UNIP
EDUCAÇÃO, POR SUA ESCOLHA

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 14

CQA/UNIP – Comissão de Qualificação e Avaliação da UNIP

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 14

Christiane Mazur Doi

Doutora em Engenharia Metalúrgica e de Materiais, Mestra em Ciências - Tecnologia Nuclear, Especialista em Cálculo e Matemática Aplicada, Especialista em Estatística Aplicada e Ciência de Dados, Especialista em Língua Portuguesa e Literatura, Especialista em Recursos Gramaticais para Revisão Textual, Engenheira Química e Licenciada em Matemática, com Aperfeiçoamento em Estatística e MBA em Engenharia Econômica. Professora titular da Universidade Paulista.

Larissa Rodrigues Damiani

Doutora em Ciências pelo programa de Engenharia Elétrica - Microeletrônica, Mestra pelo mesmo programa e Engenheira Elétrica - modalidade Eletrônica (ênfase em Telemática). Professora titular da Universidade Paulista.

Questão 1

Questão 1.[1]

Leia o texto a seguir.

A etapa de definição de requisitos é voltada para estabelecer quais as funções são requeridas pelo sistema e as restrições sobre a operação e o desenvolvimento do software. Os requisitos de software podem ser classificados como requisitos funcionais e não funcionais.

SOMMERVILLE, I. *Engenharia de Software*, 10. ed. São Paulo: Pearson Education, 2019 (com adaptações).

Considerando as informações do texto, assinale a alternativa em que o item é um requisito funcional.

A. O software deve ser operacionalizado no sistema Linux.
B. O tempo de desenvolvimento não deve ultrapassar seis meses.
C. O software deve emitir relatórios de compras a cada quinze dias.
D. O tempo de resposta do sistema não deve ultrapassar 30 segundos.
E. A base de dados deve ser protegida para acesso apenas de usuários autorizados.

1. Introdução teórica

A questão testa conhecimentos básicos a respeito de requisitos de sistemas de software. Vamos, a seguir, relembrar os principais conceitos envolvidos nesse tema.

Requisitos funcionais e requisitos não funcionais

Os requisitos estabelecem quais funcionalidades um software deve apresentar, assim como as restrições às quais ele será submetido. Os requisitos de sistema são comumente classificados em funcionais e não funcionais.

Os **requisitos funcionais** são as declarações dos serviços que o sistema deve prestar, de como o software deve reagir a determinadas entradas e de como deve se comportar em determinadas situações. De forma geral, eles dizem o que o software "deve fazer". Em alguns casos, os requisitos funcionais podem, também, estabelecer tarefas ou comportamentos que o sistema deve evitar.

Como exemplo, podemos imaginar um sistema de cadastro de clientes encomendado por uma empresa a um desenvolvedor. Os itens listados a seguir correspondem a alguns dos requisitos funcionais desse sistema, expressos em linguagem natural.

[1]Questão 9 – Enade 2021.

- Cada funcionário que utiliza o sistema deve ser identificado exclusivamente por seu número funcional, que corresponde a um código numérico de 6 dígitos.
- Um funcionário deve poder fazer a busca de clientes por nome e sobrenome.
- O sistema deve gerar, semanalmente, um relatório contendo a lista e a contagem do número de clientes cadastrados no período.

A imprecisão na especificação de requisitos pode levar a conflitos entre clientes e desenvolvedores. Por isso, requisitos devem ser bem especificados, sem ambiguidades.

Os **requisitos não funcionais**, conforme o nome sugere, são aqueles que não têm relação direta com as funcionalidades oferecidas pelo software. Esses requisitos especificam ou restringem as características do sistema como um todo. Eles podem ser relacionados a propriedades do sistema, como confiabilidade, tempo de resposta e uso de memória. Muitos requisitos não funcionais também surgem por necessidades dos usuários relacionadas a restrições orçamentárias, políticas organizacionais, necessidade de interoperabilidade com outros sistemas, normas de segurança ou legislação relativa à privacidade.

Como exemplo, vamos voltar a imaginar o sistema de cadastro de clientes. Os itens listados abaixo correspondem a alguns dos requisitos não funcionais desse sistema.

- O sistema deve ficar disponível para os funcionários de todas as filiais da empresa durante o expediente (de segunda a sexta, das 8h às 17h).
- O tempo que o sistema pode permanecer fora do ar no expediente não deve ultrapassar 5 segundos em qualquer dia.
- O sistema deve implementar providências para garantir a privacidade dos clientes cadastrados.
- O programa deve ser executado no sistema operacional Windows 11.

Sempre que possível, os requisitos não funcionais devem ser escritos adotando critérios quantitativos, de forma que possam ser testados objetivamente. Isso ajuda a evitar divergências interpretativas entre clientes e equipes de desenvolvimento. Como exemplo, considere o requisito a seguir.

- Os funcionários devem ser capazes de utilizar todas as funções do sistema com pouco tempo de treinamento.

Perceba que o significado de "pouco tempo" é subjetivo. Para o cliente, pode significar algumas horas de treinamento. Para os desenvolvedores, pode significar algumas semanas. Como fazemos, então, para contornar essa inconsistência? Devemos tornar o requisito objetivo, conforme exposto a seguir, com um critério quantitativo estipulado.

- Os funcionários devem ser capazes de utilizar todas as funções do sistema após três horas de treinamento.

2. Análise das alternativas

A – Alternativa incorreta.

JUSTIFICATIVA. O sistema operacional no qual o software deverá ser operacionalizado corresponde a um requisito não funcional, pois não corresponde diretamente a uma funcionalidade esperada para o software.

B – Alternativa incorreta.

JUSTIFICATIVA. O tempo disponibilizado para o desenvolvimento do software é um requisito não funcional, pois não corresponde diretamente a uma funcionalidade esperada.

C – Alternativa correta.

JUSTIFICATIVA. A emissão de relatórios periódicos é uma possível funcionalidade de um software. Logo, esse requisito é classificado como funcional.

D – Alternativa incorreta.

JUSTIFICATIVA. O tempo de resposta do sistema é um requisito não funcional, pois não corresponde diretamente a uma funcionalidade.

E – Alternativa incorreta.

JUSTIFICATIVA. O nível de proteção da base de dados é um requisito não funcional, pois não corresponde diretamente a uma funcionalidade.

3. Indicações bibliográficas

- HIRAMA, K. *Engenharia de software: qualidade e produtividade com tecnologia*. Rio de Janeiro: Elsevier, 2011.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018.

Questão 2

Questão 2.[2]

Leia o texto a seguir.

A engenharia de requisitos é uma área que inclui quatro subprocessos relacionados de alto nível. Esses subprocessos são: 1) avaliação se o sistema será útil para a empresa (estudo de viabilidade); 2) obtenção de requisitos (elicitação de requisitos); 3) conversão desses requisitos em alguma forma padrão (especificação); 4) verificação se os requisitos realmente definem o sistema que o cliente deseja (validação).

SOMMERVILLE, I. *Engenharia de Software*. São Paulo: Pearson Addison-Wesley, 2017 (com adaptações).

Uma equipe de Tecnologia da Informação (TI) de uma empresa de consultoria desenvolverá um software de Suporte Técnico para uma grande empresa fornecedora de equipamentos eletrônicos. O estudo de viabilidade do software já foi realizado e aprovado. A equipe de Tecnologia da Informação seguirá os três subprocessos seguintes de alto nível de engenharia de requisitos descritos no texto de Sommerville, ou seja, os subprocessos de elicitação de requisitos, especificação e validação. Para esses três subprocessos, quais são os artefatos que podem ser utilizados por essa equipe de Tecnologia da Informação?

A. Documento de entrevista com usuários; modelo de caso de uso para os requisitos funcionais; prototipação de telas.
B. Documento de estudo de viabilidade; modelo de caso de uso para os requisitos funcionais; prototipação de telas.
C. Matriz de rastreabilidade; modelo de caso de uso para os requisitos não-funcionais; prototipação de telas.
D. Documento de entrevista com usuários; modelo de caso de uso para os requisitos não-funcionais; matriz de rastreabilidade.
E. Documento de estudo de viabilidade; modelo de caso de uso para os requisitos funcionais; matriz de rastreabilidade.

1. Introdução teórica

A questão avalia conhecimentos a respeito das atividades envolvidas na engenharia de requisitos, abordando técnicas e notações utilizadas em cada uma dessas atividades. Vamos, a seguir, estudar o tema.

[2]Questão 18 – Enade 2021.

1.1. Atividades da engenharia de requisitos

Os requisitos de software são o conjunto de funcionalidades, características e restrições que um sistema deve atender para satisfazer às necessidades dos usuários e às metas do projeto.

A **engenharia de requisitos** é uma disciplina da área de engenharia de software que envolve três atividades fundamentais:

- a descoberta dos requisitos por meio da interação com os stakeholders (elicitação);
- a conversão desses requisitos em uma forma padrão (especificação);
- a averiguação de que os requisitos realmente definem o sistema que o cliente espera (validação).

O estudo de viabilidade não faz parte diretamente da engenharia de requisitos, mas é uma etapa preliminar importante no ciclo de vida de um projeto de software. O estudo de viabilidade ocorre antes do início das três atividades fundamentais da engenharia de requisitos, e está relacionado à decisão de determinar se é viável ou apropriado prosseguir com um projeto de software. Vamos, a seguir, estudar cada uma das atividades mencionadas.

1.2. Elicitação de requisitos

A **elicitação de requisitos** é a primeira atividade fundamental da engenharia de requisitos e envolve um contato direto com os stakeholders do sistema. Os **stakeholders** são as partes interessadas no sistema, como usuários finais, gerentes de produto, equipe de desenvolvimento, equipe de marketing, acionistas e autoridades reguladoras que certificam a aceitabilidade do sistema.

Os objetivos da elicitação de requisitos são compreender o trabalho que os stakeholders realizam e entender como eles usariam o novo sistema. As tarefas envolvidas nesse processo são elencadas a seguir.

1) Descoberta e compreensão dos requisitos: é o processo de interagir com os stakeholders para "descobrir" os seus requisitos de usuário.
2) Classificação e organização dos requisitos: é o processo de pegar um conjunto não estruturado de requisitos, agrupar os requisitos relacionados e organizá-los em grupos coerentes.

3) Priorização e negociação de requisitos: quando há o envolvimento de muitos stakeholders, os requisitos provavelmente entrarão em conflito. Este estágio envolve a resolução desses possíveis conflitos.
4) Documentação dos requisitos: um rascunho da documentação dos requisitos de usuário é feito neste estágio.

Na figura 1, vemos um diagrama que ilustra a atividade de elicitação de requisitos. Note que a figura sugere tarefas iterativas, com feedback contínuo de cada tarefa para as demais. O ciclo do processo começa com a descoberta dos requisitos e termina com a sua documentação, mas a compreensão que o analista tem dos requisitos aumenta a cada rodada do ciclo, que só deve terminar quando o documento de requisitos de usuário for produzido de forma definitiva. Os requisitos de usuário quase sempre são escritos em linguagem natural, complementados por diagramas e tabelas apropriados.

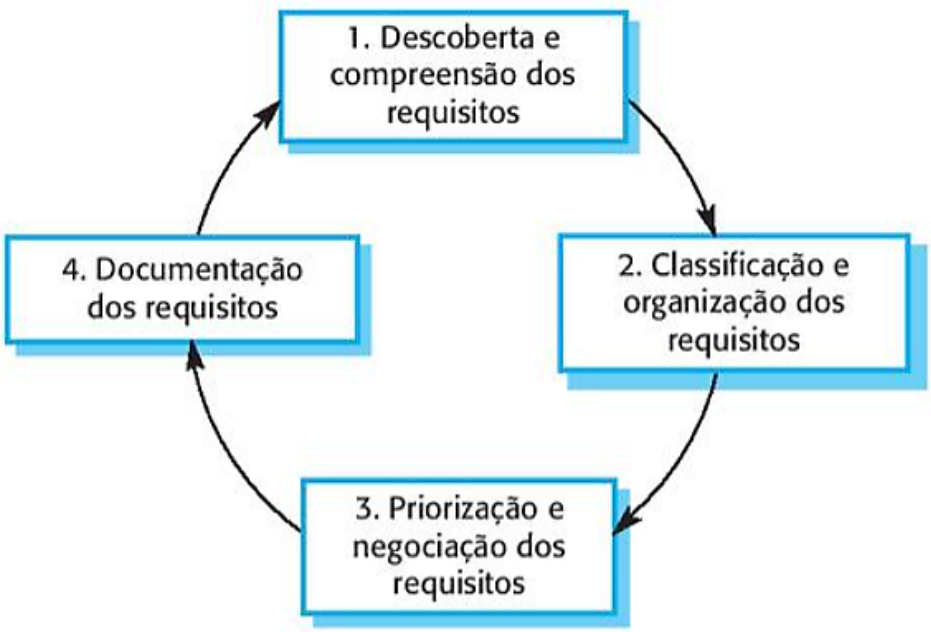

Figura 1. O processo de elicitação de requisitos.
SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018, p. 97.

As **técnicas de elicitação dos requisitos** envolvem encontros com os stakeholders para descobrir informações sobre o sistema proposto. É necessário investir tempo para entender como as pessoas trabalham e o que elas produzem. Existem duas abordagens fundamentais para a elicitação de requisitos, elencadas a seguir.

- Entrevistas: nesta abordagem, há uma conversa com as pessoas a respeito do que elas fazem.
- Observação ou etnografia: nesta abordagem, observam-se as pessoas executando seus trabalhos para ver quais ferramentas elas usam, como usam etc.

1.3. Especificação de requisitos

A **especificação de requisitos** é a segunda atividade fundamental da engenharia de requisitos. A atividade é destinada a escrever os requisitos de usuário do sistema em um documento de requisitos de sistema. Esses requisitos de sistema devem ser claros, completos e consistentes.

Os requisitos de usuário quase sempre são escritos em linguagem natural. Os requisitos de sistema são versões ampliadas dos requisitos de usuário, que os engenheiros de software utilizam como ponto de partida para o projeto do sistema de software.

Os requisitos de sistema também podem ser escritos em linguagem natural, mas outras notações baseadas em formulários, em gráficos ou em modelos matemáticos podem ser aplicadas. Essas notações garantem maior precisão e maior nível de detalhamento aos requisitos, quando comparados aos escritos em linguagem natural.

No quadro 1, são mostradas as possíveis notações que podemos utilizar para escrever requisitos de sistema.

Quadro 1. Notações pelas quais podemos escrever requisitos de sistema.

Notação	Descrição
Sentenças em linguagem natural	Por meio dessa notação, os requisitos são escritos usando frases numeradas em linguagem natural. Cada frase deve expressar um requisito.
Linguagem natural estruturada	Os requisitos são escritos em linguagem natural em um formulário ou template. Cada campo fornece informações sobre um aspecto do requisito.
Notações gráficas	Modelos gráficos, suplementados por anotações em texto, são utilizados para definir os requisitos funcionais do sistema. São usados, com frequência, os diagramas de casos de uso e de sequência da UML (Unified Modeling Language).
Especificações matemáticas	Essas notações baseiam-se em conceitos matemáticos, como as máquinas de estados finitos ou conjuntos.

SOMMERVILLE, 2018, p. 104 (com adaptações).

1.4. Validação de requisitos

A **validação de requisitos** é a terceira atividade fundamental da engenharia de requisitos. A atividade é destinada a conferir se os requisitos, tanto de usuário quanto de

sistema, definem o sistema que o cliente realmente quer. Essa atividade sobrepõe-se às etapas de elicitação e de análise, já que é voltada a encontrar problemas em cada uma delas. A validação de requisitos é muito importante, pois erros em um documento de requisitos podem levar a grandes custos de retrabalho, quando esses problemas são descobertos durante o desenvolvimento ou após o sistema entrar em funcionamento.

Algumas técnicas de validação de requisitos podem ser utilizadas individualmente ou em conjunto. Essas técnicas são resumidas no quadro 2.

Quadro 2. Técnicas de validação de requisitos.

Técnica	Descrição
Revisões de requisitos	Os requisitos são sistematicamente analisados por uma equipe de revisores, que procuram por erros e por inconsistências.
Prototipação	Essa técnica envolve o desenvolvimento de um modelo executável do sistema e o uso desse modelo com os usuários finais e clientes para verificar se ele satisfaz suas necessidades e expectativas. Os stakeholders experimentam o sistema e opinam sobre mudanças nos requisitos.
Geração de casos de teste	Os requisitos devem ser testáveis. Se os testes dos requisitos forem concebidos como parte do processo de validação, eles frequentemente revelarão os problemas existentes nos requisitos.

SOMMERVILLE, 2018, p. 113 (com adaptações).

2. Análise das alternativas

A – Alternativa correta.

JUSTIFICATIVA. A equipe de Tecnologia da Informação seguirá as três atividades de engenharia de requisitos: elicitação, especificação e validação. O documento de entrevista com usuários é um artefato da etapa de elicitação. O modelo de caso de uso para os requisitos funcionais é uma notação da etapa de especificação. Já a prototipação de telas é uma técnica da etapa de validação. Logo, esses artefatos podem ser utilizados por essa equipe.

B – Alternativa incorreta.

JUSTIFICATIVA. O documento de estudo de viabilidade não faz parte das atividades de elicitação, de especificação ou de validação de requisitos. O estudo de viabilidade é realizado antes desses processos.

C e D – Alternativas incorretas.

JUSTIFICATIVA. As mudanças no ambiente de negócios, nas organizações e nas tecnologias nos levam a mudanças nos requisitos de um sistema de software. O gerenciamento de requisitos é o processo de gerenciar e controlar essas mudanças. Nesse contexto, podem ser utilizadas as matrizes de rastreabilidade, que associam os requisitos às suas origens e os rastreia durante todo o ciclo de vida do projeto. Logo, as matrizes de rastreabilidade não são um artifício específico das etapas de elicitação, de especificação e de validação de requisitos. Além disso, diagramas de casos de uso, de forma geral, descrevem as funcionalidades propostas para o sistema. Logo, seu foco são os requisitos funcionais.

E – Alternativa incorreta.

JUSTIFICATIVA. O documento de estudo de viabilidade não faz parte das atividades de elicitação, de especificação ou de validação de requisitos. Além disso, as matrizes de rastreabilidade não são um artifício específico das etapas de elicitação, de especificação e de validação de requisitos.

3. Indicações bibliográficas

- HIRAMA, K. *Engenharia de software: qualidade e produtividade com tecnologia*. Rio de Janeiro: Elsevier, 2011.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018.

Questão 3

Questão 3.[3]

Leia o texto a seguir.

Arquitetura de software é uma representação que permite analisar a efetividade do projeto no entendimento dos requisitos declarados. Durante a fase de concepção da arquitetura, podem-se considerar alternativas de arquitetura em um estágio em que mudanças ainda são realizadas com menor esforço, diminuindo riscos associados à construção do software ainda na fase inicial.

PRESSMAN, R. S. *Engenharia de Software*: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016 (com adaptações).

A respeito dos estilos e padrões arquiteturais contidos na engenharia de software, avalie as afirmativas.

I. Em arquiteturas orientadas a objetos, a comunicação entre os componentes do software é realizada por intermédio da troca de mensagens.
II. As arquiteturas monolíticas consistem de um sistema dividido em pequenas partes, possibilitando que estas tenham sua manutenção, execução e evolução individual.
III. No padrão arquitetural Modelo-Visão-Controle (MVC), a camada de Modelo armazena as interações realizadas no Controle, podendo ser apresentados/manipulados posteriormente na Visão.
IV. Nas arquiteturas microsserviços, o software possui componentes altamente acoplados, dificultando a manutenção.

É correto apenas o que se afirma em

A. I e III.
B. II e III.
C. II e IV.
D. I, II e IV.
E. I, III e IV.

1. Introdução teórica

A questão testa conhecimentos a respeito de alguns modelos arquiteturais de software, estudados em disciplinas correlatas à área de engenharia de software. Vamos, a seguir, relembrar o tema.

[3]Questão 19 – Enade 2021.

1.1. Arquitetura de software

Do mesmo modo como a planta baixa de uma casa é apenas uma representação do edifício físico, a representação da arquitetura de software não é o software em si. Ela é apenas uma representação, que permite:

- analisar o atendimento dos requisitos;
- considerar alternativas de arquitetura em um estágio em que fazer mudanças de projeto ainda é fácil;
- reduzir os riscos associados ao desenvolvimento do software.

A arquitetura de software oferece uma ampla visão do sistema que será construído. Ela representa a estrutura e a organização dos componentes de software, suas propriedades e as conexões entre eles. Os componentes de software incluem módulos de programas e as várias representações de dados manipuladas por eles.

De forma geral, um estilo de arquitetura fornece diretrizes gerais para o design do sistema, e um padrão de arquitetura oferece uma solução mais específica para um problema conhecido, de forma que um padrão pertença a um estilo. A seguir, vamos estudar alguns estilos de arquiteturas.

1.2. Arquiteturas monolíticas

Nas arquiteturas monolíticas, o sistema conta com baixa modularização, sendo enxergado e implementado como um único projeto. Todos os componentes de software em um sistema monolítico são interdependentes, devido aos mecanismos de troca de dados dentro do sistema.

Como exemplo, vamos imaginar um sistema para uma loja de calçados. Todos os usuários utilizarão o mesmo sistema, e será preciso que ele esteja dividido em: autenticação e perfis de usuários, compra de produtos, estoque e vendas.

No padrão monolítico, todos os módulos mencionados acima estarão em um único projeto. Se é feita uma alteração no serviço de vendas, todos os outros módulos do projeto também terão que subir novamente para o servidor junto com essa alteração.

Em tempo de execução, todos os módulos de um sistema são executados, pelo sistema operacional, como um processo único. A figura 1 ilustra um sistema de nove módulos (de M1 a M9) que segue o estilo arquitetural monolítico. Em tempo de execução, o sistema executa como um único processo, que é representado pelo quadrado que envolve os nove módulos.

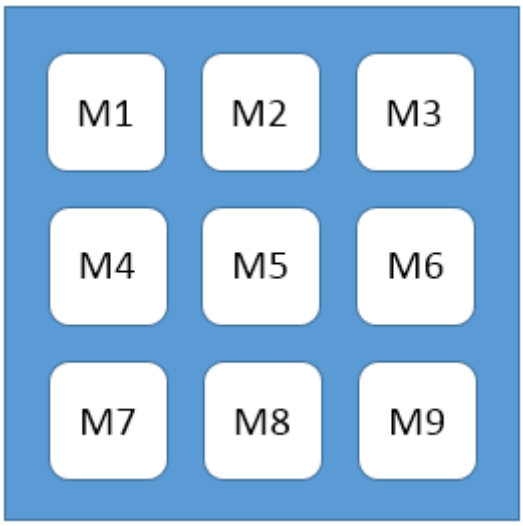

Figura 1. Sistema de nove módulos que segue o estilo arquitetural monolítico.
VALENTE, [s.d.] (com adaptações).

Em geral, um sistema monolítico é mais simples de ser desenvolvido, pois a organização fica concentrada em um único sistema. No entanto, é trabalhoso modificá-lo, pois pequenas mudanças afetam grandes áreas da base de código. Além disso, não há flexibilidade em relação à linguagem de programação: aquela que for escolhida no início do projeto terá que ser seguida para todos os módulos a serem implementados.

1.3. Arquiteturas em microsserviços

Para contornar as desvantagens dos sistemas em arquiteturas monolíticas, algumas empresas passaram a migrar os seus "monólitos" para arquiteturas baseadas em microsserviços. Nessas arquiteturas, fazemos com que certos módulos do sistema sejam executados em processos independentes, sem compartilhamento de memória entre eles. Nesse caso, o sistema é altamente modularizado não apenas em tempo de desenvolvimento, mas também em tempo de execução. Com isso, é menos provável que mudanças em um módulo causem problemas em outro.

O nome desse estilo de arquitetura refere-se a cada módulo como um serviço. Além disso, os serviços são "micro" porque cada um deles implementa apenas uma funcionalidade simples.

Na figura 2, podemos ver o diagrama que ilustra um sistema de nove módulos implementados por meio de uma arquitetura em microsserviços. Esses módulos são executados por seis processos independentes, representados pelos retângulos externos aos módulos. Os módulos M1, M2, M3 e M6 são executados individualmente, ou seja, cada um

em um processo independente. Os módulos M4 e M5 são executados em dupla. Por fim, os módulos M7, M8 e M9 são executados, em conjunto, em um processo.

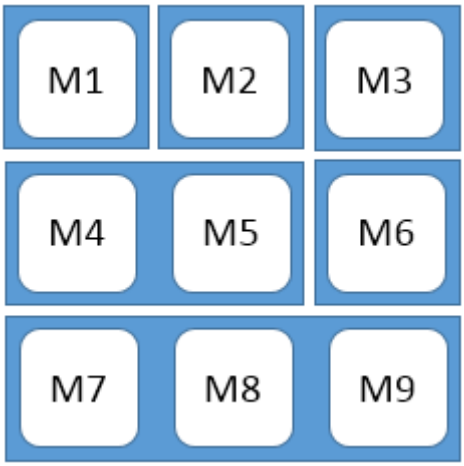

Figura 2. Sistema de nove módulos que segue o estilo arquitetural em microsserviços.
VALENTE, [s.d.] (com adaptações).

1.4. Arquiteturas orientadas a objetos

Nas arquiteturas orientadas a objetos, os componentes de um sistema encapsulam dados e as operações que devem ser aplicadas para manipular esses dados. A comunicação entre componentes é realizada por meio da passagem de mensagens.

Arquiteturas orientadas a objetos utilizam os princípios e os conceitos da programação orientada a objetos. Cada objeto tem uma interface definida pelos seus métodos, e a comunicação ocorre por meio da chamada desses métodos.

1.5. Padrão Modelo-Visão-Controle

O padrão arquitetural Modelo-Visão-Controle (MVC, do termo em inglês Model-View-Controller) pode ser classificado como um padrão de arquitetura orientado a objetos, que define que as classes de um sistema devem ser organizadas em três grupos, elencados a seguir.

- **Visão**: grupo de classes responsáveis pela apresentação da interface gráfica do sistema.
- **Controle**: conjunto de classes que tratam e interpretam eventos gerados por dispositivos de entrada, como o teclado ou o mouse. Como resultado da interpretação desses eventos, as classes de controle podem solicitar uma alteração no estado do modelo ou da visão.
- **Modelo**: grupo de classes que armazenam os dados manipulados pela aplicação e que têm a ver com o domínio do sistema, ou seja, com a lógica do negócio.

O relacionamento entre esses grupos de classes pode ser exemplificado conforme exposto na figura 3, que ilustra um sistema construído no modelo MVC.

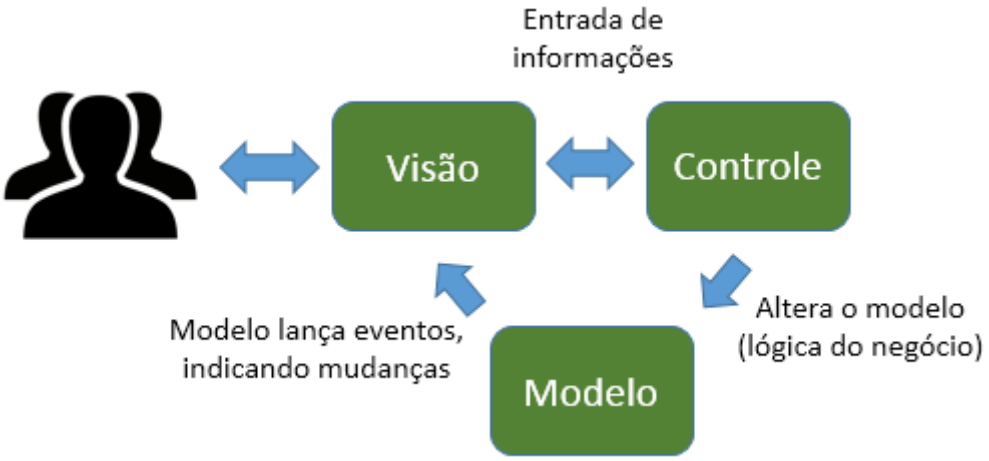

Figura 3. Fluxo de eventos e de informações em um sistema construído em modelo MVC. UNIVERSIDADE FEDERAL DE CAMPINA GRANDE, [s.d.]. Disponível em <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm >. Acesso em 19 abr. 2026 (com adaptações).

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. Nas arquiteturas orientadas a objetos, os objetos interagem entre si trocando mensagens. Cada objeto tem uma interface definida pelos seus métodos, e a comunicação ocorre por meio da chamada desses métodos.

II – Afirmativa incorreta.

JUSTIFICATIVA. Em arquiteturas monolíticas, o sistema é geralmente construído como uma única unidade. Se uma parte do sistema precisar de atualização, todo o sistema deve ser implantado novamente. Isso pode dificultar a manutenção de partes específicas do software.

III – Afirmativa correta.

JUSTIFICATIVA. No padrão orientado a objetos MVC, o modelo representa a camada de dados e lógica de negócios, o controle processa as interações do usuário e atualiza o modelo, e a visão exibe os dados armazenados no modelo.

IV – Afirmativa incorreta.

JUSTIFICATIVA. Nas arquiteturas de microsserviços, os módulos são projetados para serem independentes. Cada microsserviço é uma unidade que pode ser desenvolvida, implantada e

escalada separadamente. Isso facilita a manutenção, pois as alterações em um microsserviço não devem afetar os outros.

Alternativa correta: A.

3. Indicações bibliográficas

- HIRAMA, K. *Engenharia de software: qualidade e produtividade com tecnologia*. Rio de Janeiro: Elsevier, 2011.
- MORAIS, I. S. de (org.). *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2017. PENNA, W. *Arquitetura monolítica e microsserviços*. Disponível em <https://www.zappts.com.br/arquitetura-monolitica-e-microsservicos/>. Acesso em 19 abr. 2026.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018.
- UNIVERSIDADE FEDERAL DE CAMPINA GRANDE. Disponível em <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/mvc/mvc.htm>. Acesso em 19 abr. 2026.
- VALENTE, M. T. *Engenharia de software moderna*. Disponível em <https://engsoftmoderna.info/>. Acesso em 19 abr. 2026.

Questão 4

Questão 4.[4]

Leia o texto a seguir.

No decorrer de um projeto de desenvolvimento de software, é compreensível que mudanças venham a ocorrer, sejam por novos entendimentos dos atores envolvidos ou até mesmo de novas demandas apresentadas pelos clientes. Nessa perspectiva, a Gestão de Configuração de Software estabelece um conjunto de atividades para gerenciar alterações por todo o ciclo de vida de um software.

PRESSMAN, R. S. *Engenharia de Software*: uma abordagem profissional. 8. ed. Porto Alegre: Artmed Editora S.A. e McGraw-Hill Education, 2016 (com adaptações).

Com base no texto, assinale a opção correta acerca das atividades de gestão de configuração de software.

A. Identificação de itens na configuração de software, gerenciamento de alterações, validação de versões e controle de qualidade.
B. Identificação de objetos na configuração de software, controle de versão, controle de alterações, auditoria de configuração e relatório de status.
C. Gerência de projeto, gerência de configuração de software, ciclo de vida do projeto, processo de software e evolução das configurações.
D. Construção de modelo de requisitos, métricas para modelo de projeto, métricas de controle de versão e relatório de manutenção.
E. Identificação de alterações, controle de versão, mapeamento de alteração e notificação da alteração aos envolvidos.

1. Introdução teórica

A questão testa conhecimentos a respeito das atividades envolvidas na gestão de configuração de software. É importante frisar que essas atividades podem ser especificadas de diferentes formas por diferentes autores da área de engenharia de software. A nomenclatura presente nas alternativas da questão corresponde à utilizada por Roger Pressman e Bruce Maxim, no livro intitulado "Engenharia de Software: uma abordagem profissional". Vamos, a seguir, estudar um pouco desse conteúdo.

[4]Questão 21 – Enade 2021.

Gestão de configuração de software

Na construção de um sistema computacional, alterações são inevitáveis. Essas mudanças podem causar confusão quando não são analisadas antes de serem implementadas, ou quando não há boa comunicação entre os membros da equipe responsável pelo projeto.

À medida que o trabalho de engenharia de software avança, a equipe de software cria uma hierarquia de **itens de configuração de software** (SCIs, do termo em inglês software configuration items). Um SCI é uma entidade única identificável, que é gerenciada e documentada durante o ciclo de vida de um projeto. Essas entidades podem incluir códigos-fonte, documentos, scripts e outros elementos relacionados ao sistema. Os SCIs são chamados coletivamente de configuração de software.

Nesse cenário, a **gestão de configuração de software** (SCM, do termo em inglês software configuration management) é responsável por identificar, organizar e controlar modificações ao longo do clico de vida do software. O objetivo da SCM é maximizar a produtividade e minimizar possíveis erros.

O processo de gestão de alterações de software define uma série de tarefas que têm quatro objetivos principais:

- identificar todos os itens que definem a configuração do software;
- gerenciar alterações de um ou mais desses itens;
- facilitar a construção de diferentes versões de uma aplicação; e
- assegurar que a qualidade do software seja mantida conforme a configuração evolui.

Com o intuito de atingir esses objetivos, são definidas **cinco tarefas SCM**: identificação, controle de versão, controle de alteração, auditoria de configuração e elaboração de relatórios. Essas tarefas são mostradas no diagrama da figura 1. Conforme a disposição da figura, as tarefas SCM podem ser vistas como camadas. Os SCIs "fluem" para fora através dessas camadas por toda a sua vida útil, tornando-se, finalmente, parte da configuração do software de uma ou de várias versões do sistema de software.

Figura 1. Camadas do processo SCM.
PRESSMAN e MAXIM, 2021 (com adaptações).

A atividade de **identificação** envolve a identificação clara dos SCIs que estão sujeitos à configuração. A gestão de configuração requer a capacidade de identificar e de rastrear esses elementos específicos dentro do sistema.

A atividade de **controle de alterações** pode combinar procedimentos humanos e ferramentas automatizadas, proporcionando um mecanismo para o controle de alterações. Resumidamente, nessa atividade, uma solicitação de alteração é apresentada e avaliada. Os resultados da avaliação são apresentados como um relatório de alterações e, por fim, uma ordem de alteração é gerada para cada alteração aprovada.

Na atividade de **controle de versão**, é criada uma atualização do arquivo original logo que a alteração for feita. Alternativamente, os objetos a serem alterados podem ser "retirados" do banco de dados do projeto e, assim que a alteração é feita, os objetos são então "colocados" novamente no banco de dados, e são usados mecanismos de controle de versão apropriados para gerar a próxima versão do software. O controle de versão permite o acompanhamento das alterações feitas, facilitando o trabalho colaborativo e a gestão de diferentes versões do software.

A **auditoria de configuração** complementa a revisão técnica, avaliando o objeto de configuração quanto a características que, em geral, não são consideradas durante a revisão. A revisão técnica focaliza a exatidão técnica do objeto de configuração modificado. Os revisores avaliam o SCI para determinar a consistência com outros SCIs, omissões ou efeitos colaterais em potencial. A auditoria entra como complemento, verificando se a configuração

do software está em conformidade com as políticas, com os padrões e com os requisitos estabelecidos.

É produzido, também, o **relatório de status de configuração**, que reporta o que aconteceu no processo de modificações, quem fez as alterações, quando elas foram feitas e o que será afetado por elas. O relatório não necessariamente é gerado ao final do processo. Ele pode ser gerado em diferentes fases do desenvolvimento, para informar sobre o estado da configuração.

2. Análise das alternativas

A – Alternativa incorreta.

JUSTIFICATIVA. A identificação de itens na configuração de software é uma das atividades da SCM, assim como o gerenciamento de alterações. No entanto, o controle de qualidade não é uma atividade típica da SCM, pois esse é um conceito muito amplo e abrangente.

B – Alternativa correta.

JUSTIFICATIVA. As tarefas de identificação de objetos na configuração de software, controle de versão, controle de alterações, auditoria de configuração e relatório de status compõem o conjunto das cinco tarefas SCM.

C – Alternativa incorreta.

JUSTIFICATIVA. Nesta alternativa, temos conceitos muito amplos, como gerência de projeto e ciclo de vida, que não são atividades exclusivas da SCM.

D – Alternativa incorreta.

JUSTIFICATIVA. Nesta alternativa, temos aspectos como modelo de requisitos e métricas, que estão fora do escopo da gestão de configuração de software.

E – Alternativa incorreta.

JUSTIFICATIVA. As etapas de identificação de alterações, controle de versão, mapeamento de alteração e notificação da alteração aos envolvidos, de certa forma, descrevem superficialmente a sequência de atividades envolvidas na SCM. No entanto, a alternativa não especifica e não nomeia as cinco atividades estabelecidas.

3. Indicações bibliográficas

- MORAIS, I. S. de (org.). *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2017.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018.

Questão 5

Questão 5.[5]

Um ponto importante para fábricas de software é a garantia de que seus produtos e/ou serviços acompanhem as diversas mudanças ocasionadas pelo mundo corporativo. As restrições de orçamento ou cronograma, reestruturação do negócio, modificações de regras de negócio ou novas necessidades do cliente, geram mudanças inevitáveis. Nesse cenário, a utilização de um processo de Gerência de Configuração de Software (GCS) que contemple características necessárias para gerenciar e controlar a evolução de um software, por meio do controle formal de versão e solicitação de mudanças, é extremamente importante.

Sobre o processo de Gerência de Configuração de Software (GCS), avalie as afirmativas.

I. O controle de versão possibilita o compartilhamento de dados e a edição colaborativa, permitindo a gestão de diferentes ramos de desenvolvimento e possibilitando a existência de diferentes versões de forma simultânea.

II. A auditoria de configuração é uma atividade de garantia de qualidade do sistema que tem por objetivo assegurar que a qualidade do software seja mantida quando feitas alterações requisitadas.

III. O relatório de status de configuração possui informações sobre o GCS, tendo por objetivo manter o cliente informado sobre as alterações, e deve ser gerado ao final do processo.

IV. A GCS proporciona um conjunto de atividades de acompanhamento e controle de mudanças que é iniciado logo depois que o software for fornecido ao cliente e colocado em operação.

É correto apenas o que se afirma em

A. I e II.

B. II e III.

C. III e IV.

D. I, II e IV.

E. I, III e IV.

[5]Questão 22 – Enade 2021.

1. Introdução teórica

Gestão de configuração de software

A questão testa conhecimentos a respeito das atividades envolvidas na gestão de configuração de software. A nomenclatura presente nas alternativas da questão corresponde à utilizada por Roger Pressman e Bruce Maxim, no livro intitulado "Engenharia de Software: uma abordagem profissional".

Conforme abordamos na Introdução Teórica da questão 4, a **gestão de configuração de software** (SCM, do termo em inglês "software configuration management") é responsável por identificar, organizar e controlar modificações ao longo do clico de vida do software. No enunciado da presente questão, é usado o termo **gerência de configuração de software** (GCS), que corresponde ao mesmo conceito.

Se você não está confortável com esse assunto, consulte a Introdução Teórica da questão 4, onde as atividades envolvidas na GCS são detalhadas.

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. Na atividade de controle de versão, é criada uma atualização do arquivo original logo que a alteração for feita. Alternativamente, os objetos a serem alterados podem ser "retirados" do banco de dados do projeto e, assim que a alteração é feita, os objetos são então "colocados" novamente no banco de dados, e são usados mecanismos de controle de versão apropriados para gerar a próxima versão do software. O controle de versão permite o acompanhamento das alterações feitas, facilitando o trabalho colaborativo e a gestão de diferentes versões.

II – Afirmativa correta.

JUSTIFICATIVA. A auditoria de configuração é uma das atividades da GCS que verifica se a configuração do software está em conformidade com as políticas, com os padrões e com os requisitos estabelecidos. Desse modo, a auditoria de configuração tem o objetivo de assegurar que a qualidade do software seja mantida quando feitas alterações requisitadas.

III – Afirmativa incorreta.

JUSTIFICATIVA. O relatório de status de configuração não necessariamente é gerado apenas ao final do processo. Ele pode ser gerado em diferentes fases do desenvolvimento, para informar sobre o estado da configuração.

IV – Afirmativa incorreta.

JUSTIFICATIVA. As atividades de GCS geralmente acontecem desde as fases iniciais do desenvolvimento e continuam por todo o ciclo de vida do software. Portanto, tais atividades não são limitadas apenas ao período após o fornecimento do software ao cliente.

Alternativa correta: A.

3. Indicações bibliográficas

- MORAIS, I. S. de (org.). *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2017.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018.

Questão 6

Questão 6.[6]

Uma fábrica de software está realizando entrevistas para contratação de um profissional que esteja alinhado às exigências do atual mundo corporativo. Sabe-se que processos ágeis de desenvolvimento têm se tornado essenciais para empresas que desejam realizar entregas rápidas e frequentes de produtos e/ou serviços de software. Essa empresa possui uma equipe de desenvolvimento que faz uso de processos ágeis como o Scrum e eXtreme Programming (XP) e o acompanhamento por meio do quadro Kanban. Sendo assim, um conjunto de características deve ser verificado durante a entrevista para garantir que o candidato a ser contratado possua conhecimentos necessários para atuar juntamente a esta equipe.

Com base no texto e nos processos ágeis de desenvolvimento de software, avalie as afirmativas.

I. Métodos ágeis são baseados em ciclos iterativo e incremental que se concentram no desenvolvimento rápido e na flexibilidade às mudanças, com a participação do cliente no processo de software.

II. Uma forte característica da XP é a garantia da qualidade do código produzido e, para isso, os desenvolvedores produzem testes automatizados antes mesmo de codificar uma funcionalidade.

III. O planejamento no Scrum é baseado na elaboração dos itens do product backlog, que é uma lista de funcionalidades desejadas pelo cliente, sendo o Scrum Master o responsável por gerenciá-lo.

IV. O quadro Kanban permite monitorar a evolução das tarefas necessárias durante o processo ágil de desenvolvimento de software, possibilitando um acompanhamento de forma visual das atividades em construção.

É correto apenas o que se afirma em

A. II.

B. I e III.

C. I, II e IV.

D. I, III e IV.

E. II, III e IV.

[6]Questão 12 – Enade 2021.

1. Introdução teórica

A questão testa conhecimentos a respeito de métodos ágeis. Em especial, precisamos conhecer os métodos Scrum, eXtreme Programming e Kanban. Estudaremos esses tópicos na sequência.

1.1. Métodos ágeis

Um **processo de desenvolvimento de software** define um conjunto de passos, tarefas, eventos e práticas que devem ser seguidos por desenvolvedores na elaboração de um sistema. A adoção de um processo específico é particularmente importante em projetos atuais, pois os sistemas de software modernos são, majoritariamente, muito complexos para serem desenvolvidos por um único programador. Nesse caso, a equipe de desenvolvimento precisa seguir diretrizes para produzir softwares com qualidade e com produtividade. Essas diretrizes são estabelecidas pelo processo de desenvolvimento adotado.

Os primeiros processos de desenvolvimento de software, propostos na década de 1970, eram estritamente sequenciais, seguindo o modelo em cascata. Os projetos eram iniciados com a especificação de requisitos e avançavam até chegar às fases de implementação, de testes e de manutenção. A figura 1 ilustra um processo de desenvolvimento de software seguindo modelo cascata, no qual o sistema é implementado e testado apenas nas etapas finais.

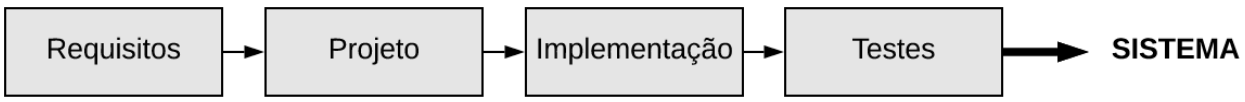

Figura 1. Desenvolvimento de software usando um processo baseado em modelo cascata.
VALENTE, [s.d.] M. T. *Engenharia de software moderna*. Disponível em <https://engsoftmoderna.info/>. Acesso em 15 dez. 2023.

Com o passar dos anos, profissionais da indústria de software propuseram alternativas aos processos em modelo em cascata. Eles afirmaram que um produto de software é diferente de produtos tradicionais de engenharia e, por isso, os softwares demandam um processo de desenvolvimento diferente.

Os requisitos de um software mudam com frequência, mais do que os requisitos de um computador ou de uma ponte, por exemplo. Adicionalmente, os clientes, muitas vezes, não têm uma ideia precisa do que querem, ou não conseguem expressar seus anseios adequadamente. Com isso, há o risco de se projetar por um longo tempo um produto de

software que, depois de pronto, não será mais necessário, ou porque o mundo mudou, ou porque as necessidades dos clientes mudaram.

Nesse cenário, uma das alternativas propostas pelos profissionais da indústria foi um novo modelo de processo de software, que são os **métodos ágeis**. Os métodos ágeis são processos de desenvolvimento de software baseados em ciclos iterativos e incrementais de trabalho, nos quais são incentivadas a colaboração e a comunicação entre todas as pessoas envolvidas no projeto. Desse modo, um sistema de software pode ser implementado de maneira gradativa, começando pelas partes mais urgentes para o cliente.

De forma geral, podemos imaginar que, inicialmente, a equipe implementa uma primeira versão do sistema, contendo as funcionalidades prioritárias. Em sequência, essa versão é validada pelo cliente e, caso seja aprovada, um novo ciclo de implementação se inicia. A segunda versão do sistema, portanto, engloba mais funcionalidades do que a primeira e também deve ser validada pelo cliente. O processo continua iterativamente, de forma incremental, até o cliente atestar que todos os requisitos foram atendidos. Etapas do processo podem ser realizadas em paralelo, e os processos ágeis costumam ser bem flexíveis em relação às suas etapas do processo de desenvolvimento.

A figura 2 ilustra um processo de desenvolvimento de software seguindo um método ágil, no qual o sistema é implementado, testado e validado em etapas incrementais. Na figura, cada iteração é representada por um retângulo e os incrementos no sistema são representados por S++. Observe que, a cada iteração, a equipe de desenvolvimento gera um incremento no sistema, que pode ser validado e testado pelos clientes antes da próxima iteração. Na prática, incrementos distintos podem ser desenvolvidos em paralelo, dependendo do tipo de projeto.

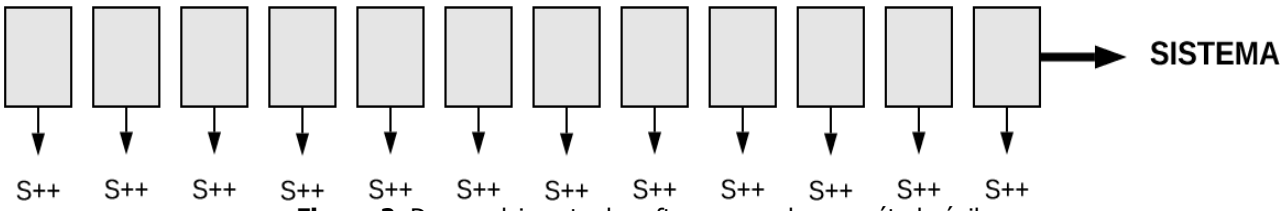

Figura 2. Desenvolvimento de software usando um método ágil.
VALENTE, [s.d.] M. T. *Engenharia de software moderna*. Disponível em <https://engsoftmoderna.info/>. Acesso em 15 dez. 2025.

Três dos principais processos de desenvolvimento de software que seguem a metodologia ágil são: Scrum, eXtreme Programming e Kanban. Estudaremos brevemente cada um deles, a seguir.

1.2. Scrum

O **método Scrum**, entre os métodos ágeis, é o mais conhecido e utilizado. Trata-se de um método para gerenciamento de projetos que, comumente, é aplicado a desenvolvimento de software, mas que não se restringe a essa aplicação. Como esse método pode ser aplicado a projetos em geral, o método não propõe qualquer prática de programação específica.

Na definição do Scrum, são estabelecidos papéis, eventos e artefatos. Veremos alguns detalhes a respeito desses três itens, a seguir.

Os **papéis** referem-se à especificação das pessoas envolvidas no projeto, com suas respectivas responsabilidades. São estabelecidos três papéis principais, elencados a seguir.

- **Product Owner**: tem o papel de representar os clientes. Para isso, o Product Owner deve entender do domínio do sistema que será construído, além de ser responsável por escrever os documentos que descrevem os requisitos do software, em linguagem natural.
- **Scrum Master**: é o especialista em Scrum do time, que é responsável por garantir que as regras do método estejam sendo seguidas.
- **Desenvolvedores**: trata-se de um grupo multifuncional, cujo número de integrantes varia entre 3 e 9. Os desenvolvedores são responsáveis por fazer a análise, a implementação, os testes e outras atividades pertinentes ao projeto.

Os **eventos** referem-se às atividades planejadas com o intuito de criar uma rotina de acompanhamento e de desenvolvimento do projeto. Os dois principais eventos são elencados a seguir.

- **Sprints**: assim como em todo método ágil, Scrum é um método iterativo. No Scrum, o nome dado para cada iteração é sprint. O método estabelece que cada sprint deve durar até um mês.
- **Planejamento de sprints**: é uma reunião na qual todo o time se reúne para decidir as histórias que serão implementadas no sprint que vai se iniciar. As histórias constituem uma descrição resumida das funcionalidades que devem ser implementadas no projeto, ou seja, estabelecem os requisitos dos usuários.

Os **artefatos** dizem respeito aos documentos que os membros do projeto criam e utilizam ao longo do seu desenvolvimento. Há dois artefatos oficiais, elencados a seguir.

- Product backlog: trata-se de uma lista de histórias e de outros itens de trabalho relevantes, ordenada por prioridades. Conforme já abordamos, as histórias são uma descrição

resumida das funcionalidades que devem ser implementadas no projeto. O product backlog é escrito pelo Product Owner.

- Sprint backlog: é o documento gerado ao final do evento de planejamento de sprint. Ele é constituído por uma lista de tarefas do sprint seguinte, com suas respectivas durações. Basicamente, o sprint backlog é uma lista de itens selecionados do product backlog, que devem ser implementados na próxima sprint.

1.3. eXtreme Programming

O método eXtreme Programming (XP) é um método ágil destinado ao desenvolvimento de softwares cujos requisitos sejam imprecisos ou estejam sujeitos a mudanças. Portanto, esse método, diferentemente do Scrum, é destinado especificamente para projetos de software.

O método envolve um conjunto de regras e práticas em suas iterações, que envolvem quatro atividades: planejamento, projeto, codificação e testes.

No **planejamento**, o representante do cliente cria as histórias de usuários, que descrevem o resultado, as características e a funcionalidade solicitados para o software. Cada história é colocada em uma ficha, e o cliente deve atribuir uma prioridade a cada uma delas. Os membros da equipe XP avaliam cada história e atribuem um custo (medido em semanas de desenvolvimento) a ela. A qualquer momento, podem ser escritas novas histórias.

O **projeto** XP estimula o uso de cartões CRC (classe-responsabilidade-colaborador) como uma ferramenta para pensar o software em um padrão orientado a objetos. Os cartões CRC identificam e organizam as classes relevantes para o incremento do software corrente. Os cartões CRC são, comumente, o único artefato de projeto produzido.

Após criar as histórias e elaborar o projeto, o time passará para a etapa de **codificação**. A equipe deve, primeiramente, desenvolver uma série de testes de unidades, que exercitarão as histórias que serão incluídas na versão atual do software. Uma vez criado o teste, os desenvolvedores podem se concentrar melhor no que deve ser implementado para que o software seja aprovado. Assim, com o código da versão atual do software completo, ele poderá ser testado imediatamente.

Na etapa de **testes**, a equipe precisa executar os testes de unidades, para corrigir eventuais falhas. Os testes de unidades devem ser implementados de forma automatizada pois, assim, poderão ser executados diversas vezes.

De acordo com o método XP, o desenvolvimento de projetos de software deve ser norteado por três **valores** principais, elencados a seguir.

- **Comunicação**: uma boa comunicação entre as partes envolvidas é importante em qualquer projeto, tanto para evitar que erros aconteçam quanto para aprender com os erros cometidos.
- **Simplicidade**: em todo sistema complexo, existem subsistemas mais simples, que devem ser considerados.
- **Feedback**: estar aberto ao feedback das partes envolvidas no projeto permite que correções de rota sejam implementadas o quanto antes, controlando os riscos aos quais o projeto está exposto.

Além dos valores, o XP é definido por um conjunto de princípios e de práticas de desenvolvimento, cujo extenso conteúdo não será abordado neste material.

1.4. Kanban

O método Kanban consiste em um processo visual, que utiliza cartões para descrever as histórias de usuário e os passos do processo de desenvolvimento do projeto. Quando comparamos o Kanban com o Scrum, percebemos que esse é um método mais simples, que não tem eventos, papéis pré-definidos ou diversos artefatos. O único artefato Kanban é o quadro de tarefas, que é chamado de **Quadro Kanban**. Esse quadro inclui, também, o product backlog.

O Quadro Kanban é dividido em colunas. A primeira coluna representa o próprio product backlog, com as histórias de usuários. As demais colunas são os passos que devem ser seguidos para transformar uma história em uma funcionalidade. Por exemplo, podemos criar as colunas Análise, Desenvolvimento e Testes. Além disso, cada coluna pode ser dividida em duas subcolunas: "tarefas em execução" e "tarefas concluídas". Por exemplo, a coluna Desenvolvimento pode ter duas subcolunas: "tarefas em desenvolvimento" e "tarefas desenvolvidas".

Portanto, a ideia do Kanban é que as histórias sejam processadas passo a passo, da esquerda para a direita, como em uma linha de montagem. Esse processo visual ajuda o time a identificar gargalos e a melhorar a eficiência do desenvolvimento do projeto. Podemos ver um exemplo de Quadro Kanban na figura 3, no qual duas colunas (Análise e Desenvolvimento) foram subdivididas.

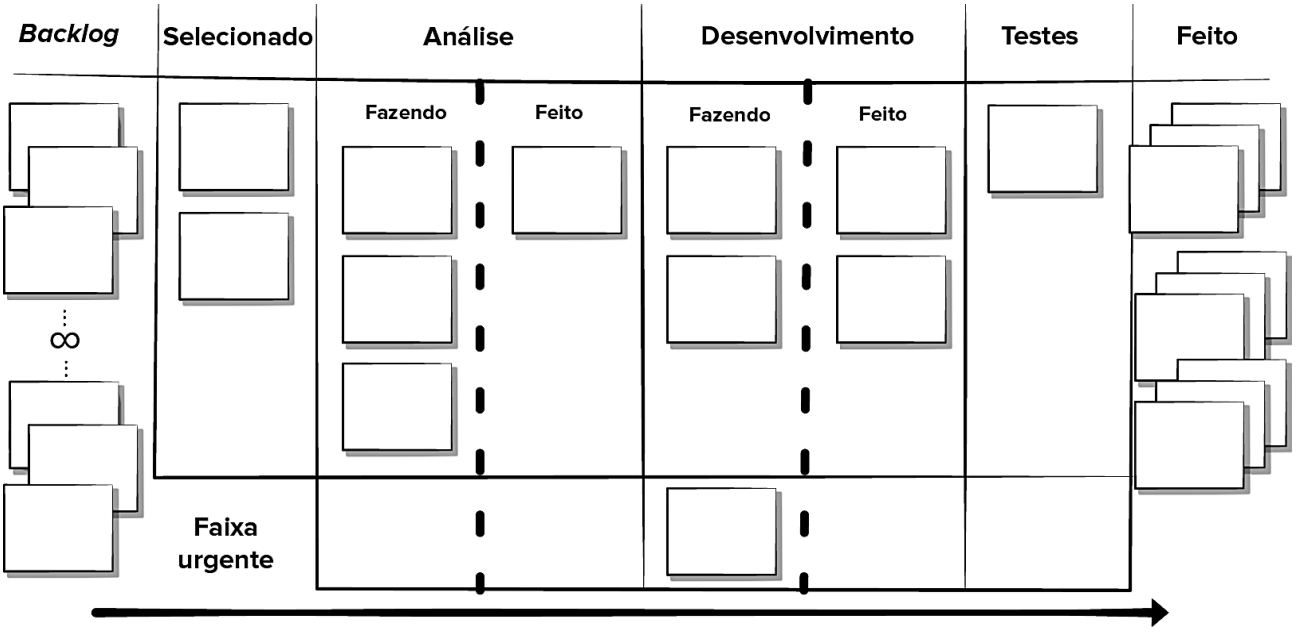

Figura 3. Exemplo de Quadro Kanban.
PRESSMAN e MAXIM, 2021.

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. Os métodos ágeis, como Scrum, XP e Kanban, são conhecidos por seu enfoque em ciclos iterativos e incrementais, buscando entregas rápidas e flexibilidade às mudanças. A participação do cliente é valorizada ao longo do processo.

II – Afirmativa correta.

JUSTIFICATIVA. O método XP incentiva a criação dos testes automatizados antes mesmo da codificação do software em si. Isso faz com que o desenvolvimento seja orientado a testes e garante a qualidade do código.

III – Afirmativa incorreta.

JUSTIFICATIVA. No método Scrum, o Product Owner é a pessoa responsável pela elaboração e pelo gerenciamento do product backlog, não o Scrum Master.

IV – Afirmativa correta.

JUSTIFICATIVA. O Quadro Kanban é uma ferramenta que permite o acompanhamento visual das tarefas de um projeto, ajudando a identificar gargalos e a melhorar a eficiência do desenvolvimento do projeto.

Alternativa correta: C.

3. Indicações bibliográficas

- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018.
- VALENTE, M. T. *Engenharia de software moderna*. Disponível em <https://engsoftmoderna.info/>. Acesso em 15 dez. 2025.

Questão 7

Questão 7.[7]

Uma determinada empresa do ramo de piscicultura solicitou a uma instituição especializada no desenvolvimento de softwares de gestão a construção de um sistema que fosse capaz de gerenciar o processo de criação dos peixes, que começava ainda na fase de alevinos, estendia-se para a engorda, finalizando na fase adulta com a venda do peixe. O sistema deveria disponibilizar funcionalidades para armazenar informações sobre toda a etapa da criação, mortandade e rações utilizadas na engorda dos peixes, viabilizando uma melhor tomada de decisão dos investidores. Como não havia nenhuma solução e a demanda por peixes estava aumentando, a empresa solicitou extrema urgência na construção da aplicação.

Considerando o processo de desenvolvimento de software que deverá ser utilizado para construir o sistema mencionado no texto, avalie as afirmativas.

I. Para desenvolver o sistema, deve ser adotado um processo baseado no modelo cascata, pois ele se adapta melhor a cenários com muitas mudanças.
II. Por se tratar de uma aplicação com escopo pequeno, o modelo espiral deverá ser utilizado no desenvolvimento do sistema solicitado, já que reduzirá riscos por meio da prototipação de especificações de cada fase dos peixes e diminuirá o tempo de entrega.
III. O modelo incremental deve ser utilizado na construção do software descrito no cenário, já que partes do sistema referentes à fase inicial dos peixes poderiam ser disponibilizadas, enquanto as demais estão nas etapas de concepção e desenvolvimento.
IV. Em razão da extrema urgência no desenvolvimento do software solicitado, a empresa deve elaborar seu processo utilizando o modelo RAD (Rapid Application Development).

É correto apenas o que se afirma em

A. I.
B. IV.
C. I e II.
D. II e III.
E. III e IV.

[7]Questão 25 – Enade 2021.

1. Introdução teórica

A questão requer conhecimentos a respeito de tipos distintos de modelos de processos de software. Abordaremos, a seguir, alguns deles: cascata, espiral e incremental. Adicionalmente, faremos algumas comparações com o modelo ágil, já abordado na Introdução Teórica da questão 6.

1.1. Modelos de processos de software

Conforme vimos na Introdução teórica 6, um processo de software pode ser definido como uma sequência de atividades, tarefas, métodos e práticas que são realizadas durante o desenvolvimento de um sistema de software. Nesse contexto, um **modelo de processos de software** é uma representação abstrata de como um processo de software deveria ser organizado e executado.

Um mesmo modelo pode dar origem a diferentes processos. Por exemplo, o modelo ágil representa um conjunto de abordagens colaborativas e iterativas, como Scrum, XP e Kanban.

Na prática, é possível combinar elementos de dois ou de diversos modelos de processo de software em um único processo personalizado, criando um modelo híbrido. Nesse caso, o objetivo é tirar proveito das vantagens de diferentes modelos para atender às necessidades específicas de um projeto ou de uma organização. Vamos estudar, a seguir, alguns dos modelos descritos na literatura de engenharia de software.

1.1.1. Modelo cascata

O modelo cascata prevê para o projeto um fluxo de trabalho previsível. Esse modelo é útil quando os requisitos do projeto são estáveis e foram bem compreendidos no início do projeto.

O modelo cascata traz uma abordagem linear e sistemática para o desenvolvimento do software, que começa com a especificação de requisitos de usuário (na etapa de comunicação) e avança para o planejamento, a modelagem, a construção e, finalmente, para a entrega. Na etapa de entrega, está previsto, também, o suporte ao usuário. No diagrama da figura 1, podemos ver as etapas previstas pelo modelo cascata.

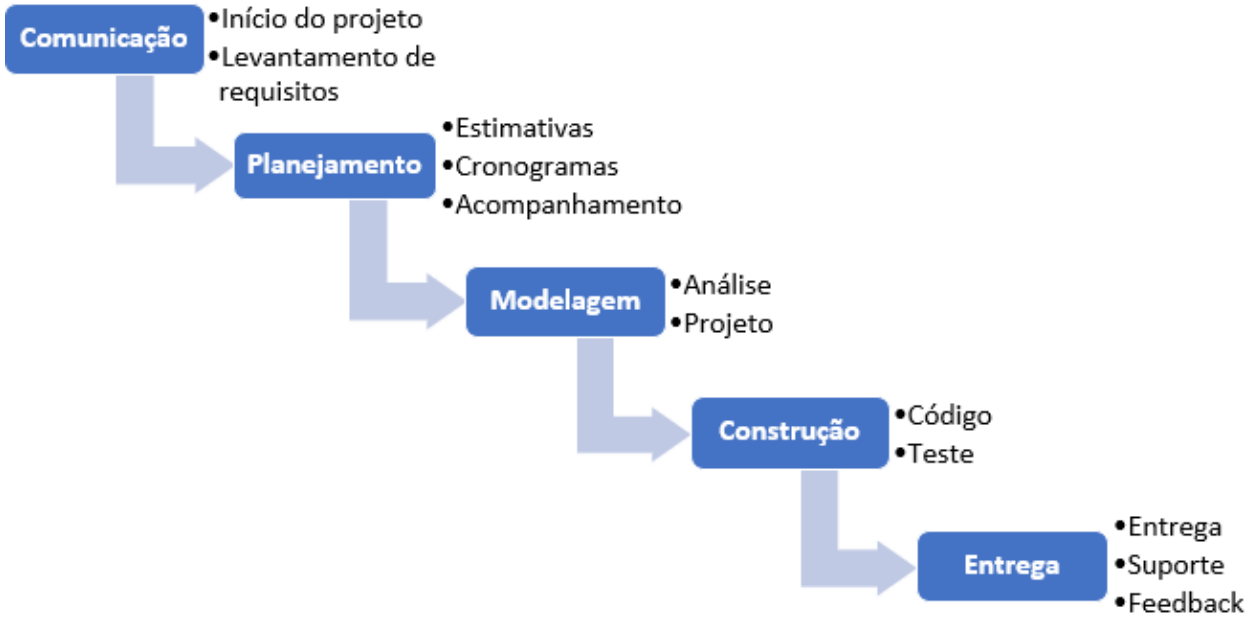

Figura 1. Modelo cascata.
PRESSMAN e MAXIM, 2021 (com adaptações).

1.1.2. Modelo espiral

O modelo espiral traz uma abordagem evolutiva ao desenvolvimento do software. Ele é adequado para casos em que um conjunto importante de requisitos é bem conhecido, mas precisa ser aprimorado ao longo da evolução do projeto. O modelo espiral é um modelo de processo de software que une o dinamismo da prototipação aos aspectos sistemáticos do modelo cascata. Processos que adotam esse modelo promovem o rápido desenvolvimento de versões cada vez mais completas do software desejado.

O modelo espiral é dividido em etapas, e cada uma constitui uma atividade definida pela equipe de engenharia de software. Para exemplificar, utilizaremos as etapas de comunicação, planejamento, modelagem, construção e entrega, conforme vemos na figura 2. Cada uma dessas atividades representa um segmento do caminho espiral ilustrado. Assim que esse processo começa, a equipe de software realiza atividades indicadas por um circuito em torno da espiral, no sentido horário, começando pelo seu centro.

O primeiro circuito em volta da espiral (iniciando na região da linha de fluxo mais próxima ao centro) pode resultar no desenvolvimento de uma especificação de produto. Passagens posteriores em torno da espiral podem ser usadas para desenvolver um protótipo e, então, progressivamente, versões cada vez mais sofisticadas do software. Cada passagem pela região de planejamento resulta em ajustes no planejamento do projeto. Custo e cronograma são ajustados de acordo com o feedback do cliente após a entrega. Além disso,

o gerente de projeto faz um ajuste no número de iterações planejadas para concluir o software.

O modelo espiral destaca a avaliação contínua dos resultados e as atividades de gerenciamento de riscos. Por isso, os riscos são levados em conta à medida que cada revolução é realizada. Ele usa a prototipação como mecanismo de redução de riscos. O modelo espiral, quando aplicado apropriadamente, reduz os riscos antes de eles se tornarem problemáticos.

Em comparação com o modelo ágil, o modelo espiral pode ser mais apropriado para projetos nos quais a gestão de riscos é crítica, enquanto o modelo ágil é mais adequado para projetos em que os requisitos estão sujeitos a mudanças frequentes.

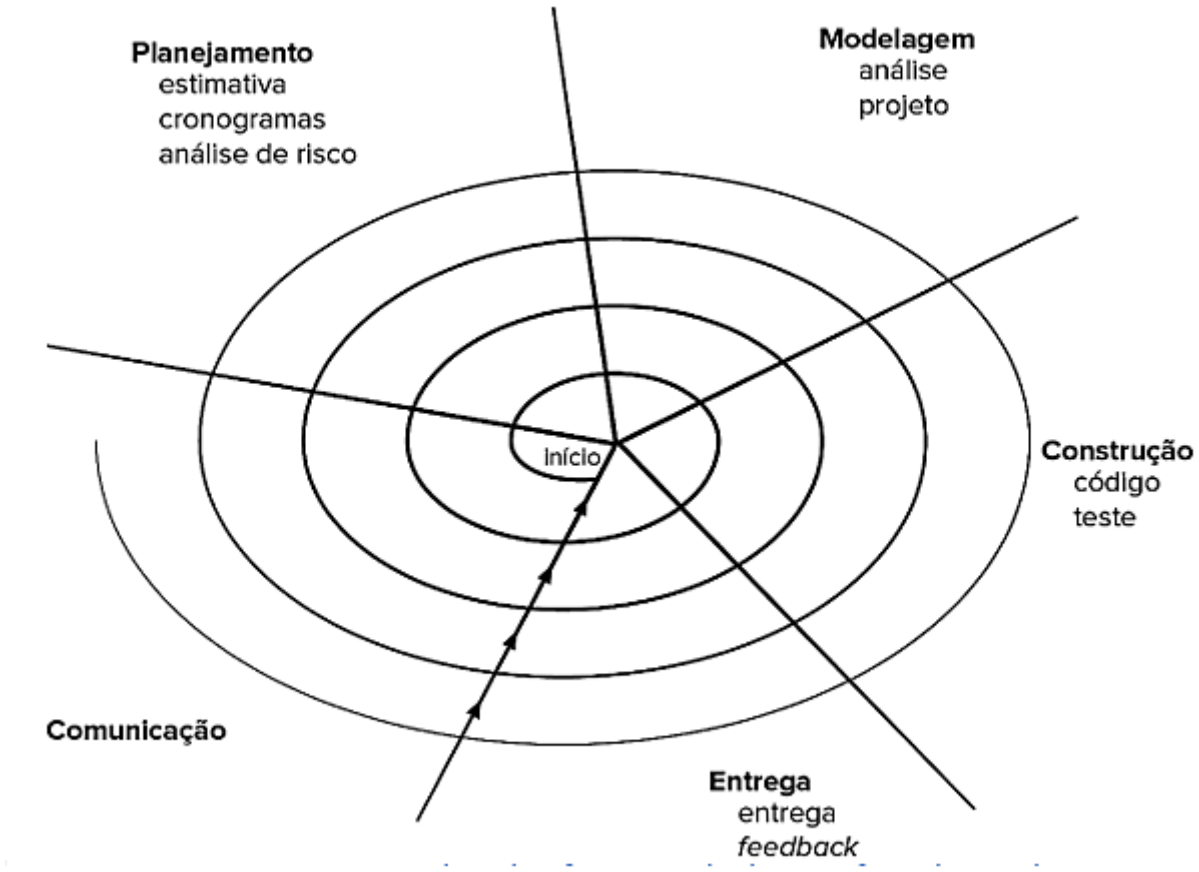

Figura 2. Modelo espiral.
PRESSMAN e MAXIM, 2021.

1.1.3. Modelo incremental

O modelo incremental traz uma abordagem evolutiva ao desenvolvimento do software. Geralmente, ele combina elementos dos fluxos de processos tanto lineares quanto paralelos.

Cada uma das sequências lineares gera um incremento do software, que pode ser apresentado aos clientes.

O modelo incremental aplica sequências lineares de etapas (como no modelo cascata) de forma escalonada, à medida que o tempo avança. Essas sequências podem ter etapas paralelas entre si, conforme podemos ver na figura 3. Cada uma das sequências lineares gera um incremento do software.

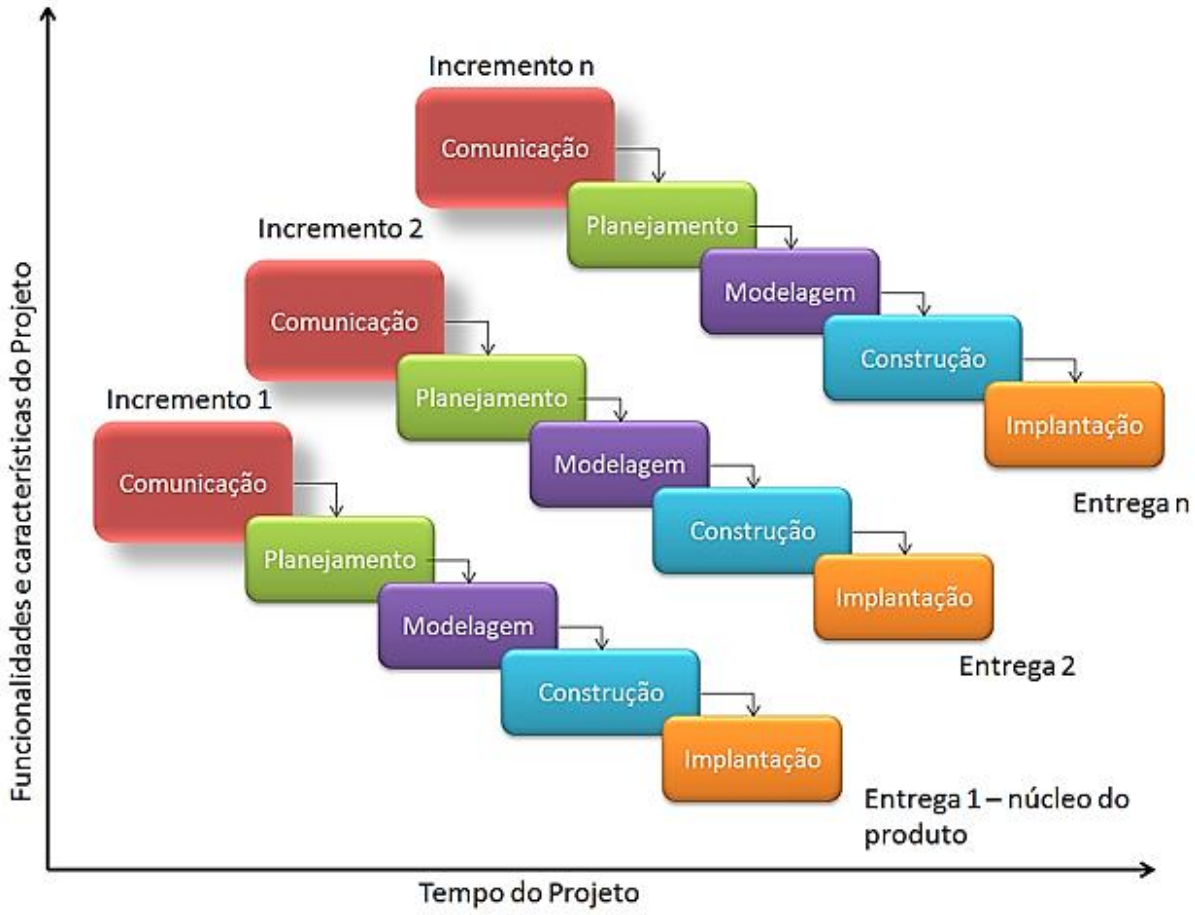

Figura 3. Modelo incremental.
Disponível em <http://mainedlystorm.weebly.com/blog/ciclo-de-vida-de-software-incremental>. Acesso em 19 jan. 2026.

Nesse modelo, é importante gerenciar adequadamente a coordenação entre as equipes e as diferentes partes do projeto, para garantir a integração dos incrementos. Além disso, é necessário considerar a comunicação eficiente e a resolução de possíveis conflitos que podem surgir quando várias partes do sistema estão sendo desenvolvidas simultaneamente, de forma paralela.

Apesar de poder ser considerado, por si, como um modelo de processo de software, o termo "incremental" pode ser utilizado para se referir a qualquer modelo que utilize incrementos em sua metodologia. Nesse sentido, o modelo ágil pode ser considerado como

um dos modelos incrementais. Enquanto o modelo ágil é geralmente considerado incremental, nem todos os processos baseados em modelo incremental são necessariamente ágeis.

O modelo ágil dá maior ênfase à flexibilidade, à colaboração e à entrega contínua, e o modelo incremental não ágil pode ser mais estruturado em fases, com entrega de incrementos mais definidos.

O modelo **RAD (Rapid Application Development)** também pode ser considerado como um tipo de modelo incremental. Esse modelo prioriza a entrega rápida do produto de software, por meio do desenvolvimento de protótipos e iterações, geralmente utilizando equipes multifuncionais. O RAD, portanto, é uma abordagem que visa à entrega rápida de um sistema por meio do uso intensivo de protótipos, ou seja, versões preliminares e simplificadas do sistema, que são desenvolvidas durante a fase inicial do desenvolvimento.

Podemos dizer que o RAD é uma abordagem específica dentro da categoria mais ampla de modelos incrementais. Como o RAD é conhecido por suas entregas rápidas, com a ênfase na obtenção de um produto funcional em um curto período, ele é um modelo adequado para o projeto descrito no enunciado da questão, já que a empresa solicitou extrema urgência na construção da aplicação.

2. Análise das afirmativas

I – Afirmativa incorreta.

JUSTIFICATIVA. O modelo cascata é mais adequado para projetos com requisitos estáveis, e não para cenários com muitas mudanças. Ele segue uma abordagem sequencial e pode ser menos flexível para acomodar alterações nos requisitos.

II – Afirmativa incorreta.

JUSTIFICATIVA. O modelo espiral é eficaz na gestão de riscos, mas não necessariamente auxiliar na redução do tempo de entrega, o que é mandatório para o projeto descrito no enunciado.

III – Afirmativa correta.

JUSTIFICATIVA. O modelo incremental é apropriado quando é possível entregar partes do sistema em fases. Isso permite que partes específicas do software estejam disponíveis enquanto outras estão sendo desenvolvidas, de forma paralela.

IV – Afirmativa correta.

JUSTIFICATIVA. O modelo incremental RAD é conhecido por sua capacidade de entrega rápida. Logo, ele é apropriado para modelar processos em situações em que há urgência no desenvolvimento do produto de software, como é o caso da situação descrita no enunciado.

Alternativa correta: E.

3. Indicações bibliográficas

- DEVMEDIA. *Introdução aos processos de software e o modelo incremental e evolucionário*. Disponível em <https://www.devmedia.com.br/introducao-aos- processos-de-software-e-o-modelo-incremental-e-evolucionario/29839>. Acesso em 19 jan. 2026.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.

Questão 8

Questão 8.[8]

Um projeto de integração de um ERP com um CRM não implementou todos os serviços esperados e há falhas nos serviços já implementados. Uma auditoria avaliou a situação e identificou que há desentendimentos entre os membros da equipe; os membros da qualidade não compreendem o que deve ser avaliado; demandas de novos solicitantes aparecem constantemente; e o patrocinador demora muito a responder às solicitações da equipe. Tudo isso fez com que o projeto ficasse bastante atrasado e com orçamento excedido.

Considerando o cenário descrito, o gerente de projeto deve priorizar o gerenciamento de

A. custo.
B. tempo.
C. escopo.
D. qualidade.
E. partes interessadas.

1. Introdução teórica

A questão retrata uma situação que diz respeito, diretamente, a uma das áreas de conhecimento estipuladas no guia PMBOK (Project Management Body of Knowledge).

Áreas de conhecimento do PMBOK

O guia PMBOK é o principal documento de referência para os profissionais da gestão de projetos. Nele, são descritas boas práticas que devem ser adotadas por gestores.

O PMBOK está dividido em 10 áreas de conhecimento, visando a apresentar os processos críticos para a gestão de projetos. De forma resumida e simplificada, cada uma dessas áreas é apresentada, a seguir.

- **Integração**: o gerenciamento da integração do projeto envolve tomadas de decisão ligadas aos objetivos do projeto, assim como o processo de desenvolvimento e de execução do plano de gerenciamento. De forma resumida, essa área faz a coordenação dos diversos elementos do projeto.

[8]Questão 14 – Enade 2021.

- **Escopo**: o gerenciamento do escopo é relativo ao atendimento dos requisitos para o desenvolvimento e a finalização do projeto. Entre as responsabilidades dessa área, estão a coleta e a definição dos requisitos.
- **Tempo**: o gerenciamento do tempo é relativo a processos que asseguram a conclusão do projeto nos prazos estabelecidos.
- **Custo**: a gestão dos custos visa a garantir que o projeto será concluído dentro do orçamento aprovado.
- **Qualidade**: o gerenciamento da qualidade objetiva garantir que o projeto seja concluído em conformidade com os requisitos e com as especificações definidas em seu escopo.
- **Recursos humanos**: o gerenciamento de recursos humanos visa a utilizar de forma eficaz a força de trabalho das pessoas envolvidas no desenvolvimento do projeto.
- **Comunicações**: o gerenciamento de comunicações inclui um conjunto de processos que asseguram a geração, a coleta, a distribuição, o armazenamento e o controle das informações do projeto.
- **Riscos**: o gerenciamento de riscos envolve a definição, a análise e a resposta aos riscos do projeto. Ele antecipa futuras necessidades ou mudanças, permitindo a tomada de decisão com antecedência.
- **Aquisições**: o gerenciamento de aquisições procede determinando o que, quando e como comprar ou adquirir bens ou serviços externos à organização.
- **Partes interessadas**: o gerenciamento de partes interessadas consiste em processos necessários para identificar, planejar, gerenciar e controlar as partes interessadas de um projeto. Essa área é responsável por coordenar as interações entre os stakeholders.

2. Resolução da questão

De acordo com o enunciado, uma auditoria identificou que há desentendimentos entre os membros da equipe. Esses membros são considerados como partes interessadas (ou stakeholders), pois fazem parte do grupo de pessoas que são influenciadas pelo projeto. O gerenciamento de partes interessadas é, justamente, responsável por coordenar as interações entre os stakeholders.

Alternativa correta: E.

3. Indicações bibliográficas

- KEELING, R.; BRANCO, R. H. F. *Gestão de projetos*. São Paulo: Saraiva Educação, 2019.
- SOUZA, C. P. da S. *Gestão de projetos*. Curitiba: Contentus, 2020.

Questão 9

Questão 9.[9]

Em razão da covid-19, muitos estabelecimentos viram no delivery uma forma de manter os negócios funcionando. As mudanças de hábitos dos brasileiros revelaram um aumento do interesse nesse tipo de serviço.

Nesse contexto, um restaurante solicitou o desenvolvimento de um sistema web que possibilite gerenciar automaticamente os pedidos de entrega. O sistema foi projetado com base em tecnologias web e modelado com UML (Unified Modeling Language). Durante o processo de desenvolvimento, foram descritos alguns itens importantes que merecem atenção no processo de teste de software.

Considere as seguintes descrições definidas pelo analista de teste:

1. Verificar se o método "Finalizar Pedido" da classe "Pedido" está funcionando corretamente.
2. Garantir que o sistema funcione corretamente em diferentes navegadores de internet.
3. O sistema deve passar por testes rigorosos na estrutura lógica interna do software.
4. O sistema deve garantir, no mínimo, o registro de 100 pedidos simultâneos.
5. O usuário deve testar o sistema em um ambiente controlado sob a supervisão dos desenvolvedores.

Considerando o texto e as descrições apresentadas, avalie as afirmativas.

I. A descrição 1 deve ser verificada com teste unitário.
II. A descrição 2 deve ser executada com a abordagem caixa-branca.
III. A descrição 3 deve ser executada com a abordagem caixa-preta.
IV. A descrição 4 deve ser verificada com teste carga.
V. A descrição 5 deve ser classificada como teste alfa.

É correto apenas o que se afirma em

A. II e III.
B. I, II e IV.
C. I, III e V.
D. I, IV e V.
E. II, III, IV e V.

[9]Questão 24 – Enade 2021.

1. Introdução teórica

A questão requer conhecimentos a respeito de tipos distintos de testes de software. Particularmente, precisamos estar familiarizados com as abordagens de testes (testes de caixa-preta e de caixa-branca), com os testes de desenvolvimento (testes de unidade, de componentes e de sistema), com os testes de release e com os testes de usuário (testes alfa, beta e de aceitação). Abordaremos a seguir, de forma sucinta, cada um desses testes mencionados.

1.1. Teste de software

De forma geral, um sistema de software passa pelos três estágios de teste elencados a seguir.

- **Estágio de testes de desenvolvimento**, no qual o sistema é testado ainda durante a etapa de desenvolvimento, geralmente pela própria equipe de programadores.
- **Estágio de testes de release**, em que uma equipe distinta testa uma versão completa do sistema, antes de ela ser entregue aos usuários, com o intuito de averiguar se a versão cumpre os requisitos impostos pelos stakeholders.
- **Estágio de testes de usuário**, no qual os potenciais usuários do sistema têm a oportunidade de testá-lo.

Além disso, os diversos tipos de testes podem ser classificados de acordo com duas abordagens: caixa-preta ou caixa-branca. Estudaremos, a seguir, essas abordagens.

1.1.1. Abordagem do teste

A abordagem do teste pode ser classificada como caixa-preta ou caixa-branca. Quando se usa uma técnica **caixa-preta**, os testes são escritos com base apenas na interface do sistema sob testes. Por isso, eles também são chamados de testes funcionais.

Por sua vez, na técnica **caixa-branca**, os testes devem considerar informações sobre o código-fonte e sobre a estrutura do sistema analisado. Por isso, eles também são chamados de testes estruturais.

1.1.2. Testes de desenvolvimento

O estágio de testes de desenvolvimento abrange as atividades de teste executadas pela equipe responsável pelo sistema. Nesse estágio, os testadores costumam ser os próprios programadores. O foco desses testes é procurar e corrigir eventuais bugs existentes no programa.

Um teste de desenvolvimento pode tanto seguir a abordagem caixa-preta como a abordagem caixa-branca, a depender da ênfase dada a ele.

Os testes de desenvolvimento são subdivididos em três etapas, elencadas a seguir.

- **Teste de unidade**: nesta etapa, são testadas unidades de programa ou classes individuais. O teste deve ser focado em verificar a funcionalidade e a qualidade dos objetos e de seus métodos.
- **Teste de componente**: nesta etapa, são testadas múltiplas unidades integradas, de forma a constituir componentes. O foco é verificar as interfaces dos componentes.
- **Teste de sistema**: nesta etapa, o sistema é testado mais amplamente, após a integração de alguns ou de todos os seus componentes. O foco é verificar as interações entre os diversos componentes.

1.1.3. Testes de release

Os testes de release são normalmente executados por uma equipe de testes específica, e não pelo time de desenvolvimento. O foco, nesse caso, é testar uma versão completa do sistema antes de ela ser entregue aos usuários, para averiguar se ela cumpre os requisitos impostos. Os testes de release, normalmente, seguem a abordagem caixa-preta, focando na funcionalidade do sistema como um todo.

O objetivo central do teste de release é convencer o fornecedor do sistema de que ele já se encontra em um estado bom para uso. Se comprovado, o sistema pode ser disponibilizado para o cliente.

Os testes de release podem seguir modelos distintos. Esses modelos são elencados a seguir.

- **Testes baseado em requisitos**: neste modelo, considera-se cada requisito estabelecido, a fim de derivar deles um conjunto de testes. Esse tipo de teste consiste em uma validação, que tem o objetivo de demonstrar que o sistema implementou cada um de seus requisitos corretamente.

- **Testes de cenário**: neste modelo, são criados cenários de usos típicos esperados, com o objetivo de empregá-los para desenvolver testes para o sistema. Os cenários devem ser realistas, para que os verdadeiros usuários possam se identificar com eles. Nesses cenários, devem ser observados quaisquer problemas que apareçam, inclusive problemas de desempenho, como lentidão excessiva. Geralmente, são testados diversos requisitos dentro do mesmo cenário.
- **Testes de desempenho**: neste modelo, os testes são desenvolvidos para garantir que o sistema possa processar sua carga pretendida, de forma a testar o desempenho e a confiabilidade dele. Comumente, isso envolve executar uma série de testes nos quais se aumenta a carga até o desempenho do sistema ficar inaceitável. Como exemplo, vamos considerar um sistema de processamento de transações que foi projetado para processar até 100 transações por segundo. Inicialmente, o teste começa com menos de 100 transações e, ao longo de sua execução, a carga é gradualmente aumentada, até o sistema falhar.

1.1.4. Testes de usuário

O estágio de testes de usuário corresponde ao processo em que os usuários fornecem entradas e conselhos sobre o teste do sistema. Esse estágio é essencial, pois as influências do ambiente de trabalho do usuário podem ter um importante efeito na confiabilidade e na usabilidade do sistema. Naturalmente, os testes de usuário seguem a abordagem caixa-preta. Há três tipos distintos de testes de usuário, elencados a seguir.

- **Teste alfa**: neste tipo, um grupo restrito de usuários trabalha sob a supervisão direta da equipe de desenvolvimento, com o intuito de testar lançamentos iniciais do sistema. Os testes, nesse caso, são feitos em um ambiente controlado.
- **Teste beta**: neste tipo de teste, uma versão do software é disponibilizada para um grupo mais abrangente de usuários. O intuito é que esses usuários testem o sistema em seus próprios ambientes e reportem aos desenvolvedores eventuais problemas encontrados. O teste beta é utilizado principalmente em sistemas aplicados a muitos contextos distintos e funciona, também, como uma ferramenta de marketing, já que os usuários aprendem sobre o sistema ao longo do processo.
- **Teste de aceitação**: este tipo de teste é uma parte intrínseca do desenvolvimento de sistemas personalizados, em que os clientes testam o sistema por conta própria e decidem

se ele deve ser aceito do desenvolvedor na versão em que se encontra ou não. De forma geral, a aceitação implica que o pagamento final pelo software deve ser realizado.

2. Análise das afirmativas

I – Afirmativa correta.

JUSTIFICATIVA. A descrição 1 é focada na verificação de um método específico de uma classe. Logo, o teste unitário (ou teste de unidade) é o mais adequado para essa verificação.

II – Afirmativa incorreta.

JUSTIFICATIVA. A descrição 2 não foca na estrutura interna do sistema, mas sim na verificação de um requisito não funcional. Logo, um teste que segue a abordagem caixa-branca não é indicado para essa verificação.

III – Afirmativa incorreta.

JUSTIFICATIVA. A descrição 3 diz que o sistema deve passar por testes na estrutura lógica interna do software. Essa abordagem de teste corresponde à técnica caixa-branca, e não à caixa-preta.

IV – Afirmativa correta.

JUSTIFICATIVA. A descrição 4 diz que o sistema deve garantir um número mínimo de pedidos simultâneos registrados. Queremos testar, nesse caso, um requisito não funcional, e a abordagem do teste de carga (ou testes de desempenho) é adequada para essa finalidade.

V – Afirmativa correta.

JUSTIFICATIVA. A descrição 5 diz que o usuário deve testar o sistema em um ambiente controlado, sob a supervisão dos desenvolvedores. Essa situação é justamente a imposta por um teste alfa, que é um dos tipos de testes de usuário.

Alternativa correta: D.

3. Indicações bibliográficas

- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.
- SOMMERVILLE, I. *Engenharia de software*. São Paulo: Pearson Education do Brasil, 2018.
- VALENTE, M. T. *Engenharia de software moderna*. Disponível em <https://engsoftmoderna.info/>. Acesso em 18 dez. 2025.

ÍNDICE REMISSIVO