o XP, adotar o desenvolvimento guiado por testes é seguir o caminho da PREVENÇÃO. Você incorpora alguns hábitos que irão assegurar que o seu software tenha uma probabilidade menor de contrair uma doença. A ideia da prevenção é diminuir as chances de que um problema ocorra.

O XP trabalha com dois tipos de testes: testes de unidade e testes de aceitação.

- Teste de unidade: realizado sobre cada classe do sistema para verificar se os resultados gerados são corretos.
- Teste de aceitação: efetuados sobre cada funcionalidade, ou estória do sistema. Verificam não apenas uma classe em particular, mas a interação entre um conjunto de classes que implementam uma dada funcionalidade.

Em ambos os casos, os testes devem ser gerados em primeiro lugar, isto é, no caso dos testes de unidade, eles devem ser escritos ANTES das classes do sistema; os testes de aceitação, por sua vez, devem ser criados antes das estórias serem implementadas. Além disso, ambos os testes devem ser automatizados, de modo que possam ser executados uma infinidade de vezes ao longo do desenvolvimento.

Complementando, para um melhor entendimento da questão...

Para testar uma classe, é necessária a compreensão de quais são suas características (conhecer os parâmetros que seus métodos podem receber e os que não podem; conhecer as respostas corretas, as incorretas e as que não poderiam ocorrer em nenhuma circunstância). É preciso, portanto, conhecer bem o problema e que tipo de solução é adequada para ele.

Quando o desenvolvedor pensa no teste ANTES DE PENSAR NA IMPLEMENTAÇÃO, ele é forçado a compreender melhor o problema!!

V – Certo. Um protótipo é uma versão inicial de um sistema de software, utilizado para demonstrar conceitos, experimentar opções de projeto e, geralmente, conhecer mais sobre o problema e suas possíveis soluções. Desenvolvimento rápido e iterativo do protótipo é essencial, de modo que os custos são controlados e os stakeholders do sistema podem experimentar o protótipo mais cedo no processo de software (SOMMERVILLE, 2007).

**Letra e.**

---

**051.** (CESGRANRIO/BR DISTRIBUIDORA/INFRAESTRUTURA/ 2008) Considere a sequência de atividades e eventos apresentada a seguir.

- Projeto de concepção de um novo software de prateleira
- Projeto de desenvolvimento do novo software
- Projeto de marketing de lançamento do software
- Distribuição do software por dois anos
- Projeto de alteração das funcionalidades do software
- Projeto de marketing do software ampliado para atingir novos segmentos de mercado

Esta sequência de atividades é conhecida como

- a) ciclo de vida do produto.
- b) ciclo de vida de projetos.
- c) ciclo de projetos.
- d) projetos de ciclo de vida.
- e) projetos de produtos.



A sequência de atividades e eventos apresentada está relacionada ao ciclo de vida do PRODUTO. No ciclo de vida do projeto seriam mencionadas atividades do PMBOK, a abordagem XP utiliza Scrum para gerenciamento, etc.

**Letra a.**

**052.** (FUMARC/CEMIG/ANALISTA DE TI JUNIOR/2010) Sobre modelos de processo de desenvolvimento de software, assinale a alternativa INCORRETA:

- a)** O Scrum é um processo de desenvolvimento ágil de software baseado em grupos de práticas e papéis pré-definidos. Ele é um processo iterativo e incremental para gerenciamento de projetos e desenvolvimento de sistemas, onde cada *sprint* é uma iteração que segue um ciclo PDCA (*Plan, Do, Check, Act*) e entrega um incremento de software pronto.
- b)** O design centrado no usuário (UCD) é uma abordagem do processo de desenvolvimento de software baseada no entendimento explícito dos usuários, tarefas, e tem como objetivo principal o casamento entre o modelo conceitual embutido no sistema pelo projetista e o modelo mental do usuário.
- c)** Programação extrema (XP – *extreme programming*) é um processo de desenvolvimento ágil baseado em *feedback* rápido, e simplicidade; com enfoque explícito em tempo, custo e qualidade no desenvolvimento, que são alcançados através de uma definição rígida do escopo das funcionalidades da aplicação.
- d)** O modelo em espiral é um processo de desenvolvimento de software que intercala etapas de projeto e prototipação, combinando conceitos de desenvolvimento *top-down* e *bottom-up*, e permitindo, desta forma, análise de riscos e estimativas do progresso do trabalho mais realistas.



a) Certa. O **Scrum** é um *framework* de processo ágil utilizado para gerenciar e controlar o desenvolvimento de um produto de software por meio de práticas iterativas e incrementais. É composto por um conjunto de boas práticas de gestão que admite ajustes rápidos, acompanhamento e visibilidade constantes e planos realísticos.

**Scrum é um processo bastante leve para gerenciar e controlar projetos de desenvolvimento de software e para a criação de produtos.**

b) Certa. O **design centrado no usuário (UCD)** é uma forma de se trabalhar o desenvolvimento de qualquer produto a partir das necessidades dos usuários. Ele (UCD) envolve a aplicação de metodologias de usabilidade essenciais ao desenvolvimento de sistemas interativos e produtos. Alguns conceitos:

Donald A. Norman → The Design of Everyday Things	"Princípios de design que – quando seguidos – dão respostas aos usuários tornando o uso dos dispositivos mais fácil".
Albert N. Badre → Shaping Web Usability	"Facilidade de uso, e facilidade de aprendizado".
Jeffrey Rubin → Handbook of Usability Testing	"Um conjunto de quatro fatores reunidos em um dispositivo: 1) capacidade de ser usado com sucesso; 2) facilidade de ser usado; 3) capacidade do usuário aprender a usar o dispositivo de forma simples e rápida; 4) provocar satisfação visual ao usuário".
Brian Shackel → Usability—context, framework, definition, design and evaluation	"Capacidade, em termos funcionais humanos, de um sistema ser usado com facilidade e de forma eficiente".
Jesse James Garrett → The Elements of User Experience	"Pensar em usabilidade é pensar em produtos fáceis de usar".
Kreta Chandler e Karen Hyatt → Customer-centered Design	"Efetividade, eficiência e satisfação com a qual usuários conseguem atingir seus objetivos ao utilizar um dispositivo".
Mark Pearson → Web Site Usability Handbook	"A ciência de aplicação de metodologias ao design para a criação de dispositivos fáceis de usar, de fácil aprendizado e que sejam úteis com o menor índice de desconforto possível".
Jakob Nielsen → Usability Engineering	"Um conjunto de propriedades de uma interface que reúne os seguintes componentes: 1) Fácil aprendizado; 2) Eficiência; 3) Capacidade de memorização; 4) Baixo índice de erros; 5) Satisfação e prazer ao uso".

- c) Errada. A **Programação extrema (XP – extreme programming)** não prevê a definição rígida do escopo das funcionalidades da aplicação, uma vez que é uma metodologia voltada para projetos cujos requisitos são vagos e mudam com frequência.
- d) Certa. A principal inovação do **Modelo Espiral** é guiar o processo de desenvolvimento gerado a partir de um *metamodelo* com base em análise de riscos e planejamento que é realizado durante toda a evolução do desenvolvimento.

**Letra c.**

## REFERÊNCIAS

ATLASSIAN. **O que é Kanban.** Disponível em: <[www.atlassian.com/agile/kanban](http://www.atlassian.com/agile/kanban)>. Acesso em: jun. 2021.

BECK. K. **Extreme Programming Explained: Embrace Change**, 2ed., Addison-Wesley, 2004.

BORGES, E. N. **Conceitos e Benefícios do Test Driven Development.** Universidade Federal do Rio Grande do Sul. Disponível em: <<http://www.inf.ufrgs.br/~cesantin/TDD-Eduardo.pdf>>. Acesso em: jun. 2021.

CAELUM. **Test-Driven Development.** Disponível em: <<http://tdd.caelum.com.br/>>. Acesso em: 10/11/2020.

CUKIER, D. **DDD – Introdução a Domain Driven Design.** 2010. Disponível em: <http://www.agileandart.com/2010/07/16/ddd-introducao-a-domain-driven-design/>. Acesso em: jun. 2021.

DEVMEDIA. **Test Driven Development: TDD Simples e Prático.** Disponível em: <<https://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>>. 2019. Acesso em: jun. 2021.

\_\_\_\_\_. **TDD: fundamentos do desenvolvimento orientado a testes.** Disponível em: <<http://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151>>. Acesso em: 10/11/2020.

ELEMARJR. **BDD na prática – parte 1 – Conceitos básicos e algum código.** 2012. Disponível em: <<https://elemarjr.wordpress.com/2012/04/11/bdd-na-prtica-parte-1-conceitos-bsicos-e-algun-cdigo/>>. Acesso em: jun. 2021.

ENAP. **Curso: O papel do DevOps na Transformação Digital dos Serviços Públicos.** 2020.

FREEMAN, Steve. Pryce, Nat. **Desenvolvimento de Software Orientado a objetos, Guiado por Testes.** Rio de Janeiro: Alta Books, 2012.

GASpareto, O. **Test Driven Development.** Universidade Federal do Rio Grande do Sul. Disponível em: <<http://www.inf.ufrgs.br/~cesantin/TDD-Otavio.pdf>>. Acesso em: 10/11/2020.

GOMES, A. F. **Desenvolvimento Ágil com kanban.** Disponível em: <[http://www.devmedia.com.br/websys.5/webreader.asp?cat=6&artigo=2955&revista=javamagazine\\_84#a-2955](http://www.devmedia.com.br/websys.5/webreader.asp?cat=6&artigo=2955&revista=javamagazine_84#a-2955)>. Acesso em: jun. 2021.

KRUCHTEN, P. **Introdução ao RUP Rational Unified Process**, 2. ed., Rio de Janeiro: Ciência Moderna, 2003.

LEAN INSTITUTE BRASIL. **Os 5 Princípios do Lean Thinking (Mentalidade Enxuta)**. Disponível em: [http://www.lean.org.br/5\\_principios.aspx](http://www.lean.org.br/5_principios.aspx). Acesso em: jun. 2014.

**Manifesto Ágil**. Disponível em: <<https://agilemanifesto.org/iso/ptbr manifesto.html>>. Acesso em: jun. 2021.

PÍCOLO, M. **Test Driven Development**. 2017. Disponível em: <<https://github.com/fga-eps-mds/A-Disciplina-MDS-EPS/wiki/Test-Driven-Development>>. Acesso em: 10/11/2020.

PFLEGER, S. L. **Engenharia de Software: Teoria e Prática**. 2.ed., São Paulo: Prentice Hall, 2004.

POPPENDIECK, M.; CUSUMANO, M. A. **Lean Software Development: A Tutorial**. IEEE Software, v. 29, n. 5, p. 26-32, 2012.

POSSAMAI, Edson. **A contribuição do Lean Manufacturing na Obtenção de Vantagem Competitiva: Estudo de Caso em Duas Empresas do Setor Moveleiro na Região de São Bento do Sul/SC**. 2011. 116 f. Dissertação (Mestrado) - Curso de Mestrado em Engenharia de Produção, Sociedade Educacional de Santa Catarina Instituto Superior Tupy, Joinville/SC, 2011.

PRESSMAN, R. S. **Engenharia de Software: Uma Abordagem Profissional**, 7. ed. Porto Alegre: Editora Mc GrawHill, 2011.

**Princípios por Trás do Manifesto Ágil**. Disponível em: <<https://agilemanifesto.org/iso/ptbr principles.html>>. Acesso em: jun. 2021.

QUINTÃO, P. L. **Tecnologia da Informação para Concursos**. 2023.

REDHAT. **O que é CI/CD?** 2022. Disponível em: <<https://www.redhat.com/pt-br/topics/devops/what-is-ci-cd>>. Acesso em jan. 2023.

SBROCCO, J. H. T. de C.; MACEDO, P. C. **Metodologias ágeis: engenharia de software sob medida**. 1. ed. - São Paulo: Érica, 2012.

SCHWABER, K. e Beedle, M. **Agile Software Development with Scrum**, Prentice Hall, 2002.

\_\_\_\_\_. **Agile Software Development**, Cockburn Highsmith Series Editors, Alistair Cockburn, 2000.

SOMMERVILLE, I., **Engenharia de Software**, 9. ed., São Paulo: Pearson Addison - Wesley, 2011.

SOTILE, M. **O Que é Scrumban?** Disponível em: <<https://blog.pmtech.com.br/scrumban/#>>. 2018. Acesso em: jun 2021.

STONE, K. B. **Four Decades of Lean: a Systematic Literature Review.** International Journal of LeanSix Sigma, v. 3, n. 2, p. 112-132, 2012.

TASCA, J. et al. **An approach for selecting a theoretical framework for the evaluation of training programs.** Journal of European Industrial Training, v.34, p. 631-655, 2010.

TELES, V. M. **Extreme Programming.** Novatec, 2004.

WOMACK, J. P.; JONES, D. T. **Lean thinking: Banish waste and create wealth in yourorganisation.** Simon and Shuster, New York, NY, v. 397, 1996.

WOMACK, J. P.; JONES, D. T.; ROOS, D. **Machine that changed the world. Simon and Schuster,** 1990.

---

### Patrícia Quintão

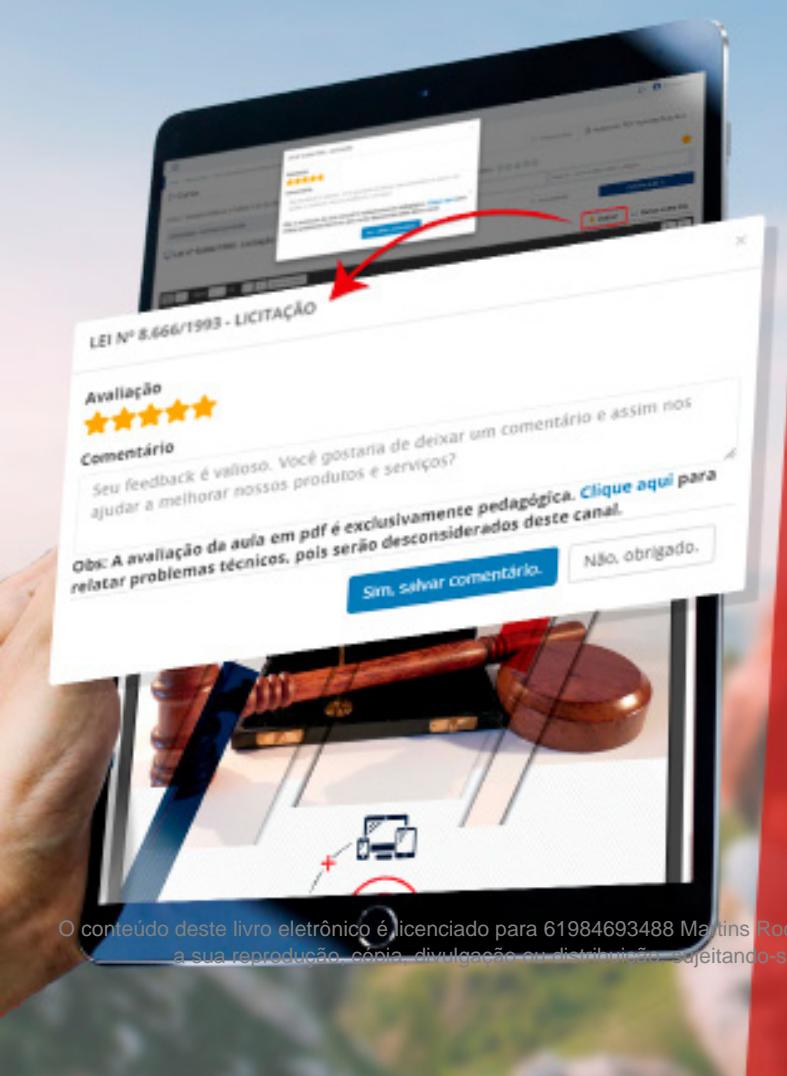


Mestre em Engenharia de Sistemas e computação pela COPPE/UFRJ, Especialista em Gerência de Informática e Bacharel em Informática pela UFV. Atualmente é professora no Gran Cursos Online; Analista Legislativo (Área de Governança de TI), na Assembleia Legislativa de MG; Escritora e Personal & Professional Coach.

Atua como professora de Cursinhos e Faculdades, na área de Tecnologia da Informação, desde 2008. É membro: da Sociedade Brasileira de Coaching, do PMI, da ISACA, da Comissão de Estudo de Técnicas de Segurança (CE-21:027.00) da ABNT, responsável pela elaboração das normas brasileiras sobre gestão da Segurança da Informação.

Autora dos livros: Informática FCC - Questões comentadas e organizadas por assunto, 3<sup>a</sup>. edição e 1001 questões comentadas de informática (Cespe/UnB), 2<sup>a</sup>. edição, pela Editora Gen/Método.

Foi aprovada nos seguintes concursos: Analista Legislativo, na especialidade de Administração de Rede, na Assembleia Legislativa do Estado de MG; Professora titular do Departamento de Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia; Professora substituta do DCC da UFJF; Analista de TI/Suporte, PRODABEL; Analista do Ministério Público MG; Analista de Sistemas, DATAPREV, Segurança da Informação; Analista de Sistemas, INFRAERO; Analista - TIC, PRODEMGE; Analista de Sistemas, Prefeitura de Juiz de Fora; Analista de Sistemas, SERPRO; Analista Judiciário (Informática), TRF 2<sup>a</sup> Região RJ/ES, etc. Redes Sociais: @coachpatriciaquintao (Instagram) /profapatriciaquintao (YouTube) / @plquintao (Twitter) / t.me/coachpatriciaquintao (Telegram)



## NÃO SE ESQUEÇA DE AVALIAR ESTA AULA!

SUA OPINIÃO É MUITO IMPORTANTE  
PARA MELHORARMOS AINDA MAIS  
NOSSOS MATERIAIS.

ESPERAMOS QUE TENHA GOSTADO  
DESTA AULA!

PARA AVALIAR, BASTA CLICAR EM LER  
A AULA E, DEPOIS, EM AVALIAR AULA.

**AVALIAR** 