NÃO PODE FALTAR

CICLO DE VIDA NAS DIFERENTES ABORDAGENS

Izabelly Soares de Morais

Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

PRATICAR PARA APRENDER

Prezado aluno, estamos susceptíveis a tomar diversas decisões em nosso cotidiano que podem provocar mudanças em nossos planos, e é sabido que algumas podem ter pontos positivos, mas não podemos garantir que sempre será assim, então, acaba sendo uma aposta. No processo de desenvolvimento de software, ocorrem situações bem semelhantes, pois não é considerado algo estável, no sentido de que deve permanecer da mesma maneira que foi implementado.

Esse fator ocorre devido às nossas necessidades de mudança, fazendo, nesse cenário, com que o software tenha que se adequar de alguma forma para suprir essas necessidades. Essas ações devem ser geridas com responsabilidade e com fundamentações que garantam sua qualidade, independentemente do que for realizado, e são postas em ação por meio da manutenção do software, que pode ocorrer tanto em paralelo ao desenvolvimento quanto durante sua implantação no ambiente do cliente.

A fase de implantação, além de demonstrar a necessidade de manutenção, também apresenta algumas atividades essenciais, como prestar o suporte ao usuário, realizar testes no sistema que foi disponibilizado ao usuário e criar uma *release* do software, que se caracteriza pela entrega das funcionalidades do sistema. A depender da metodologia, pode ser em partes ou do sistema como um todo, ou seja, é uma etapa responsável por definir quais são os artefatos que

devem ser gerados, tendo sempre como base a geração de valor ao cliente. E essa escolha pode até ser com base em alguma mudança que precisou ser realizada. Além disso, também é uma ação que pode ocorrer durante o ciclo ou ao seu final.

Falando nisso, é importante compreender como ocorre o ciclo do software e o que acontece após ele ser entregue, já que as etapas de sua concepção podem ser resumidas, de forma genérica, em comunicação, planejamento, modelagem, construção e entrega, podendo apresentar algumas subatividades.

As atividades bancárias se fazem presentes no cotidiano das pessoas durante anos. Portanto, fazem parte de um setor que precisa se atualizar diante do cenário global, tendo em vista que seus clientes possuem perfis diversos. Além disso, no decorrer dos anos, a concorrência também foi um fator determinante na corrida contra o tempo para deixar o sistema sempre atualizado.

Você trabalha no setor de tecnologia de um banco, faz parte do setor de desenvolvimento e uma atividade de manutenção do sistema foi direcionada a você. Nela, você terá de desenvolver diagramas de casos de uso, trazendo melhorias e apresentando à sua equipe os detalhes de como algumas funcionalidades são executadas na atualidade, para que, com isso, consigam desenvolver um planejamento para a manutenção delas, onde poderá ou não haver alguma melhoria no sistema.

Um caso de uso representa como um usuário interage com o sistema, para isso, é possível elaborar o diagrama de caso de uso, em que as interações podem ser representadas utilizando-se o padrão UML. Conforme Pressman *et al.* (2016, p. 875), "um diagrama UML de caso de uso é uma visão geral de todos os casos de uso e de como eles estão relacionados. Fornece uma visão geral da funcionalidade do sistema."

A funcionalidade que deve ser desenvolvida é a de sacar dinheiro da conta corrente. Uma das sugestões é que você utilize a ferramenta "*lucidchart*", mas saiba que a escolha da melhor ferramenta fica a seu critério, tendo em vista que o caso de uso pode ser desenvolvido facilmente em outros tipos de ferramentas.

> O *lucidchart* é um recurso web que disponibiliza diversos recursos para elaboração de artefatos, como diagramas. Além disso, eles proporcionam o compartilhamento e a construção coletiva desses materiais.

Para nortear sua resposta, realize a leitura do Apêndice 1, do livro de Roger Pressman e Bruce R., conforme indicação nas referências.

Conhecer as particularidades de um software é fantástico! Aproveite esses conhecimentos adquiridos para aprofundá-los e aplicá-los em situações práticas de seu cotidiano!

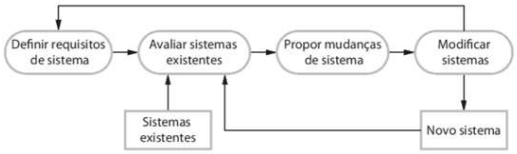
CONCEITO-CHAVE

EVOLUÇÃO DO SOFTWARE

No decorrer do tempo, algumas percepções a questões que envolviam o software se tornaram mais evidentes; os sistemas que foram desenvolvidos inicialmente estavam ficando obsoletos, alguns por desempenharem automatizações que haviam se tornado desnecessárias para o cenário atual e outras que precisavam ser atualizadas para continuar suprindo as necessidades das pessoas, como uma simples calculadora, em que, na atualidade, é possível ter acesso a aplicações que apresentam funcionalidades de calculadora científica, muito utilizada em algumas áreas como engenharia. Outro exemplo é a planilha, que foi criada para automatizar atividades que antes eram desempenhadas no papel, porém, hoje, é possível até realizar importação diretamente de banco de dados e virtualizar as informações, além de ações que envolvem a computação em nuvem.

Note, caro aluno, que tudo acaba passando por um processo evolutivo. E com isso, novos ciclos podem ser executados diante de qualquer tipo de software. E para essa ação, na Figura 4.10, Sommerville (2011) destaca alguns estágios primordiais.

Figura 4.10 | Evolução do sistema



Fonte: Sommerville (2011, p. 29).

Ainda sob o ponto de vista de Sommerville (2011), ao mencionar o ciclo de evolução do sistema, é possível perceber que o desenvolvimento e a manutenção, na realidade, são duas atividades que percorrem, juntas, todo o ciclo do software. Então, por mais que tenha um momento para **definir os requisitos do sistema**, ele não deve, necessariamente, ser executado apenas uma vez; além disso, durante esse processo evolutivo, após essa ação, existe, ainda, a necessidade de se realizar a **busca por sistemas existentes**, em que pode haver, também, uma **avaliação**. Essas etapas são importantes, pois é uma oportunidade de se verificar o que já existe no mercado e, simultaneamente, descobrir novidades tecnológicas ou novas formas de solucionar certas questões. Após algumas análises, **mudanças são propostas**, sejam de funcionalidades já existentes ou para a indicação da necessidade de criação de novas, e após a **modificação do sistema**. Além disso, a depender da mudança, o sistema pode assumir um objetivo totalmente diferente de quando foi criado inicialmente.

MANUTENÇÃO DO SOFTWARE

A manutenção também pode ser vista como uma etapa que ocorre após a entrega do sistema ao cliente, ou seja, após a implantação. Conforme Sommerville (2011), existem três diferentes tipos de manutenção de software:

- **Correção de defeitos**: nesse sentido, o autor apresenta duas perspectivas, em que menciona que ações de correção na codificação são menos onerosos do que erros de projeto, ou seja, a fase de planejamento é extremamente importante, pois pode ocasionar necessidades de mudanças após a finalização

do sistema, o que sai muito mais caro, pois imagine que você terá de refazer todas as etapas.

- **Adaptação ambiental**: ocorre quando as funcionalidades precisam se adaptar às mudanças presentes no atual contexto em que o software precisa ser utilizado. Nesse cenário, mudanças não funcionais também podem acarretar esse tipo de manutenção, como alteração nos componentes físicos do sistema, mudanças de sistema operacional, entre outros.
- **Adição de funcionalidade**: nesse tipo de manutenção, os requisitos do sistema sofrem alterações e devem atender às novas solicitações da organização. Geralmente, ocorre com mais frequência.

A manutenção também pode receber termos adicionais, como "**corretiva**" ou "**adaptativa**", em que, respectivamente, um é utilizado para situações em que o sistema precisa da correção de algum problema e o outro é para adaptação, que pode se enquadrar a um novo ambiente ou a requisitos. Por ser uma ação que, com o passar dos anos, tornou-se frequente em outras etapas, diferentes apenas da implantação, existem algumas estimativas sobre a distribuição dos esforços e custos durante o desenvolvimento de um sistema. Além disso, mencionamos que o desenvolvimento tem como companhia, durante sua execução, a manutenção, porque também pode haver a chamada "**preventiva**", que ocorre durante o processo, trazendo melhorias ao sistema e garantindo que ele esteja atendendo às necessidades do cliente.

RESPONSABILIDADES DA EQUIPE

Independentemente do tipo de manutenção, existem algumas **responsabilidades** imposta aos membros da equipe diante das possíveis ações que podem ser tomadas para realização de tal ação. Entre elas, deve haver uma estabilidade da equipe, levando-se em consideração que os envolvidos naquele processo acabam se tornando mais capacitados do que novos membros ou outras equipes, então, é indicado que a manutenção seja realizada pela mesma equipe que concebeu o sistema. Para tal feito, também se faz necessário trazer melhorias para a equipe, no sentido de promover qualificações a respeito das habilidades já existentes e às que podem se tornar necessárias no futuro, pois o sistema está em constante evolução.

> Além de responsabilidades impostas às pessoas, existem também as que englobam o âmbito legal; ou seja, toda manutenção, assim como ações referentes ao sistema em si, deve ser documentada. No caso das manutenções, o indicado é que haja um contrato, selando os acordos realizados para as mudanças que serão feitas.

Conforme Paula Filho (2019, p. 286), vários papéis da organização se destacam nesses processos de mudança em um software, cada um trazendo uma **responsabilidade** compatível com suas habilidades:

- "Gerentes de produto, responsáveis pelas atividades referentes a um produto ou a um grupo de produtos;
- Representantes dos usuários, que acionam as solicitações de manutenção;
- Proprietários dos itens, que realizam as modificações necessárias nos itens afetados;
- Proprietários dos produtos, que tomam decisões eventualmente necessárias sobre cada produto;
- Comissões de controle de configurações, que tomam as decisões importantes relativas à gestão de configurações, em nível de produto ou de grupo de produtos correlatos;
- Administrador de configurações, que centraliza o funcionamento dos mecanismos dessa disciplina, também em relação às atividades de manutenção."

IDENTIFICAÇÃO DAS VERSÕES DO PRODUTO

Como mencionamos anteriormente, as solicitações devem ser documentadas, e o processo que será executado deve ser muito bem definido, evitando que ações fora do escopo sejam executadas. Além disso, é importante que haja a identificação das versões do produto. Um exemplo que pode ser mencionado é o sistema operacional Android, desenvolvido para smarthphones. Caso você realize uma busca no site, é possível acompanhar todas as mudanças ocorridas em suas versões. O histórico das mudanças que foram implementadas também apresenta informações importantes.

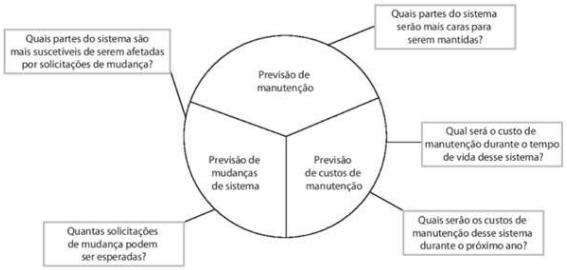
O que acontece é que, após o sistema ser colocado em uso, é possível obter algumas informações importantes para ratificar-se de que o processo utilizado para seu desenvolvimento foi bem executado. Por meio dos primeiros relatos, algumas necessidades de mudanças podem ser documentadas.

> Para isso, existe o que é chamado de solicitação de modificação, e esse registro deve conter uma descrição do problema que fez com que a necessidade da mudança fosse retratada, dados para que a situação possa ser replicada e avaliada e estimativas de esforços para que a mudança seja realizada.

MELHORIAS REALIZADAS PELA MANUTENÇÃO

Trazendo uma percepção de que a manutenção pode agregar mais valor ao sistema, ela pode também sinalizar as melhorias que podem ser realizadas. Então, é possível mensurar essas ações, como manutenção e seus respectivos tipos (preventiva, corretiva, adaptativa) e melhorias de pequeno ou grande porte. Abaixo, a Figura 4.11 traz alguns aspectos sobre a previsão de manutenção.

Figura 4.11 | Previsão de manutenção



Fonte: Sommerville (2011, p.173).

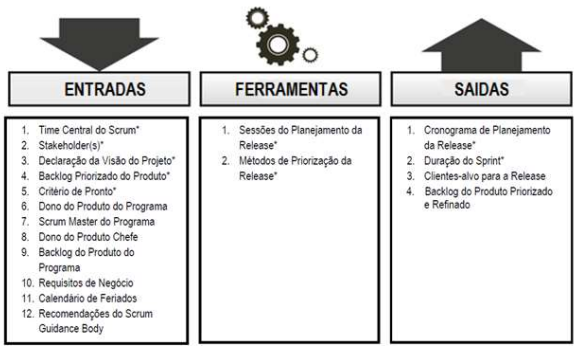
A Figura 4.11 traz aspectos relacionados à previsão de manutenção, então, alguns questionamentos podem ser realizados, e conforme as respostas são fornecidas, essa necessidade pode ser apresentada. Então, os questionamentos a serem realizados podem abranger os seguintes pontos:

1. **Custo de manutenção** diferente para cada parte do sistema, ou seja, é mais barato ir realizando testes durante o desenvolvimento, para se verificar as necessidades de mudanças, ou é melhor esperar a finalização do sistema?
2. Durante o tempo de vida do sistema, quais **mudanças** deverão ocorrer e qual custo de manutenção será obtido?
3. Existe algum planejamento para **gestão das mudanças**? Elas serão solicitadas conforme a necessidade do usuário ou a equipe fará testes constantes para verificar essas necessidades? Existem partes mais sensíveis à necessidade de mudanças? Por exemplo: um sistema hospitalar, devido ao seu alto grau de complexidade, por ser, na maioria das vezes, sistemas de tempo real, demandam testes e mudanças a todo momento, então, devem ser estimados

PLANEJAMENTO DE RELEASES

Aproveitando o exemplo do sistema hospitalar, é possível associá-lo à importância do planejamento das *releases*, ou seja, quais são as funcionalidades que serão entregues? Serão entregues a cada iteração ou apenas no final do ciclo? A depender da metodologia, as duas situações podem ocorrer. Conforme o PMI (2017), existem algumas métricas para conduzir o planejamento da *release*, que envolve entradas, ferramenta e saídas, como descrito na Figura 4.12.

Figura 4.12 | Planejamento da *release*



Fonte: PMI (2017, p. 173).

A Figura 4.12 demonstra o que pode compor cada particularidade do **planejamento de uma release**, então, como entrada, é possível notar uma lista de ações voltadas, nesse caso, ao uso da metodologia ágil *Scrum*, que aplica os principais conceitos do manifesto ágil, que são:

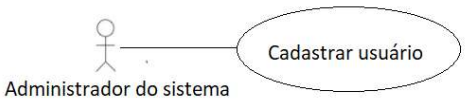
- Indivíduos e interações mais que processos e ferramentas.
- Colaboração com o cliente mais que negociação de contratos.
- Software em funcionamento mais que documentação abrangente.
- Responder a mudanças mais que seguir um plano.

Conforme o PMI (2017), entre as ferramentas utilizadas para o planejamento da release, as Sessões de Planejamento da Release tem como objetivo, no *Scrum*, desenvolver um cronograma de plano da release, trazendo transparência para que o Time *Scrum*, que é formado pelos desenvolvedores, designers, especificadores, entre outros, possa ter uma visão geral de todo o projeto e passem a priorizar as principais entregas, ou seja os requisitos mais importantes. E como artefatos de saída, espera-se que o cronograma de planejamento tenha sido concluído e que traga informações, por exemplo, da duração da *sprint*, ou seja, da iteração. Além disso, é possível visualizar na figura que os clientes interessados nas atividades da *release* e o *backlog* priorizado e refinado também são tidos como resultados finais. É importante salientar que esses cronogramas e o planejamento da release podem ser atualizados, ou seja, os passos e as definições podem passar por mudanças.

EXEMPLIFICANDO

Conforme Pressman (2016), existem os modelos de requisitos que têm como base demonstrações dos **requisitos dos clientes**, então, podem apresentar domínio da informação, funcional e comportamental do sistema. Alguns artefatos podem ser gerados da manutenção do sistema, como o caso de uso, que é retratado por meio da UML (*Unified Modeling Language*), que apresenta uma linguagem padrão para modelagem de sistemas e disponibiliza muitos elementos e diversos tipos de diagramas; entre eles, podemos citar o **diagrama de caso de uso**, como mostra a Figura 4.13, em que uma funcionalidade do sistema, que, no caso, é "cadastrar usuário", é visualmente retratada pelo ator e pelo caso de uso. Nesse caso, o ator é a representação de quem vai executar tal ação, podendo, até mesmo, ser o próprio sistema; já o caso de uso é a ação, que é representada, conforme os padrões UML, pela figura geométrica elipse.

Figura 4.13 | Exemplo de Caso de Uso.



Fonte: elaborada pela autora.

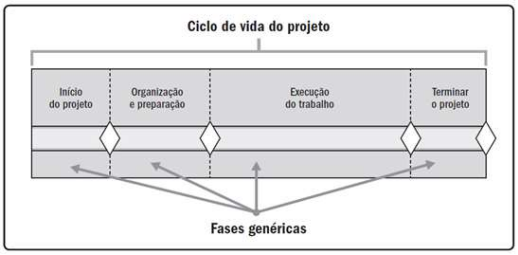
Uma ferramenta que pode ser utilizada para o desenvolvimento de caso de uso é a *lucidchart*, que possui algumas funcionalidades gratuitas e é online.

Além disso, há os **modelos de análise e projetos**, que trazem informações técnicas referentes à arquitetura, interface, entre outros. A UML também abrange esse tipo de modelagem.

CICLO DE VIDA DE UM PROJETO

O **ciclo de vida de um projeto** pode ser originado a partir de alguns passos genéricos, entre eles, o início do projeto, a organização, a preparação, a execução do trabalho e o término do projeto, conforme a Figura 4.14.

Figura 4.14 | Ciclo de vida do projeto



Fonte: PMI (2017, p. 548).

O **início do projeto** envolve a etapa de comunicação, que é primordial para dar início ao projeto de software. Ela é responsável por amenizar as complexidades apresentadas pela demanda do cliente, ou seja, por meio da comunicação, é possível deixar as informações mais relevantes acessíveis a todos os interessados no projeto, e é durante esse processo que, geralmente, os requisitos do software são obtidos, pois acabam definindo as metas e os objetivos do sistema de forma geral.

A **organização e a preparação** estão relacionadas ao planejamento, que é um momento dedicado à análise do que foi obtido durante a comunicação. Então, são definidas estratégias para alcançar os objetivos antes traçados, já que todo o escopo do projeto deve ser compreendido. Além disso, todos os envolvidos devem estar cientes das prioridades do projeto e da modelagem, que proporciona a apresentação das particularidades do sistema em diferentes níveis de abstração, podendo apresentar informações com linguagens simples e comuns ao nosso cotidiano, mas também podem ser utilizados termos mais técnicos.

A **execução do trabalho** é o momento que envolve a implementação, uma vez que as etapas anteriores já aprimoraram as ideias do que deve ser desenvolvido. Dessa forma, além da codificação na linguagem de programação escolhida, existem, também, as atividades de testes.

ASSIMILE

O diferencial de uma metodologia ágil e uma metodologia tradicional ou sequencial, como também é conhecida, é que a metodologia ágil traz ideias do manifesto ágil, mudando um pouco algumas abordagens levantadas ao longo dos anos pelas metodologias mais tradicionais. Como exemplo, é possível mencionar a execução de uma etapa associada à finalização da etapa anterior, o que não necessariamente ocorre em uma metodologia

ágil. Como exemplo, podemos mencionar o *Scrum*, Xp, e como metodologia tradicional, temos a modelo cascata, evolucionário, iterativo e incremental, entre outros.

Então, é possível observar que existe um ciclo considerado genérico, mas que nada impede de que haja adaptações de suas etapas para suprir as necessidades dos envolvidos no projeto. Inclusive, **após a entrega** do sistema, existem etapas que ainda fazem parte do ciclo de vida do projeto, tais como a manutenção, o treinamento da equipe que utilizará o sistema (cliente e usuários), as melhorias no sistema que foi entregue, entre outros.

REFLITA

Na metodologia ágil *Scrum*, por exemplo, é possível visualizar a aplicação da mudança a partir do momento em que os artefatos gerados nas *sprints*, ou seja, nas iterações ou ciclos, são ou não aprovados conforme os critérios de avaliação do produto do cliente. Então, caso não seja, haverá mudança, a fim de que, ao passar novamente pelo ciclo, seja aprovado.

Porém, apesar de trazer um aspecto de liberdade e de ser totalmente aberto às mudanças, todas devem passar pelo gerenciamento, tendo em vista que se faz necessária, para um bom fluxo de desenvolvimento, a avaliação do que realmente faz ou não sentido para o projeto.

Então, o desafio é: como lidar e como definir o que deve ou não ser inserido nesse processo? Será que todas as funcionalidades realmente fazem sentido para o projeto da forma como foram definidas durante o processo de planejamento inicial?

O software é um artefato inconstante, que necessita de etapas bem definidas para ser desenvolvido, mas que, em relação ao seu uso, precisa fazer parte de um sistema de iterações e interações diversas, por isso, é importante conhecer um pouco sua manutenção, as responsabilidades envolta dessa etapa, o planejamento de releases e o ciclo de vida do projeto, tanto na perspectiva inicial, quanto após a entrega do sistema pronto.

FAÇA A VALER A PENA

Questão 1

No ciclo de vida de um software são executadas diversas etapas, então, inicialmente, deve haver um planejamento, fundamentando as demais ações e os passos que deverão ser executados posteriormente. Os custos de investimento durante esse ciclo variam conforme a etapa; dessa forma, caso seja detectado algum erro no sistema e uma mudança tenha quer ser realizada, os custos serão variados.

A respeito da etapa mais onerosa para detecção de erros, assinale a alternativa correta.

a. A etapa mais onerosa é a de manutenção.

b. A etapa mais onerosa é a de planejamento.

Ver anotações 0

c. A etapa mais onerosa é a de implementação.

d. A etapa mais onerosa é a de modelagem.

e. A etapa mais onerosa é a de comunicação.

Questão 2

O desenvolvimento de um software se torna complexo, porém, não é uma complexidade pautada pela dificuldade e sim pela quantidade de funcionalidades, tecnologias, dispositivos que ele deve atender, até porque o seu uso só faz sentido se ele conseguir desempenhar sua função da maneira correta. Dessa forma, a existência de um ciclo de vida do projeto é muito importante para nortear todas essas atividades.

Considerando as informações apresentadas, analise as afirmativas a seguir:

I. A comunicação é uma etapa executada no pós-entrega, quando o cliente recebe o sistema em seu ambiente.

II. A organização e a preparação estão relacionadas ao planejamento e à modelagem do sistema que precisa ser desenvolvido.

III. A implementação envolve não só a codificação do sistema, tendo como base as ideias aprimoradas anteriormente, mas também os testes.

IV. O treinamento dos usuários é uma ação que caracteriza o início do ciclo de vida do projeto de software.

Considerando o contexto apresentado, é correto o que se afirma em:

a. I e II, apenas.

b. II e III, apenas.

c. III e IV, apenas.

d. II, III e IV, apenas.

e. I, II, III e IV.

Questão 3

> [...] organizações sempre almejam entregar produtos satisfatórios para clientes e usuários (alta qualidade), dentro de custos e prazos compensadores (alta produtividade) e previsíveis (alta previsibilidade). Para atingir esses objetivos, as organizações têm que entender seus próprios processos de negócio, a fim de aperfeiçoá-los e obter resultados cada vez melhores. No caso de organizações produtoras de software, os processos de negócio são os processos relativos ao ciclo de vida do software.
> — (PAULA FILHO, 2019, p. 4)

De acordo com as informações apresentadas no quadro a seguir, faça a associação dos feitos contidos na Coluna A com seus respectivos autores na Coluna B.

COLUNA A	COLUNA B

I. Neste tipo de manutenção, os requisitos do sistema sofrem alterações e devem atender às novas solicitações da organização. Geralmente, ocorre com mais frequência.	1. Correção de defeitos.
II. Nesse sentido, o autor apresenta duas perspectivas em que menciona que ações de correção na codificação são menos onerosas do que erros de projeto, ou seja, a fase de planejamento é extremamente importante, pois pode ocasionar necessidades de mudanças após a finalização do sistema, saindo muito mais caro, uma vez que todas as etapas terão de ser refeitas.	2. Adaptação ambiental.
III. Ocorre quando as funcionalidades precisam se adaptar às mudanças presentes no atual contexto em que o software precisa ser utilizado. Nesse cenário, mudanças não funcionais também podem acarretar esse tipo de manutenção, como alteração nos componentes físicos do sistema, mudanças de sistema operacional, entre outros.	3. Adição de funcionalidade.

Ver anotações

Assinale a alternativa que apresenta a associação correta entre colunas.

a. I-1; II-3; III-2.

b. I-1; II-2; III-3.

c. I-2; II-1; III-3.

d. I-2; II-3; III-1.

e. I-3; II-1; III-2.

REFERÊNCIAS

PAULA FILHO, W. de P. **Engenharia de software**: projetos e processos. 4. ed. Rio de Janeiro: LTC, 2019.

PMI — Project Management Institut. **Guia do conhecimento em gerenciamento de projetos**: guia PMBOK. 6. ed. [S.l.]: PMI, 2017.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software**: uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.