UNIP
EDUCAÇÃO, POR SUA ESCOLHA

SISTEMAS DA INFORMAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 13

CQA/UNIP – Comissão de Qualificação e Avaliação da UNIP

SISTEMAS DA INFORMAÇÃO

MATERIAL INSTRUCIONAL ESPECÍFICO

TOMO 13

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.

José Carlos Morilla

Doutor e Mestre em Engenharia de Materiais, Mestre em Engenharia de Produção, Especialista em Engenharia Metalúrgica, Especialista em Física e Engenheiro Mecânico, com MBA em Gestão de Empresas. Professor adjunto 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.

Tarcísio de Souza Peres

Mestre em Ciências, Especialista em Gestão de Projetos e Engenheiro da Computação, com MBA em Gestão de TI. Cursou Gestão Estratégica e Competitividade. Professor adjunto da Universidade Paulista.

Tiago Guglielmeti Correale

Doutor em Engenharia Elétrica, Mestre em Engenharia Elétrica e Engenheiro Elétrico (ênfase em Telecomunicações). Professor titular da Universidade Paulista.

Questão 1

Questão 1.[1]

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.

[1]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 12 dez. 2025.

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.

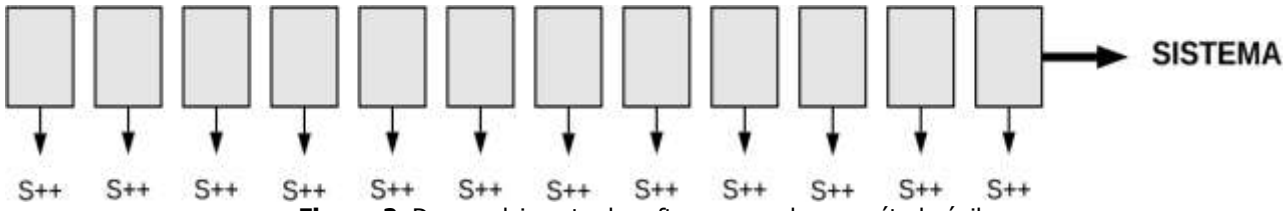


Figura 2. Desenvolvimento de software usando um método ágil.
VALENTE, [s.d.] M. T. *Engenharia de software moderna*. Disponível em <https://engsoftmoderna.info/>. Acesso em 12 dez. 2025.

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 Scrum é um *framework* leve destinado a ajudar pessoas, equipes e organizações a gerarem valor por meio de soluções adaptativas para problemas complexos. Embora seja frequentemente aplicado ao desenvolvimento de software, o Scrum não se restringe a essa área. Sua estrutura é definida por responsabilidades, eventos, artefatos e regras que os relacionam.

No Scrum, o trabalho é realizado por um *Scrum Team*, composto por:

- *Product Owner;*
- *Scrum Master;*
- *Developers*.

O *Product Owner* é responsável por maximizar o valor do produto resultante do trabalho do Scrum Team. Também é responsável pela gestão efetiva do *Product Backlog*, o que envolve ordenar seus itens, torná-lo transparente, comunicá-lo de forma clara e garantir que ele seja compreendido.

O *Scrum Master* também é responsável por estabelecer o *Scrum* conforme definido no guia oficial. Sua atuação envolve ajudar a equipe e a organização a compreenderem a teoria e a prática do *Scrum*, apoiar o *Product Owner* na gestão do *Product Backlog*, favorecer a efetividade do *Scrum Team* e remover impedimentos ao progresso do trabalho.

Os *Developers* são os integrantes *do Scrum Team* comprometidos com a criação de um incremento utilizável a cada *sprint*. Eles planejam o trabalho da *sprint*, adaptam diariamente seu plano em direção ao objetivo da *sprint* e se responsabilizam pela qualidade do incremento produzido.

Os eventos do *Scrum* criam regularidade e favorecem a inspeção e a adaptação. A sprint é o ciclo de trabalho em que as ideias são transformadas em valor. Há a duração fixa de até um mês. O planejamento da *sprint* define por que a *sprint* é valiosa, o que pode ser feito durante ela e como o trabalho escolhido será realizado.

Os artefatos do Scrum são:

- *Product Backlog*;
- *Sprint Backlog*;
- *Increment*.

O *Product Backlog* é uma lista emergente e ordenada daquilo que é necessário para melhorar o produto.

O *Sprint Backlog* reúne o objetivo da *sprint*, os itens selecionados do *Product Backlog* e o plano para entregar o incremento.

O *Increment* corresponde ao resultado integrado e utilizável produzido durante a *sprint*, desde que atenda à definição de pronto.

1.3. eXtreme Programming

O 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;
- 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.

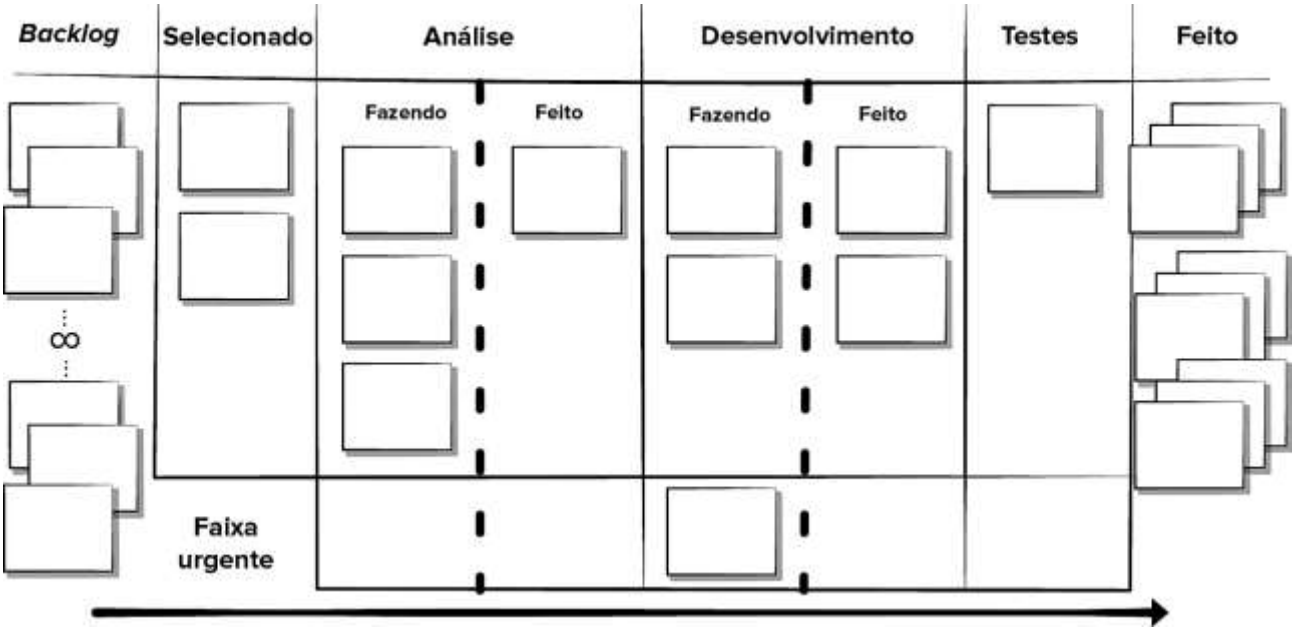


Figura 3. Exemplo de Quadro Kanban.
PRESSMAN e MAXIM, 2021.

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 Scrum, o *Product Owner* é responsável pela gestão efetiva do *Product Backlog*. O *Scrum Master* ajuda a equipe e a organização a compreenderem e aplicarem o Scrum, mas não é o responsável por gerenciar o Product Backlog.

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 12 dez. 2025.

Questão 2

Questão 2.[2]

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.

[2]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.

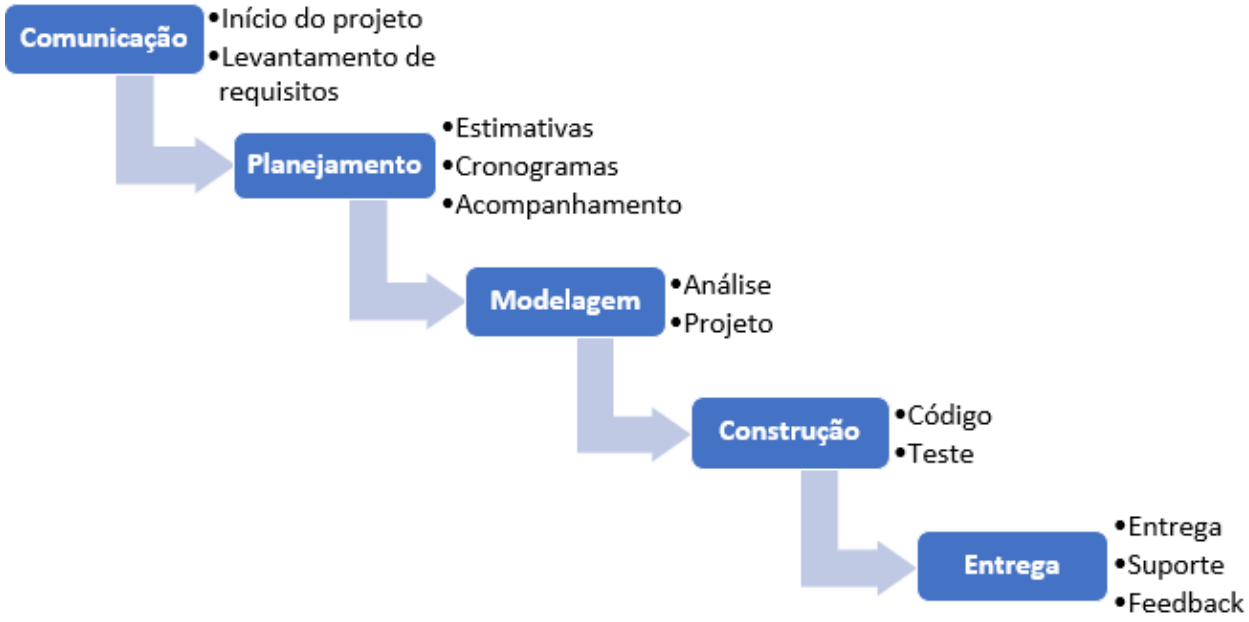


Figura 1. Modelo cascata.
PRESSMAN e MAXIM, 2021 (com adaptações).

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.

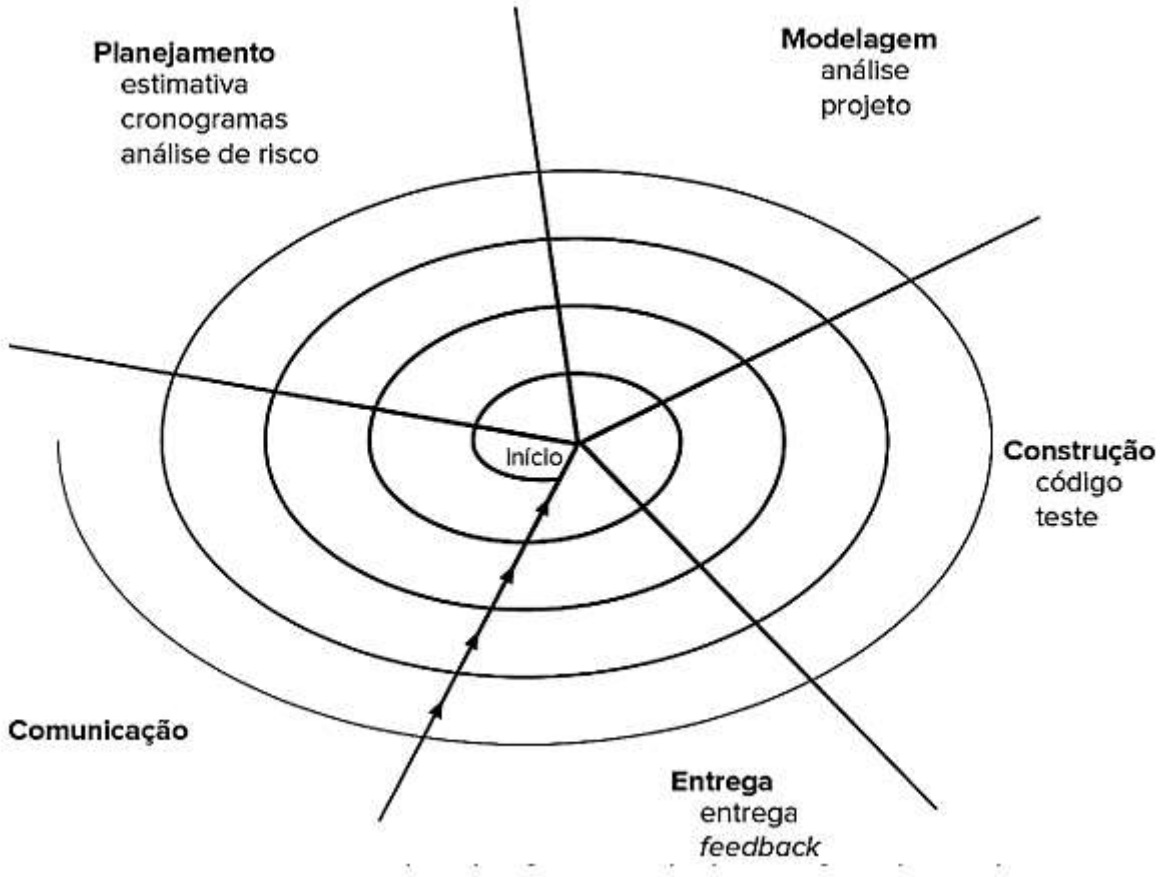


Figura 2. Modelo espiral.
PRESSMAN e MAXIM, 2021.

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.

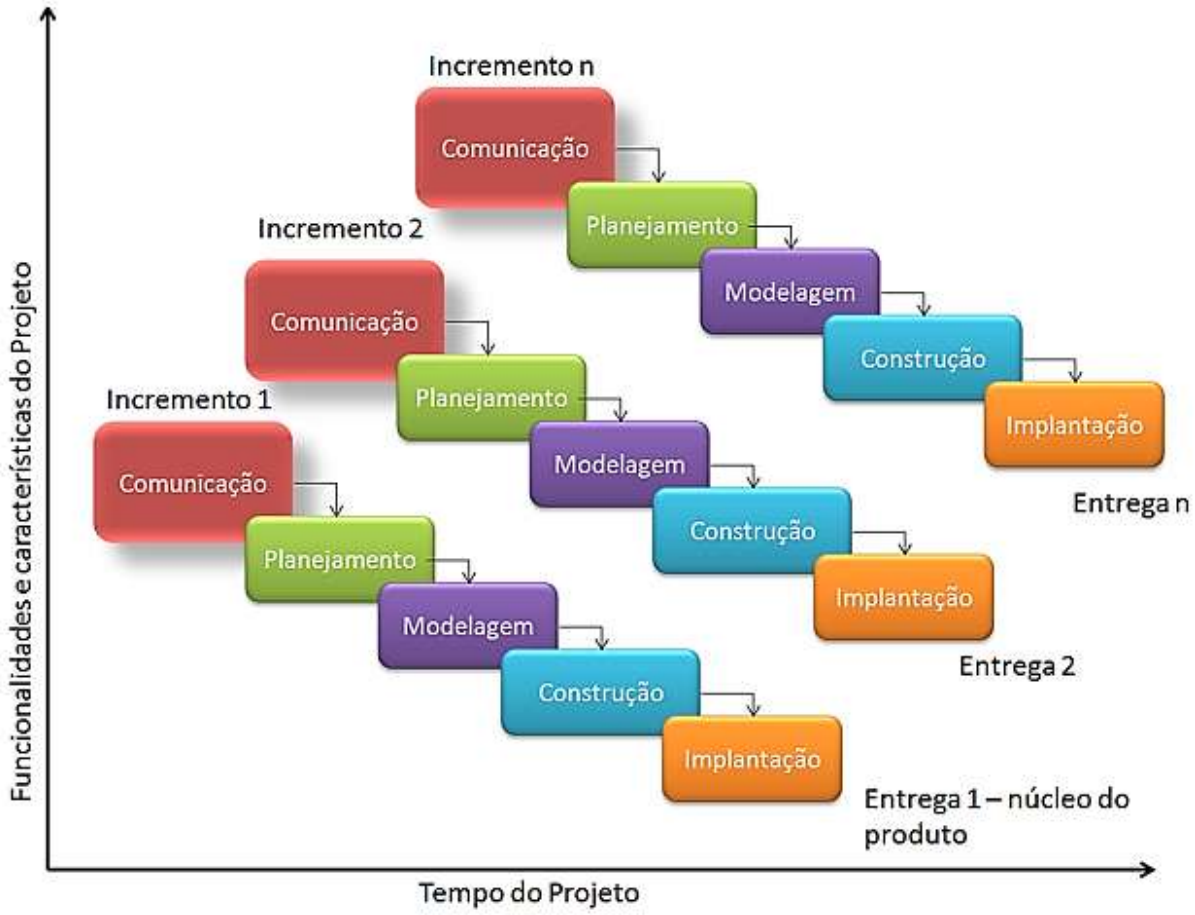


Figura 3. Modelo incremental.
Disponível em <http://mainedlystorm.weebly.com/blog/ciclo-de-vida-de-software-incremental>. Acesso em 12 dez. 2025.

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 auxilia 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 12 dez. 2025.
- PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de software: uma abordagem profissional*. Porto Alegre: AMGH, 2021.

Questão 3

Questão 3.[3]

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

Guia PMBOK

O Guia PMBOK, em sua 8ª edição, apresenta boas práticas, conceitos e orientações para a gestão de projetos em diferentes contextos organizacionais. A edição atual estrutura o gerenciamento de projetos a partir de princípios centrais e domínios de desempenho, buscando relacionar mentalidade de gestão, práticas técnicas e adaptação ao contexto do projeto.

Os princípios de gerenciamento de projetos orientam comportamentos esperados na condução do trabalho. Entre eles, destacam-se:

- a adoção de uma visão holística;
- o foco na entrega de valor;
- a incorporação da qualidade aos processos e aos entregáveis;
- a liderança responsável;
- a integração da sustentabilidade;
- a construção de uma cultura de equipe empoderada.

[3]Questão 14 – Enade 2021.

Os domínios de desempenho representam áreas críticas para a entrega efetiva dos projetos. No PMBOK 8, esses domínios são:

- governança;
- escopo;
- cronograma;
- finanças;
- partes interessadas;
- recursos;
- riscos.

Cada domínio reúne atividades, decisões e práticas que precisam ser consideradas de forma integrada, pois o desempenho de um projeto depende da articulação entre esses elementos.

O domínio de partes interessadas concentra-se na identificação, na compreensão, no engajamento e na gestão das pessoas, dos grupos ou das organizações que podem influenciar o projeto ou ser influenciados por seus resultados. A gestão adequada das partes interessadas envolve compreender expectativas, interesses, responsabilidades, necessidades de comunicação, formas de participação e possíveis impactos das decisões do projeto sobre esses atores.

Como projetos englobam diferentes grupos com interesses, responsabilidades e níveis de influência variados, o desempenho do projeto depende da capacidade de manter alinhamento, comunicação adequada e engajamento produtivo entre as partes interessadas. Falhas nessa dimensão podem afetar decisões, prioridades, entendimento do trabalho, aceitação das entregas e fluidez da execução.

2. Resolução da questão

O cenário descreve problemas de alinhamento entre diferentes atores do projeto: há desentendimentos entre membros da equipe, dificuldade de compreensão por parte da qualidade, surgimento constante de demandas de novos solicitantes e demora do patrocinador em responder às solicitações. Esses elementos indicam falhas de comunicação, de engajamento e de coordenação entre partes interessadas.

Embora o projeto também apresente atraso, estouro de orçamento e problemas de qualidade, esses efeitos decorrem, no cenário apresentado, de dificuldades de interação e de alinhamento

entre os envolvidos. Portanto, o gerente de projeto deve priorizar o gerenciamento de partes interessadas.

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.
- PROJECT MANAGEMENT INSTITUTE. *A guide to the project management body of knowledge (PMBOK Guide)*. 8th ed. Newtown Square: Project Management Institute, 2025.

Questão 4

Questão 4.[4]

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.

[4]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 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.

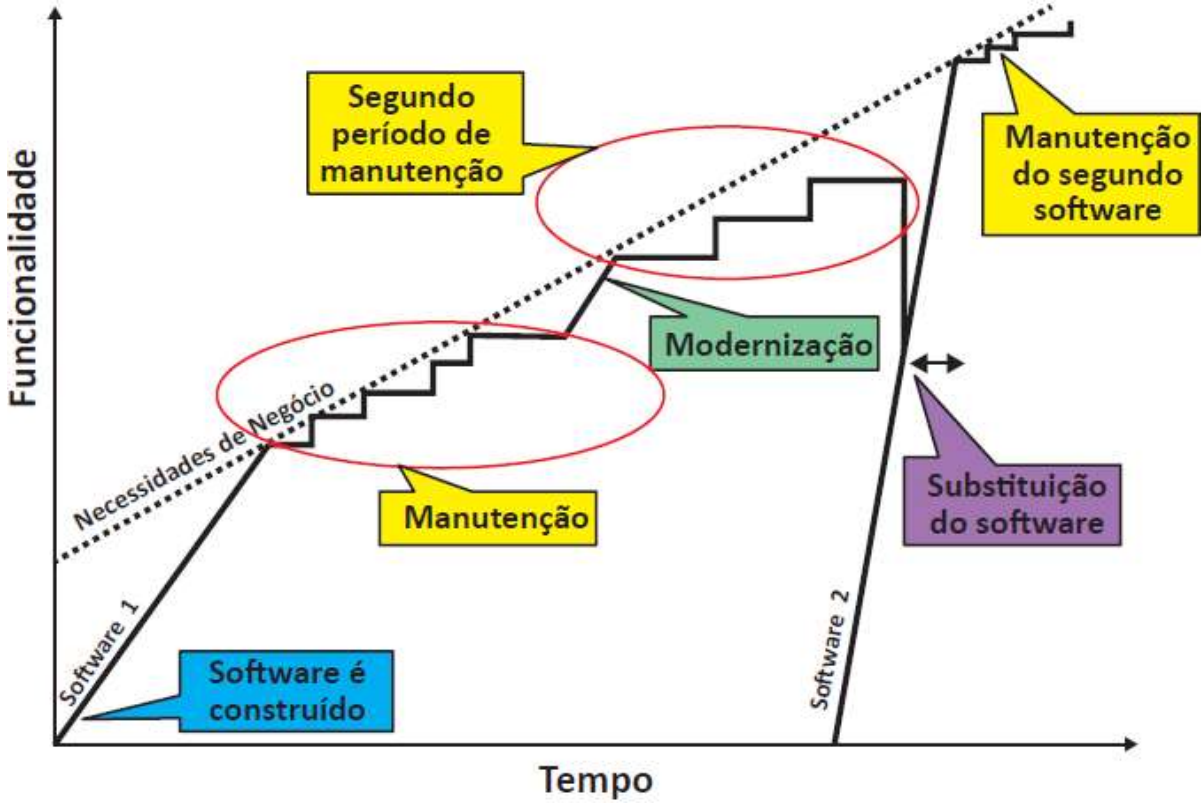
3. Indicações bibliográficas

- 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 12 dez. 2025.

Questão 5

Questão 5.[5]

A evolução de sistemas de software legados pode ser dividida em três categorias: manutenção, modernização e substituição. De acordo com a figura a seguir, conforme o tempo aumenta, a quantidade de funcionalidades também cresce. No primeiro período de evolução a manutenção é realizada, pois as necessidades de negócio são alteradas e o sistema precisa suprir as mudanças por meio da manutenção. No segundo período é realizada a modernização do sistema, pois as necessidades de negócio continuam a crescer, mas há mudanças mais significativas que se fazem necessárias. Além da substituição do sistema de software, nos casos em que o modelo de negócio não atende mais a necessidade da empresa, a manutenção e modernização também não são mais suficientes.



SEACORD, R. C.; PLAKOSH, D.; LEWIS, G. A. *Modernizing Legacy Systems*. Boston: Pearson Education, 2003 (com adaptações).

De acordo com a figura apresentada e considerando um sistema de software implantado de ERP (Enterprise Resource Planning - Planejamento de Recursos Empresariais) contendo um conjunto de módulos que integra todos os departamentos existentes de uma empresa, observou-se a necessidade de criação de um relatório gerencial de comissão da equipe de vendas.

[5]Questão 23 – Enade 2021.

Nesse contexto, é correto afirmar que a manutenção a ser realizada é

A. corretiva.
B. evolutiva.
C. funcional.
D. preventiva.
E. adaptativa.

1. Introdução teórica

Tipos de manutenção de software

Sistemas de software legados são sistemas já implantados, frequentemente utilizados há longo tempo e ainda relevantes para a operação de uma organização. Eles não são necessariamente sistemas inúteis ou obsoletos; ao contrário, muitas vezes continuam sendo críticos para o negócio. O problema é que podem se tornar difíceis de manter, integrar ou evoluir em virtude de tecnologias antigas, documentação insuficiente, dependência de infraestrutura específica, acúmulo de adaptações ou alto custo de alteração.

Nesse contexto, a **manutenção** de um sistema de software legado consiste nos processos de atualização, de correção e de prevenção de problemas em um sistema de software, com o intuito de manter o software operacional no ambiente do usuário. As atividades de manutenção são tipicamente gerenciadas pelo processo de gestão de configuração de software (SCM), que administra as alterações do sistema ao longo de todo o seu ciclo de vida. Existem quatro tipos principais de manutenção de software, elencados a seguir.

- **Corretiva**: este tipo de manutenção é realizado para corrigir erros existentes no software, que não foram identificados ao longo do processo de testes. A correção de bugs, de erros de lógica e de problemas de desempenho são exemplos de manutenções corretivas.
- **Adaptativa**: esta manutenção tem como intuito adaptar o software a novas condições de trabalho, como inseri-lo em um novo ambiente ou fazê-lo funcionar sob novos requisitos de sistema. É exemplo de manutenções adaptativas a tarefa de tornar um software compatível com um novo sistema operacional ou com um novo hardware.

- **Perfectiva (ou evolutiva)**: a manutenção perfectiva tem como foco a adição de novas funcionalidades ao sistema, assim como a melhoria das funcionalidades existentes. Isso inclui a implementação de novos requisitos ou de novas funcionalidades, que não estavam previstas no projeto original.
- **Preventiva**: este tipo de manutenção tem como principal objetivo melhorar a confiabilidade do sistema. Esse tipo de manutenção procura tratar situações que poderão causar erros no software, de maneira a evitar que problemas aconteçam no futuro.

Em média, 50% do tempo dos desenvolvedores costuma ser dedicado à manutenção perfectiva, enquanto as manutenções adaptativa, corretiva e preventiva ocupam, respectivamente, 25%, 21% e 4%. Podemos observar essa distribuição de tempo no gráfico da figura 1.

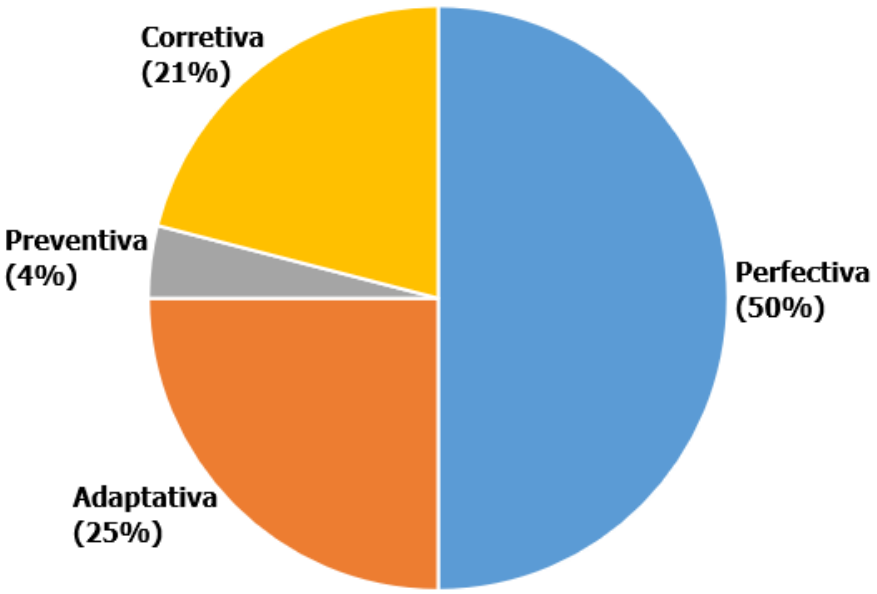


Figura 1. Distribuição do esforço de manutenção.
PRESSMAN e MAXIM, 2021 (com adaptações).

2. Análise das alternativas

A – Alternativa incorreta.

JUSTIFICATIVA. A manutenção corretiva é realizada para corrigir erros existentes no software. Esse não é o caso da situação explicitada no enunciado.

B – Alternativa correta.

JUSTIFICATIVA. A manutenção evolutiva (ou perfectiva) é realizada quando há necessidade de adicionar novas funcionalidades ao sistema de software, ou melhorar as funcionalidades já existentes. No contexto da questão, a criação de um relatório gerencial é considerada como a adição de uma nova funcionalidade.

C – Alternativa incorreta.

JUSTIFICATIVA. O termo "funcional" não é um termo padrão no contexto da manutenção de software.

D – Alternativa incorreta.

JUSTIFICATIVA. A manutenção preventiva é realizada para evitar futuros problemas, não para adicionar funcionalidades.

E – Alternativa incorreta.

JUSTIFICATIVA. A manutenção adaptativa é realizada para adaptar o software a novos ambientes ou a novos requisitos de sistema. Seu foco não é adicionar funcionalidades, mas sim adaptar as funcionalidades existentes a um novo cenário.

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.

ÍNDICE REMISSIVO