

ENGENHARIA DE SOFTWARE

Roque Maitino Neto

Ver anotações

Deseja ouvir este material?

Áudio disponível no material digital.

CONHECENDO A DISCIPLINA

Caro aluno, seja bem-vindo à disciplina de Engenharia de Software.

A divisão de um trabalho em etapas e o uso de procedimentos já bem estabelecidos em sua execução são ações implementadas na maioria dos ramos da atividade humana. Fica difícil conceber, por exemplo, a criação de um novo avião ou a construção de um prédio sem um conjunto já devidamente experimentado de passos que forneçam suporte a quem deverá construí-los. Mesmo não tendo natureza rigorosamente prescritiva, esses procedimentos indicam caminhos, estabelecem padrões e – por que não – previnem problemas comuns durante a criação do artefato ou a prestação do serviço.

Embora prédios e aviões tivessem começado a ser construídos muito antes dos programas de computador, foi com a Computação que a divisão de uma grande tarefa em etapas ganhou notável relevância e tornou-se imprescindível para a entrega de produtos de qualidade e adequados ao seu propósito. De forma simplista, foi da necessidade de se estabelecer um conjunto de métodos eficientes para criação de

sistemas mais confiáveis que nasceu o que hoje conhecemos como Engenharia de Software, um dos mais importantes ramos da nossa área. E foi pensando nas habilidades necessárias ao seu desenvolvimento nesta disciplina que esta obra procurou reunir temas que se estendem desde os fundamentos da Engenharia de Software até aspectos de manutenção e evolução de sistemas, passando pelo tratamento de elementos de qualidade de software e pela identificação de teste e seus tipos.

A fim de que você conheça, comprehenda e seja capaz de colocar em prática todo o conteúdo aqui desenvolvido, a divisão dos assuntos foi assim concebida: na Unidade 1, são abordados os fundamentos da Engenharia de Software, incluindo a visão da metodologia tradicional e das metodologias ágeis de desenvolvimento. Na Unidade 2, o foco se volta a aspectos de qualidade envolvidos em uma iniciativa de desenvolvimento de um software, com a clara separação entre qualidade do produto e qualidade do processo. Depois, na Unidade 3, são investigados os testes de software, com ênfase para as definições relacionadas ao tema, para os tipos de testes e para os testes automatizados. Na Unidade 4, esta obra é concluída com o tratamento de auditoria de sistemas e manutenção e evolução de software.

Dominar conceitos e práticas da Engenharia de Software é, portanto, o que define um verdadeiro profissional de Tecnologia da Informação. Aliado à habilidade em conduzir projetos, o domínio de uma metodologia capacita o profissional a gerenciar empreitadas de criação de ferramentas de software, além de habilitá-lo a exercer vários papéis durante o processo de desenvolvimento. Preparado para o contato com o conhecimento que mudará sua maneira de enxergar a TI?

Então siga nossas orientações de leitura, de elaboração de atividades e de estudo e faça bom proveito deste tema fascinante.

Bons estudos!

NÃO PODE FALTAR

INTRODUÇÃO À ENGENHARIA DE SOFTWARE

Roque Maitino Neto

Ver anotações

O QUE É A ENGENHARIA DE SOFTWARE?

A Engenharia de Software abrange um processo, um conjunto de métodos (práticas) e um leque de ferramentas que possibilitam aos profissionais desenvolverem software de altíssima qualidade (PRESSMAN; MAXIM, 2016).



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Aqui estamos na primeira unidade do nosso livro. Como não poderia deixar de ser, o caminho que tomaremos para o início de nossos estudos será o da abordagem introdutória da disciplina e, nesse sentido, trataremos, na primeira seção, os aspectos gerais da Engenharia de Software, seus princípios e o ciclo de vida pelo qual passa um produto até que ele esteja pronto para ser comercialmente viável. Depois, na segunda seção da unidade, teremos oportunidade de conhecer metodologias mais atuais e mais ágeis de desenvolvimento, com ênfase para o Extreme Programming, o Scrum e algumas ferramentas que tornam viáveis sua aplicação. Por fim, será pelo conteúdo desenvolvido na terceira seção que entenderemos os meios usados por uma equipe de desenvolvimento para compartilhar e controlar versões de um produto que conjuntamente desenvolvem.

Com o bom aproveitamento deste conteúdo, você conhecerá os fundamentos da Engenharia de Software e o modelo de desenvolvimento que, de uma forma ou de outra, inspirou e serviu como base para os demais modelos que vieram depois dele. Mais uma importante contribuição desta unidade em seu desenvolvimento será dada pelo conteúdo relacionado às metodologias ágeis, bem mais adequadas às demandas atuais de desenvolvimento e preparadas para atribuir ao cliente sua verdadeira importância no processo de criação de um produto que, afinal de contas, será direcionado às suas próprias necessidades.

Assim, a capacidade de bem aplicar os fundamentos de Engenharia de Software, o domínio das metodologias ágeis e a correta utilização de mecanismos de controle de versões de um produto são os resultados que esta unidade pretende ajudá-lo a atingir. Com sua dedicação aos estudos e o adequado aproveitamento do material didático colocado à sua disposição, esses objetivos serão atingidos, sem dúvida. Bom trabalho e sigamos adiante!

Caro aluno, iniciamos esta etapa com um necessário registro temporal: a Engenharia de Software nem sempre foi como a conhecemos hoje, tendo sido necessário considerável esforço para que os profissionais com ela envolvidos a estruturassem como um dos mais importantes ramos da Computação. Seus métodos precisaram passar pelo polimento do tempo para se tornarem bem formatados e úteis para a criação de artefatos de software de qualidade e duradouros. Mesmo com o elevado grau de transformação pelo qual vem passando essa disciplina, algumas práticas permanecem em uso apesar das adaptações e melhorias que vêm acontecendo. Um bom exemplo é a metodologia tradicional de desenvolvimento, criada nos primeiros anos de existência da Engenharia de Software e que ainda baseia muitos procedimentos de software. Como não poderia deixar de ser, a metodologia tradicional será abordada nesta primeira seção, juntamente com os princípios, desde sempre, norteadores da Engenharia de Software.

Você foi contratado por uma pequena empresa de desenvolvimento para atuar como engenheiro de software júnior e, logo nas primeiras semanas de trabalho, deparou-se com um cenário em que não havia um procedimento definido de criação ou manutenção dos produtos de software. Cada componente de dado projeto mantinha uma forma particular de desenvolvimento e de registro das alterações feitas nos produtos e, embora trabalhassem em equipe, não havia papéis definidos entre os integrantes de cada empreitada. Como era de se esperar, esse cenário de ausência de procedimentos e de papéis definidos reprimia o crescimento da empresa, fato que vinha causando frustração generalizada na organização.

Você deverá propor soluções que melhorem a dinâmica e o clima da empresa, incluindo o estabelecimento de uma metodologia de trabalho (executado em duas etapas) e um procedimento de versionamento dos produtos que cada equipe estava construindo.

Considerando a urgência dessas providências e a pouca maturidade do pessoal em atuar em conformidade com modelos e padrões, você optou por implementar, como sua primeira ação, o modelo tradicional de desenvolvimento, conhecido também como ciclo de vida clássico ou modelo em cascata. Sendo assim, ficou estabelecido que a execução do próximo projeto assumido pela empresa seria de acordo com esse modelo e, para que isso fosse possível, você deveria explicar à equipe cada etapa do modelo em cascata.

Seu desafio é o de preparar uma apresentação (em PowerPoint ou outra ferramenta similar), com no mínimo oito lâminas, contendo a visão geral de cada etapa deste modelo, incluindo os objetivos de cada um deles. Considere que essa apresentação é direcionada à equipe de profissionais que comporá o próximo projeto da organização e que seu conteúdo deve ser bastante didático e objetivo.

Mãos à obra.

DICA

Compreender bem a parte introdutória de uma disciplina é a chave para o bom aproveitamento do restante do seu conteúdo. Essa boa compreensão, no entanto, só é atingida com a leitura atenta dos textos propostos e com a devida entrega na execução das atividades. Naturalmente que a ajuda de colegas e do professor também será muito importante.

Coloque todo seu foco nesta primeira seção e bom trabalho!

CONCEITO-CHAVE

A Engenharia de Software é uma daquelas disciplinas que nos surpreendem o tempo todo. Dinâmica por natureza, ela vem agregando elementos ao seu rol de temas, vem refinando e agilizando seus processos e, com efeito, contribuindo de forma inestimável para o

desenvolvimento da Computação, entendida aqui como uma ciência. Encontrar um conceito definitivo para Engenharia de Software é, portanto, uma tarefa cujo resultado pode se apresentar ligeiramente obsoleto em um momento seguinte. Essa sua inclinação à constante mutação, no entanto, não a transforma em uma disciplina incompreensível e sem um nexo conceitual definido.

o

Ver anotações

| DEFINIÇÃO DE ENGENHARIA DE SOFTWARE

A literatura nos oferece diversas definições para a Engenharia de Software, desde as sucintas até as mais elaboradas. Porém, a grande maioria utiliza-se de termos como procedimentos, métodos e ferramentas para indicar uma natural convergência: **a qualidade do produto a ser desenvolvido**. Se o resultado da aplicação de procedimentos, métodos e ferramentas não for uma entrega de qualidade, de pouco terá valido todo o esforço em adotá-los. Como primeiro passo, convém conhecermos a definição para Engenharia de Software dada por um importante autor dessa área.

A Engenharia de Software abrange um processo, um conjunto de métodos (práticas) e um leque de ferramentas que possibilitam aos profissionais desenvolverem software de altíssima qualidade. (PRESSMAN; MAXIM, 2016).

Ao analisarmos uma definição, devemos considerar a natureza absolutamente dinâmica da Engenharia de Software, conforme já mencionamos. Com o passar dos anos, a importância que um programa de computador assumiu na vida das pessoas e em seus negócios foi sensivelmente alterada e a necessidade por qualidade e agilidade nas entregas cresceu na mesma proporção, o que acabou justificando o surgimento de novos meios de desenvolvimento de software. Dominar

conceitos da Engenharia de Software é um caminho seguro para o estabelecimento de práticas consoantes com as necessidades da organização em que o profissional de TI está inserido.

Via de regra, aquele que domina um conceito é capaz de adaptá-lo às suas necessidades ou as da organização em que atua, sem o risco de esvaziar sua essência. Por isso, uma melhor definição do conceito de Engenharia de Software pede o detalhamento dos termos que o compõem. O termo “Engenharia” está relacionado ao projeto e à manufatura de um artefato e, desde já, ressaltamos a importância dos requisitos e das especificações do artefato nesse contexto. Por não possuir uma forma física, um programa de computador não passa por processo de manufatura tradicional, conforme o conhecemos no meio industrial. Além disso, a produção de programas de computador possui situações bastante particulares, que requerem soluções especializadas e, portanto, diferentes daquelas adotadas em um processo fabril.

Avançando para o próximo termo, temos uma **definição satisfatória para software** como:

1. **Instruções** que, quando executadas, produzem a função desejada.
2. **Estruturas de dados** que possibilitam os programas manipularem a informação;
3. E **documentação** relativa ao sistema.

Esses três elementos são, de fato, a matéria-prima para a **Engenharia de Software**, que é definida pelo IEEE Computer Society **como a aplicação de uma abordagem sistemática, disciplinada e quantificável de desenvolvimento, operação e manutenção do software, além do estudo dessas abordagens** (IEEE, 2004).

PRINCÍPIOS DA ENGENHARIA DE SOFTWARE

Formada a base conceitual da disciplina, avançamos agora para elementos que costumamos chamar de princípios da Engenharia de Software justamente por darem uma feição própria a ela e por suportarem suas práticas. São esses princípios que ajudam a estruturar os processos e a padronizar as soluções propostas por esse ramo da Computação. Eles foram propostos pelo SWEBOK (*Software Engineering Body of Knowledge* ou Conjunto de Conhecimentos de Engenharia de Software), que se trata de uma importante publicação do IEEE na área.

O primeiro princípio proposto pelo IEEE (2004) foi o da **organização hierárquica**, o qual estabelece que os componentes de uma proposta de solução ou a organização de elementos de um problema devem ser apresentados em formato hierárquico, em uma estrutura do tipo árvore, com quantidade crescente de detalhamento nível após nível.

A **formalidade** pode ser apontada como o próximo princípio e ela estabelece que a resolução de um problema deve seguir uma abordagem rigorosa, metódica e padronizada. Depois, como terceiro princípio, vale mencionar a **completeza**, que estabelece a necessidade de que se verifique cuidadosamente se todos os elementos de um problema foram levantados e se a solução proposta os contempla por completo.

O IEEE (2004) sugere como próximo princípio aquele conhecido como **dividir para conquistar** e a justificativa para sua criação é simples: fica intelectualmente inviável encarar um problema complexo (e a construção de um sistema é um problema complexo!) sem que ele seja dividido em partes menores, independentes e, portanto, mais facilmente gerenciáveis. Essa é a lógica subjacente à criação de soluções modulares.

O princípio da **ocultação** estabelece que, ao conceber uma solução modular, cada módulo tenha acesso às informações necessárias para seu funcionamento, o que significa ocultar informações não essenciais.

Já o princípio da **localização** aponta para a necessidade de se colocar juntos os itens relacionados logicamente em um sistema e, pelo princípio da **integridade conceitual**, o engenheiro de software deve seguir uma filosofia e uma arquitetura de projeto coerentes (IEEE, 2004).

A publicação SWEBOK (IEEE, 2004) ainda aborda alguns outros princípios, incluindo a **abstração**, a qual determina a atenção do engenheiro de software a elementos com afinidade a um problema particular, isolando-os de outros elementos relacionados a outros tipos de problemas. Difícil de entender? Então pense na abstração como a aplicação prática da capacidade de um profissional em concentrar-se em aspectos essenciais para a resolução de um determinado problema, deixando para outro momento os problemas relacionados ou de importância secundária.

o

Ver anotações

EXEMPLIFICANDO

Certa equipe de desenvolvimento construiu um processador de texto. Ao planejar o menu da ferramenta, a funcionalidade de classificação por ordem alfabética acabou ficando no menu "Layout de página". Tempos depois, a mesma equipe foi chamada com o intuito de construir um sistema acadêmico. Para a fase de análise, a equipe escolheu a metodologia da análise estruturada e, para o projeto, a metodologia de projeto orientado a objeto.

A Engenharia de Software guia-se por princípios que devem ser respeitados, a fim de que sua prática leve ao cumprimento de seus objetivos. No caso apresentado, dois desses princípios foram ignorados pela equipe de desenvolvimento.

Ao pensar no menu, a equipe não considerou que a localização dos elementos no programa final é de suma importância para que o usuário o utilize com desenvoltura.

Quando um menu de programa é identificado como “Layout de página”, não se espera que a funcionalidade de classificação por ordem alfabética lá esteja localizada.

No segundo projeto, ao não considerar a adoção de uma metodologia do início ao fim do desenvolvimento, a equipe simplesmente ignorou o fato de que o projeto deve respeitar o princípio da integridade conceitual.

O SOFTWARE

Agora que você já teve contato os pilares da Engenharia de Software, vale focarmos naquele que é seu objeto principal: o software.

De acordo com Pressmann e Maxim (2016), software de computador é o produto que profissionais de software desenvolvem e ao qual dão suporte no longo prazo. Abrange programas executáveis em um computador conteúdos e informações descritivas.

Os conteúdos aos quais os autores se referem são dados e informações geradas conforme a execução do programa acontece. Já as informações descritivas referem-se à documentação do programa, necessária para referência e manutenção futura. Por meio dessa definição, é possível entender que o software não se resume ao programa executável.

Um dos elementos que contribui para tornar o software uma criação de natureza única é o fato de que ele é um produto e, ao mesmo tempo, o veículo usado para a distribuição desse produto. Embora o papel do software tenha passado por mudanças significativas a partir da metade final do século XX, ele se consolidou como um elemento indispensável em nossos dias e se firmou como o responsável por fazer transitar mundo afora o bem mais precioso que possuímos, que é a informação (PRESSMAN, MAXIM, 2016).

Software consiste em: (1) instruções (programas de computador) que, quando executados, fornecem características, funções e desempenho desejados; (2) estruturas de dados que possibilitam aos programas manipular informações adequadamente; (3) informação descritiva, tanto na forma impressa quanto na virtual, descrevendo a operação e o uso do programa (PRESSMAN, MAXIM, 2016).

0

Ver anotações

Quando colocamos o conceito de software em palavras simples e objetivas e o abordamos como um produto comercialmente viável, a compreensão de suas funções pode soar bastante simples. Esse fato tem o potencial de nos dar a falsa impressão de que criá-los também seja uma atividade sem complexidade e que “sentar e programar” seja uma solução sempre viável. Mas, em sentido contrário, desenvolver produtos de software tem deixado de ser uma atividade artesanal e empírica e tem se tornado sistemática, organizada e baseada em métodos, muito por conta da evolução da Engenharia de Software. No entanto, logo em seus primeiros anos, a produção de software enfrentou tempos turbulentos, com tarefas em que a incerteza era fator preponderante e os resultados dos esforços de criação de um artefato poderiam ser empreendidos em vão.

De acordo com Schach (2009), na década de 1960, alguns atores do processo de desenvolvimento de software cunharam a expressão “crise do software” na intenção de evidenciar o momento adverso que a atividade atravessava. Em seu sentido literal, crise indica estado de incerteza ou declínio e, de fato, esse era o retrato de um setor inapto a atender a demanda crescente por produção de software, a qual entregava programas que não funcionavam corretamente, construídos por meio de processos falhos e que não podiam passar por manutenção

facilmente. Além disso, a incerteza causada pela imprecisão nas estimativas de custo e de prazo afetava a confiança das equipes e principalmente dos clientes.

Era fato comum naquela época a precariedade na comunicação entre cliente e equipe de desenvolvimento, e essa realidade contribuía para que a qualidade do levantamento dos requisitos fosse perigosamente baixa, acarretando consequentes incorreções no produto final.

Trataremos mais adiante do termo “requisito”, mas já convém defini-lo como uma condição para se alcançar certo fim (REQUISITO, 2001). O cenário era ainda agravado pela inexistência de métricas que retornassem avaliações seguras dos produtos entregues pelos desenvolvedores. Uma pergunta do tipo “o produto que entregamos está adequado às demandas do cliente?” poderia não ser respondida com segurança.

Quando, apesar das adversidades, o programa era entregue, o processo de implantação tendia a ser turbulento, já que raramente eram considerados os impactos que o novo sistema causaria na organização.

Treinamentos aos usuários após a implantação não era atividade prioritária e o fator humano era ignorado com frequência, gerando insatisfações nos funcionários impactados. Por fim, há que se considerar a dificuldade em se empreender manutenção nos produtos em funcionamento, normalmente ocasionados por projetos mal elaborados.

MITOS SOBRE DESENVOLVIMENTO DE SOFTWARE

Conforme nos ensinam Pressman e Maxim (2016), daquela época ficaram alguns mitos sobre o desenvolvimento de um software. Embora os profissionais atuais estejam mais capacitados a reconhecê-los e a evitarem seus efeitos, ainda é necessário dar atenção a eles. Trataremos de dois deles aqui.

MITOS DE GERENCIAMENTO

Sob pressão, gestores podem apoiar-se em crenças relacionadas ao gerenciamento de seus projetos de software. Um mito bastante comum é que um livro repleto de práticas e padrões vai ser instrumento suficiente para o sucesso da empreitada, bastando segui-lo à risca. O questionamento necessário à validade de tal livro é sua completude e adequação às modernas práticas da Engenharia de Software. Outro mito comum é que, mediante atraso em uma entrega, a inclusão de mais desenvolvedores no projeto ajudaria a evitar atraso maior. Por fim, é perigosa a crença de que, ao terceirizar o desenvolvimento de um produto, a empresa que o terceirizou pode deixar a cargo do terceiro todo o desenvolvimento, sem necessidade de gerenciá-lo (PRESSMAM, MAXIM, 2016).

MITOS DOS CLIENTES

Aqui, os autores destacam crenças relacionadas ao papel e ao comportamento dos clientes em um projeto. Segundo um dos mitos, basta ao cliente fornecer uma definição geral dos objetivos do software para que seja possível iniciar sua elaboração. Na verdade, a realidade é que a definição mais clara e mais completa possível dos requisitos é o caminho mais seguro para o bom início de um projeto (PRESSMAM, MAXIM, 2016).

REFLITA

Em 2002, uma empresa global de pesquisa em Tecnologia da Informação chamada *Cutter Consortium* constatou que 78% das empresas de TI pesquisadas fizeram parte de ações judiciais motivadas por desavenças relacionadas a seus produtos. Na maioria desses casos (67%), os clientes reclamavam que as funcionalidades entregues não correspondiam a suas demandas. Em outros tantos casos, a alegação era de que a data prometida para entrega havia sido desrespeitada várias vezes. E, por fim, em menor

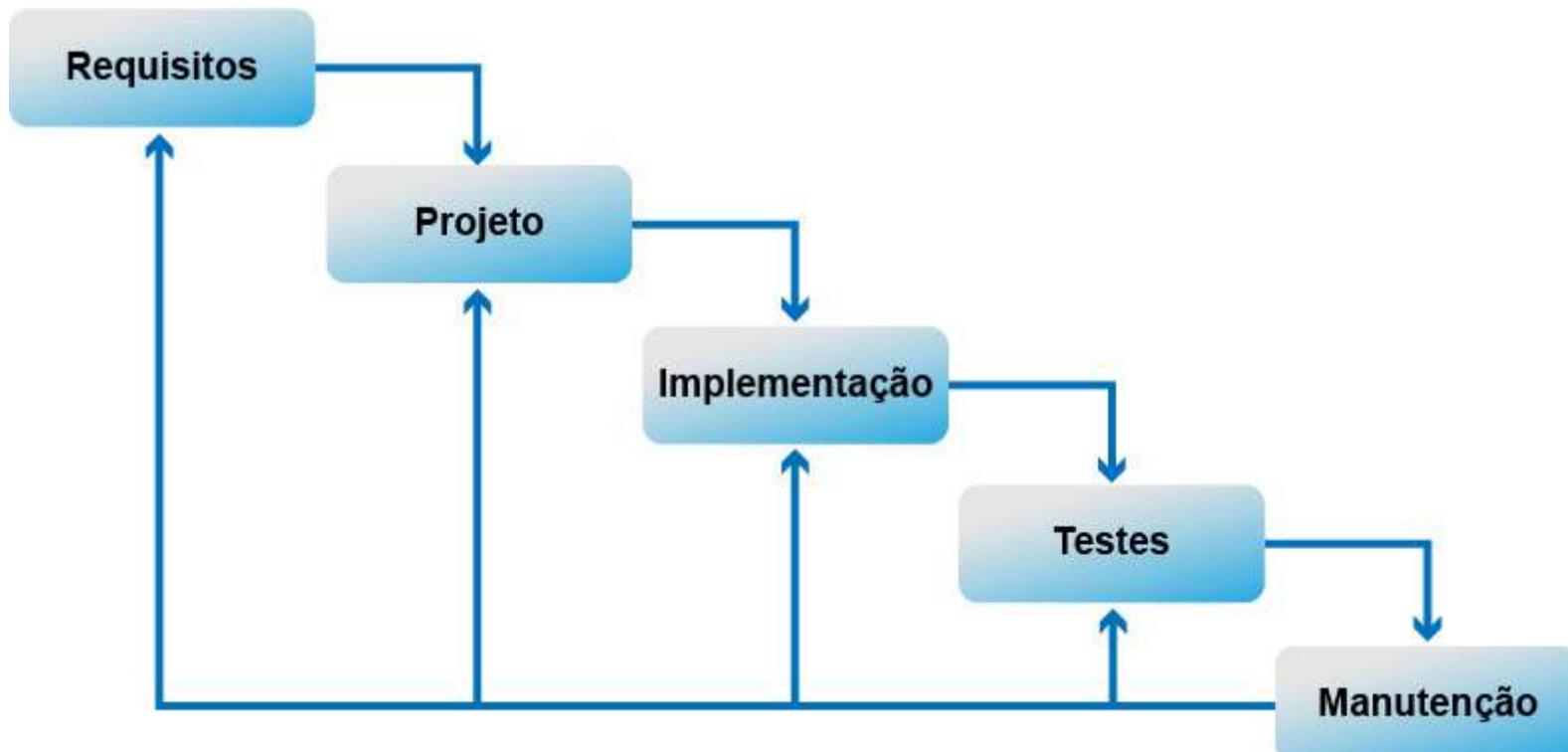
quantidade, a reclamação se originava do fato de o sistema ser tão ruim que simplesmente não se podia utilizá-lo (SCHACH, 2009).

MODELO CASCATA

Para o bem do futuro da Computação como uma atividade humana viável, uma metodologia estável e bem estruturada de desenvolvimento de software deveria ser criada. A resposta da comunidade ligada ao software, então, veio através do **modelo em cascata**, também conhecido como **modelo tradicional**, o qual ainda é utilizado para desenvolvimento de produtos de software. Ele descreve, por meio de etapas bem definidas, o ciclo que o software cumprirá durante o período compreendido entre sua concepção e sua descontinuidade.

O **ciclo de vida** natural de um software, de acordo com Resende (2005), abrange as seguintes fases: concepção, construção, implantação, implementações, maturidade, declínio, manutenção e descontinuidade. Essas fases são mais comumente descritas como **fase de requisitos**, **projeto**, **implementação**, **teste** e **manutenção**. A Figura 1.1 mostra o esquema geral das fases do modelo cascata.

Figura 1.1 | Representação das fases do modelo em cascata



Fonte: adaptado de Schach (2009, p. 51).

Observe que uma fase do processo depende do artefato gerado pela fase anterior. Um artefato nesse contexto não é exatamente um produto de software acabado. As setas de retroalimentação, dispostas no sentido contrário à cascata, indicam a possibilidade de retornos às fases anteriores, considerando a ocorrência de falhas. A volta a uma fase anterior serve, em tese, para sanar problemas que, se levados adiante, acarretariam mais prejuízo ao desenvolvimento.

Antes de terminarmos esta seção, vale a apresentação introdutória de cada uma das etapas do modelo em cascata, de acordo com Schach (2009):

- **Requisitos:** a fase de requisitos de software preocupa-se com a descoberta, a análise, a especificação e a validação das propriedades que devem ser apresentadas para resolver tarefas relacionadas ao software a ser desenvolvido. Fazem parte dos requisitos de um software suas funções, suas características, suas restrições e todas as demais condições para que ele exista e cumpra seu objetivo. O projeto de um software fica comprometido quando o levantamento dos requisitos é executado de forma não apropriada. Eles expressam as necessidades e as restrições colocadas num produto de software, as quais contribuem para a solução de algum problema do mundo real (SCHACH, 2009).
- **Projeto:** uma vez tratados os requisitos, o modelo nos leva ao desenho do produto. Se os requisitos nos mostram o que o sistema deverá fazer, o projeto deverá refletir como ele o fará. Por meio do projeto, os requisitos são refinados de modo a se tornarem aptos a serem transformados em programa. O trabalho principal de um projetista é o de decompor o produto em módulos, que podem ser conceituados como blocos de código, que se comunicam com outros blocos por meio de interfaces (SCHACH, 2009).
- **Implementação:** nessa fase, o projeto é transformado em linguagem de programação para que, de fato, o produto passe a ser

executável. Como estratégia de implementação, a equipe poderá dividir o trabalho de forma que cada programador (ou um pequeno grupo deles) fique responsável por um módulo do sistema.

Idealmente, a documentação gerada pela fase de projeto deve servir como principal embasamento para a codificação, o que não afasta a necessidade de novas consultas ao cliente e à equipe de projetistas (SCHACH, 2009).

- **Testes:** testar significa executar um programa com o objetivo de revelar a presença de defeitos. Caso esse objetivo não possa ser cumprido, considera-se o aumento da confiança sobre o programa. As técnicas funcional e estrutural são duas das mais utilizadas a fim de se testar um software. A primeira baseia-se na especificação do software para derivar os requisitos de teste e a segunda no conhecimento da estrutura interna (implementação) do programa (SCHACH, 2009).
- **Manutenção:** os esforços de desenvolvimento de um software resultam na entrega de um produto que satisfaça os requisitos do usuário. Espera-se, contudo, que o software sofra alterações e evolua. Uma vez em operação, defeitos são descobertos, ambientes operacionais mudam e novos requisitos dos usuários vêm à tona. A manutenção de software é definida como modificações em um produto de software após a entrega ao cliente a fim de corrigir falhas, melhorar o desempenho ou adaptar o produto ao um ambiente diferente daquele em que o sistema foi construído (SCHACH, 2009).

Esse foi o conteúdo que queríamos compartilhar nesta seção inicial. As informações compartilhadas aqui servirão como base para novos aprendizados e possibilitarão que comparações sejam feitas, sobretudo entre a metodologia em cascata (tida como tradicional) e a ágil, entendida como mais moderna.

Questão 1

A engenharia de requisitos começa com a concepção (uma tarefa que define a abrangência e a natureza do problema a ser resolvido). Ela prossegue para o levantamento (uma tarefa de investigação que ajuda os envolvidos a definir o que é necessário) e, então, para a elaboração (na qual os requisitos básicos são refinados e modificados) (PRESSMAN; MAXIM, 2016).

Considerando o conteúdo de requisitos de software no contexto do modelo em cascata, analise as afirmações a seguir.

- I. Requisitos são elementos desejáveis, porém opcionais em um processo de software conduzido com base no modelo em cascata.
- II. As características gerais, as funções a serem executadas e as restrições de um software são partes dos seus requisitos.
- III. A prática tem ensinado que o levantamento dos requisitos pode ser postergado para a fase de implementação do produto.

É verdadeiro o que se afirma em:

a. II e III apenas.

b. II apenas.

c. I e II apenas.

d. III apenas.

e. I, II e III.

Questão 2

Quando executado mediante aplicação de uma metodologia definida e conhecida, o processo de desenvolvimento de um software deverá alcançar um resultado muito mais satisfatório do que quando executado em um ambiente de ausência de formalidade.

Considerando esse fato, assinale a alternativa que contém a real vantagem em se adotar uma metodologia definida no processo de desenvolvimento de um software.

- a. Facilita a descoberta de maus profissionais da organização, já que o processo padronizado ajuda a destacar pessoas sem aptidão.
- b. Promove a produção meramente baseada em padrões, o que desestimula iniciativas potencialmente nocivas ao projeto.
- c. Possibilita que os membros do projeto foquem mais nas tarefas de implementação do produto do que em atividades de menor importância.
- d. Permite sejam identificados os membros da equipe que, de fato, conhecem os processos da Engenharia de Software.
- e. Promove a produção de artefatos mais uniformizados, já que a previsibilidade do processo ajuda a equipe a trabalhar de forma mais padronizada.

Questão 3

A manutenção deve ser evitada a todo custo, já que os produtos de software são entregues sempre em seu estado final. Por isso, a necessidade de manutenção em um software revela que ele não foi bem construído, ainda mais se a manutenção necessária for para corrigir falhas. Considerando esses fatos, a moderna Engenharia de Software tem desconsiderado a manutenção como uma das etapas do modelo em cascata.

Levando em consideração o conteúdo do texto, assinale a alternativa correta.

a. O conteúdo do texto revela fatos verdadeiros sobre a Manutenção de Software, mas está incorreto no trecho em que a exclui das etapas do modelo em cascata.

b. O conteúdo do texto é integralmente incorreto, pois a manutenção é parte integrante do processo de software e sua adoção sempre será necessária.

c. O conteúdo do texto não condiz com a atual realidade conceitual da manutenção, mas revela como essa etapa deveria ser idealmente.

d. O conteúdo do texto está integralmente correto, sobretudo no trecho em que afirma que a necessidade de manutenção revela um produto mal elaborado.

e. O conteúdo do texto não permite que o leitor conclua, de forma inequívoca, pela necessidade de se aplicar manutenção em um software.

REFERÊNCIAS

IEEE Computer Society. **Guide to the Software Engineering Body of Knowledge**. Piscataway: The Institute of Electrical and Electronic Engineers, 2004.

PRESSMAN, R. S., MAXIM, B. R.; **Engenharia de Software: uma abordagem profissional**. 8. ed. Porto Alegre: AMGH, 2016. Disponível em: <https://bit.ly/3r4DQ1i>. Acesso em: 23 nov. 2020.

REQUISITO. /n: HOUAISS, A, VILLAR, M. de S. **Minidicionário Houaiss da Língua Portuguesa**. Rio de Janeiro: Objetiva, 2001.

RESENDE, D. A. **Engenharia de Software e Sistemas de Informação**. 3. ed. Rio de Janeiro: Brasport, 2005.

SCHACH, S. R. **Engenharia de Software**: os paradigmas clássicos e orientados a objetos. 7. ed. São Paulo: McGraw-Hill, 2009.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011. Disponível em: <https://bit.ly/34owzjb>. Acesso em: 12 out. 2020.

WAZLAWICK, R. S. **Engenharia de software**: conceitos e práticas. Rio de Janeiro: Elsevier, 2013.

FOCO NO MERCADO DE TRABALHO

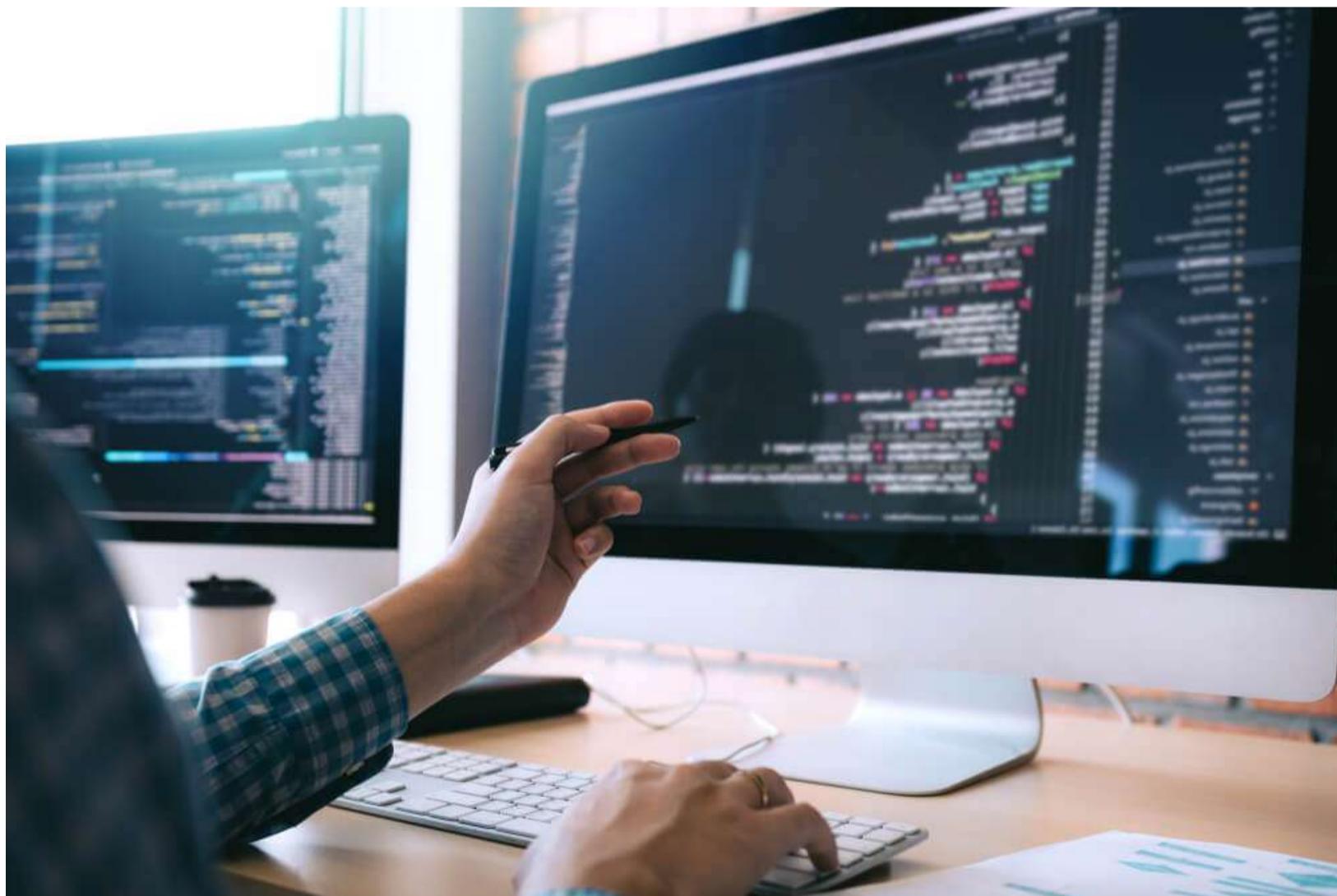
INTRODUÇÃO À ENGENHARIA DE SOFTWARE

Roque Maitino Neto

Ver anotações

APLICAÇÃO DO MODELO EM CASCATA

O modelo em cascata ou modelo tradicional, ainda é utilizado para desenvolvimento de produtos de software. Ele descreve, por meio de etapas bem definidas, o ciclo que o software cumprirá durante o período compreendido entre sua concepção e sua descontinuidade.



Fonte: Shutterstock.

Deseja ouvir este material?

SEM MEDO DE ERRAR

Iniciamos a resolução deste desafio resgatando o contexto em que ele está inserido: você foi contratado por uma pequena empresa de desenvolvimento para atuar como engenheiro de software júnior. Ao chegar na organização, notou que ela operava em um cenário caótico. Sua primeira providência foi a de implementar um procedimento viável, padronizado e consagrado de desenvolvimento de software, com etapas e tarefas definidas e conhecidas pelos integrantes do projeto. Para atingir esse objetivo, propôs-se a introduzir o modelo em cascata em uma apresentação.

De fato, não há uma resolução única para este desafio, pois apresentações podem variar conforme a visão de seu autor, mesmo com um tema definido a ser desenvolvido. O que segue, portanto, são sugestões de conteúdo das lâminas da apresentação.

Lâmina 1: conceito de modelo de processo

Nesta lâmina introdutória, você pode mostrar o conceito de um modelo de processo, categoria em que o modelo em cascata está posicionado. Na página 40 da obra de Pressman e Maxim (2016), você encontra uma boa definição de modelo. Esse conteúdo prepara o caminho para a definição de modelo em cascata.

Lâmina 2: conceito de modelo em cascata

Uma vez entendido do que trata um modelo de processo, você deve definir o modelo em cascata, fornecer suas características gerais e posicioná-lo no contexto de um modelo de processo. A partir da página 41 da obra de Pressman e Maxim (2016) você encontra a visão dos autores sobre esse modelo.

Lâmina 3: esquema geral do modelo em cascata

Esta lâmina apresenta prevalência de conteúdo gráfico, pois nela será colocado o esquema geral do modelo em cascata, com suas etapas e a indicação de retroalimentação em fluxo linear.

Lâmina 4: definição da etapa de requisitos

Nesta lâmina você deverá conceituar a etapa de requisitos, ressaltando sua criticidade no processo global de produção de um software.

Requisitos são as condições necessárias para que algo seja construído.

Um requisito de software se traduz nas funcionalidades do sistema, em suas restrições e nas condições gerais de funcionamento.

Lâmina 5: definição da etapa de projeto

Nesta lâmina você deverá inserir a etapa de projeto como **tarefa** posterior ao tratamento dos requisitos, ressaltando que se trata da solução preliminar do problema, elaborada pelo projetista e com base nos requisitos. Mais uma vez, a importância da etapa deverá ser ressaltada.

Lâmina 6: definição da etapa de implementação

Será pela implementação que os requisitos e o projeto serão transformados em um artefato executável. Nesta lâmina você deve aproximar o conceito de implementação do cotidiano dos membros da empresa de Tecnologia da Informação, pois esta sempre foi a etapa com maior foco no trabalho deles. Implementar é transformar em código o que foi levantado e organizado em etapas anteriores do processo.

Lâmina 7: definição da etapa de testes

Implementada parte ou a totalidade do sistema, chega o momento de testar o código, a fim de aumentar a confiabilidade em seu funcionamento. Esta etapa também tende a ser mais familiar à equipe, pois os testes sempre eram realizados, embora sem um procedimento padrão. Testar, portanto, é executar o código em busca de inconsistências.

Lâmina 8: definição da etapa de manutenção

Por fim, nesta lâmina, a etapa de manutenção deverá ser conceituada e posicionada como parte importante de todo o processo. É por meio da manutenção que um sistema já entregue passa por melhorias, correções e adequações, a fim de que continue a ser adequado aos propósitos do cliente.

Dessa forma, o desafio de prover a equipe com informações e de motivá-la para a adoção do modelo em cascata será muito bem concluído, e o caminho para a introdução de outras inovações estará aberto.

o

Ver anotações

AVANÇANDO NA PRÁTICA

UM MODELO SIMPLES PARA COLETA DE REQUISITOS

Certa empresa de desenvolvimento de software vem enfrentando contratemplos com a falta de padronização de seus métodos e dos documentos utilizados durante as etapas do desenvolvimento de seus produtos. No que se refere à coleta de requisitos, por exemplo, cada engenheiro de software usa seus próprios métodos de anotação e seu próprio formulário, o que acaba causando dúvidas no pessoal que deve analisar e organizar os requisitos levantados. Não raramente, há requisitos que seguem em formato incorreto para a fase de projeto e se tornam fonte de retrabalho para a equipe.

Ao ser contratado como engenheiro de software da empresa, você detectou o problema e resolveu criar um modelo de formulário para levantamento de requisitos. Esse modelo deveria ser intencionalmente simples e funcionar como a primeira versão de um modelo futuro mais elaborado e completo. Sua missão nesta atividade é, portanto, propor um formulário simples, o qual deverá ser usado como modelo de coleta de requisitos para toda a organização como forma de iniciar a padronização do procedimento.

Boa sorte e mãos à obra!

Logo que foi admitido como engenheiro de software de uma empresa desenvolvedora, você detectou que um modelo de formulário para a coleta de requisitos de software deveria ser criado como forma de organizar e de proporcionar uma padronização a essa etapa. Com esse objetivo em mente e ciente de que se trata de um formulário simples de coleta de requisitos, uma proposta possível para o desafio é a que segue:

1. Finalidade geral do sistema.
2. Características gerais do sistema e público-alvo.
3. Ambiente operacional (onde o sistema deverá ser executado).
4. Descrição das funcionalidades.
5. Restrições do sistema.

Há que se registrar que um documento completo de especificação de requisitos de software deverá conter quantidade mais variada de informações. No entanto, para os fins a que se destina, a solução aqui oferecida é satisfatória.

NÃO PODE FALTAR

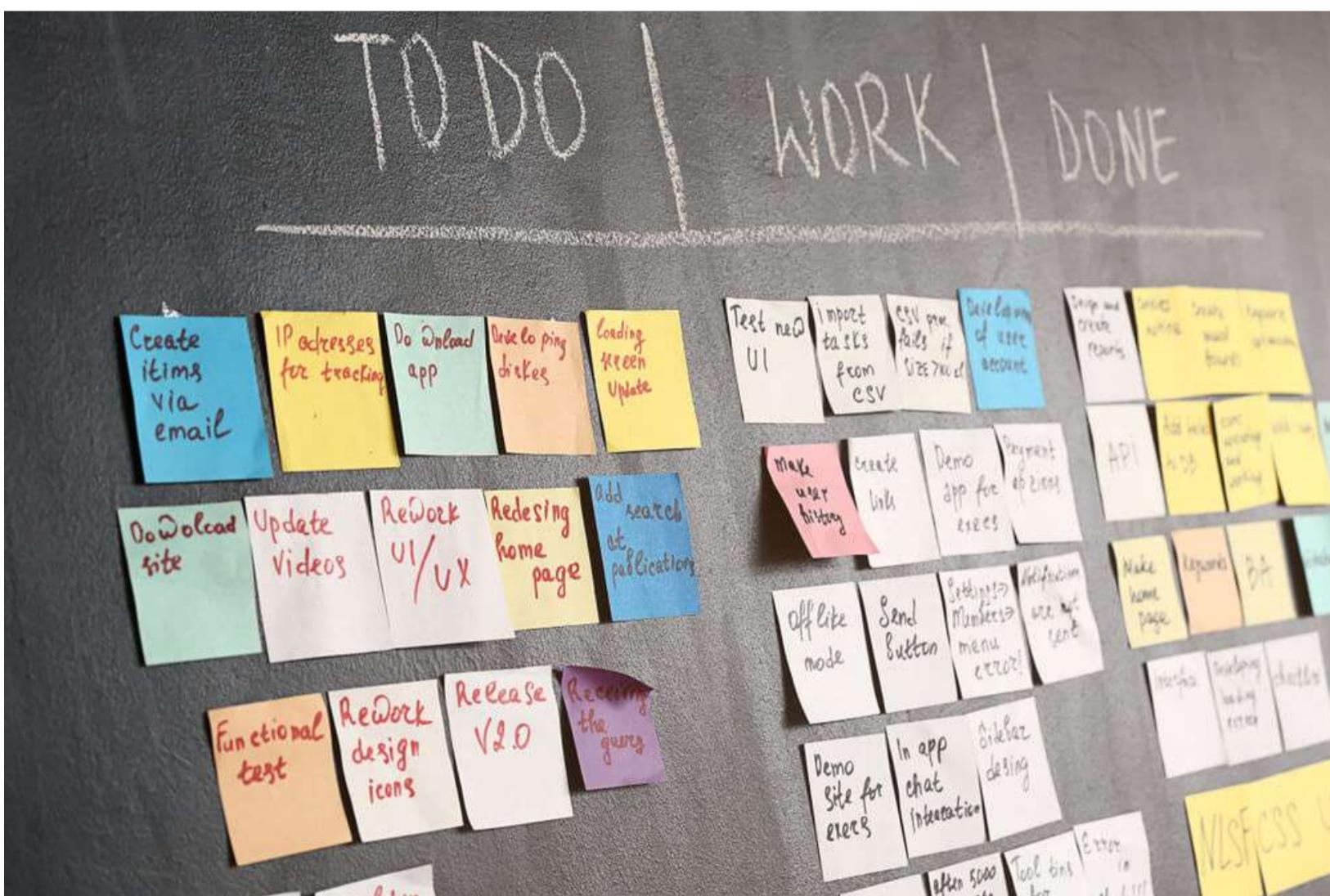
METODOLOGIAS ÁGEIS

Roque Maitino Neto

Ver anotações

MODELO ÁGIL PARA A GESTÃO DE PROJETOS DE SOFTWARE

As metodologias ágeis são conjuntos de práticas para gerenciar projetos mais adaptável às mudanças. Elas são estruturadas em pequenos ciclos de trabalho, que geram entregas, essas entregas acrescentam um incremento ao produto final, assim, a cada novo ciclo, é entregue um conjunto de funcionalidades no produto.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Caro aluno, prezada aluna, seja bem-vindo, seja bem-vinda! Aqui estamos para mais uma seção do nosso conteúdo de Engenharia de Software. Embora as empresas de software tenham prosperado durante décadas utilizando uma única metodologia (ou, na melhor das hipóteses, variações dela), parecia claro para a comunidade de engenheiros de software que algo precisava ser melhorado. A presunção dos requisitos estáveis, os longos períodos sem interações entre cliente e desenvolvedor e o entendimento de que uma linha de produção linear seria adequada para a produção de software tornaram-se gatilhos para que, no começo do século XXI, um grupo de profissionais da computação lançasse um manifesto contendo as bases para uma nova experiência de criação de software.

Chamado genericamente de desenvolvimento ágil, esse novo paradigma começou a ganhar adeptos mundo afora e hoje é adotado por importantes organizações. Como qualquer mudança proposta sobre algo que já se faz há muito tempo da mesma forma, essa renovação também provocou desconfiança, e as inevitáveis comparações entre metodologias (a tradicional e a ágil) foram frequentes.

Para que você forme suas próprias bases comparativas e seja capaz de extrair os pontos positivos de cada paradigma, esta seção se propõe a apresentar as metodologias ágeis, com ênfase no *Extreme Programming* e no *Scrum*, introduzindo suas próprias aplicações e ferramentas. Além disso, a seção também estabelecerá comparações, sempre que possível, das práticas que cada uma das metodologias consagrou, a fim de verificarmos juntos se, de fato, o esforço de mudança de paradigma tem valido a pena.

Aqui estamos em nosso segundo desafio da unidade e, logo de início, vale a pena relembrarmos as circunstâncias em que ele se desenrola. Ao iniciar as atividades em uma empresa de desenvolvimento de software, você encontrou uma situação de ausência de padrão nos projetos e,

mediante as circunstâncias de urgência e de falta de familiaridade das equipes com modelos, resolveu implementar o Ciclo de Vida Clássico como padrão de desenvolvimento dos seus produtos de software.

Embora a iniciativa tenha sido muito bem-sucedida, o modelo implementado não era exatamente adequado às novas demandas que a organização vinha assumindo e, mesmo sendo uma forma bastante estruturada de desenvolvimento, apresentava falhas que provocavam muito retrabalho e atrasos nas entregas. Por isso, você decidiu dar um passo adiante e implementar uma metodologia ágil de desenvolvimento na empresa.

Depois de um período planejando a mudança, eis que chega o momento de a empresa assumir o primeiro projeto a ser desenvolvido sob os princípios e práticas do Scrum. Considerando sua natureza simples e estruturada, o produto a ser construído foi apropriado para servir como primeira experiência no mundo ágil: o dono de uma pequena loja de itens domésticos solicitou a criação de um site que expusesse sua marca e seus produtos na internet e que possibilitasse a seu visitante cadastrar-se para ter acesso a conteúdos exclusivos.

Depois de entendidas pela equipe, as necessidades do cliente foram transformadas pelo Product Owner em uma lista de tarefas, chamada pelo Scrum de Product Backlog. Na sequência, a primeira tarefa dessa lista foi esmiuçada em tarefas menores, que se transformaram no Sprint Backlog. Considerando os processos da metodologia Scrum, seu desafio consistirá em:

1. Criar o *Product Backlog* com base na descrição do produto, feita no enunciado, contendo de quatro a seis estórias (ou requisitos), cada uma delas classificada com a prioridade alta, média ou baixa.
2. Detalhar ao menos três tarefas da primeira estória do *Product Backlog*. Essas tarefas comporão o primeiro *Sprint Backlog*.
3. Criar um *post-it* em que a primeira tarefa do *Sprint Backlog* esteja descrita e que inclua seu nível de prioridade, a classificação de

esforço para executá-la, a quantidade de horas estimada para sua conclusão, o nome do responsável, a estória à qual a tarefa pertence e sua descrição resumida. A escolha desses dados fica a seu critério, mas precisam ser coerentes com a tarefa.

Bom trabalho!

DICA

Enfim, estudar e compreender como as metodologias ágeis são efetivadas no cotidiano das organizações significa estar atualizado com o que há de mais novo nos procedimentos de desenvolvimento de software.

CONCEITO-CHAVE

Aquilo que não é bom não dura por muito tempo. Se a seleção natural é infalível na natureza, ela parece atuar com rigor semelhante no mundo do desenvolvimento de software ao permitir que apenas as metodologias mais aptas sobrevivam ao teste do tempo. Foi por isso que o Modelo em Cascata atravessou algumas décadas oferecendo aos engenheiros de software um modo razoavelmente seguro e efetivo de criar seus produtos, em substituição ao desenvolvimento especulativo e caótico praticado antes da estruturação de metodologias de desenvolvimento de software.

Mas, se o Modelo em Cascata é realmente bom, talvez o termo “razoavelmente” tenha sido mal colocado na sentença, não é mesmo? Bem, nesta seção teremos oportunidade de entender alguns elementos desse modelo que justificaram a mudança de paradigma iniciada no ano de 2001 e vivenciada até hoje. O Ciclo de Vida Tradicional – outra forma como o Modelo em Cascata é conhecido – apresenta falhas de concepção que o acabaram tornando obsoleto diante das demandas de tempo e agilidade atuais de produção de software.

Mas, afinal, quais são os problemas da metodologia tradicional? Qual o motivo da dificuldade dela em acompanhar as inevitáveis mudanças de requisitos de um produto durante seu ciclo de desenvolvimento? Por que o cliente tem dificuldade em reconhecer valor no que está sendo desenvolvido? Mais do que responder a essas questões, esta seção tem a intenção de apontar soluções. Antes de abordarmos diretamente as Metodologias Ágeis, convém tratarmos ainda de alguns aspectos do processo tradicional de desenvolvimento para que as comparações sejam feitas de forma adequada.

Conforme estudamos na primeira seção, o processo tradicional de desenvolvimento baseia-se na construção linear do sistema, seguindo uma sequência definida de fases, com a particularidade de uma etapa do processo utilizar o resultado obtido pela etapa anterior para criar seu artefato (WAZLAWICK, 2013). Há também a possibilidade de que o fluxo retorne para etapas anteriores em havendo necessidade de ajustes. Essa construção linear da metodologia baseia-se na ideia de que todas as coisas podem ser construídas como se estivessem em uma linha de montagem: a matéria-prima é inserida na linha de produção e, após algumas etapas determinadas, o produto final está pronto.

A Figura 1.2 ilustra a ideia de utilização dos princípios de uma linha de produção no processo de desenvolvimento de software. Observe que na primeira etapa os requisitos do produto de software representam as matérias-primas. Depois, como em uma linha de montagem tradicional, os especialistas em cada área do processo atuam na linha de montagem até que um produto de software acabado esteja disponível.

Figura 1.2 | Representação da criação de um software em uma linha de montagem



É essa fundamentação nas formas tradicionais de fabricação que nos permite atribuir ao **Modelo em Cascata** três características bem particulares segundo Teles (2006): o determinismo, a especialização e o foco na execução. Para explicar esses conceitos, usaremos a analogia com o processo de montagem de veículos. Vejamos:

- **Determinismo:** materiais alimentam um processo de fabricação e, ao final da linha de produção, temos um automóvel terminado. As alterações pelas quais os materiais passam são determinísticas e devem sempre gerar um resultado conhecido. Um processo assim estruturado tende a gerar segurança, redução de tempo e de custo.
- **Especialização:** o processo tradicional de manufatura divide a montagem desse carro em atividades especializadas, desenvolvidas por trabalhadores igualmente especializados. Assim, numa linha de montagem automotiva, haverá a etapa de soldagem do chassi, de colocação do motor, da montagem do interior e assim por diante, todas elas executadas por especialistas em cada função.
- **Foco na execução:** se as transformações na linha de montagem já estão determinadas e se cada etapa será executada por especialistas, então não há muito em que pensar. Basta executar.

Não é difícil inferir, então, que o **Modelo em Cascata** tenha sido concebido com base nessas ideias de desenvolvimento industrial em linha. De acordo com essas ideias, a mera obediência a eventos consecutivos (de requisitos até implantação), à especialização (funções de analista, projetista, programador, testador) e ao foco na execução já seria capaz de criar um produto de qualidade, no tempo estipulado pelo cliente e nos limites do orçamento. Havia – e ainda há – a presunção de que a sequência de etapas do projeto seja transformada corretamente em software (TELES, 2006).

Outra presunção que o modelo tradicional de desenvolvimento assume é a de que o profissional do software cumpre trabalhos repetitivos, previamente determinados e que pouco dependem das suas habilidades intelectuais. No entanto, os desenvolvedores podem ser classificados como trabalhadores do conhecimento, pois cumprem suas tarefas com base em seu raciocínio e raramente seguem um processo linear em suas investidas criativas. A eles, portanto, deve ser dada a oportunidade de aplicar tantas revisões quantas forem necessárias em suas criações.

Se as metodologias tradicionais de desenvolvimento se baseiam em modelos de construção que não se adequam especificamente ao produto que desejam entregar, as metodologias ágeis nasceram ajustadas para a criação de software. Os métodos ágeis enfatizam menos as definições de atividades e mais os fatores humanos inerentes ao desenvolvimento de programas de computador (WAZLAWICK, 2013). Eles são, portanto, claramente mais adequados à natureza do trabalho de profissionais de TI, já que preveem a necessidade de sucessivas revisões na obra. Atividades intelectuais não são executadas de forma linear e não são determinísticas.

Durante o processo de criação de um software, é mais do que natural que os requisitos sejam revistos, decisões sejam alteradas e detalhes sejam resgatados. Afinal, ao explicar pela primeira vez as funcionalidades que deseja para o produto, o cliente ainda não as conhece por completo e a visão global do que necessita ainda não está formada. Que tal um procedimento que dê a ele oportunidade de aprender e de mudar de ideia ao longo do desenvolvimento? Você certamente não notou essa atenção com o cliente ao longo do conteúdo do Modelo em Cascata.

Sobre o aprendizado do cliente em relação às suas próprias necessidades, Teles (2006) entende que ele está relacionado ao feedback que o software fornece ao cliente quando ele tem a

oportunidade de utilizá-lo, mesmo na versão não completa. No desenvolvimento ágil, o conceito de feedback está presente ao longo de todo o desenvolvimento do software e exerce um papel fundamental.

Como, então, podemos dar ao cliente a chance de aprender sobre suas reais necessidades e de mudar de opinião durante o processo de desenvolvimento?

As práticas relacionadas aos métodos ágeis serão apresentadas nas próximas páginas e você conhecerá elementos do *Extreme Programming* e do *Scrum*, duas das mais utilizadas metodologias ágeis.

Porém, antes de abordarmos diretamente as duas práticas, vale a menção ao ato que serviu como marco inicial do pensamento ágil de desenvolvimento. No ano de 2001, um documento contendo a declaração dos doze princípios que fundamentam o desenvolvimento ágil foi divulgado por um grupo de profissionais de TI. Através desse documento, chamado **Manifesto Ágil**, seus autores declaravam que indivíduos e interações importavam mais do que processos e ferramentas; que softwares em funcionamento importam mais do que documentação; que colaboração com o cliente importa mais do que negociação de contratos e que responder a mudanças é melhor do que seguir um plano (BECK et al., 2001). Você pode conferir a íntegra do manifesto e conhecer seus autores na página *Agile Manifesto*, na internet (BECK et al., 2001).

| VISÃO GERAL DO *EXTREME PROGRAMMING* (XP)

O XP é uma metodologia adequada para empreitadas em que os requisitos mudam com certa constância, além de ser ajustado para equipes pequenas e para o desenvolvimento de programas orientados a objetos (TELES, 2006). Ao contrário das metodologias tradicionais, ele é

indicado também para ocasiões em que se desejam partes executáveis do programa logo no início do desenvolvimento e que ganhem novas funcionalidades conforme o projeto avança.

o

REFLITA

O XP, assim como as outras metodologias ágeis, defende que a criação de um software segue a mesma dinâmica da criação de uma obra de arte. O trecho a seguir ilustra esse fato:

“

Escrever uma redação, um artigo ou um livro é uma atividade puramente intelectual que se caracteriza pela necessidade de sucessivas revisões e correções até que a obra adquira sua forma final. [...] Quando um pintor cria um novo quadro, é comum começar com alguns esboços, evoluir para uma representação mais próxima do formato final, fazer acertos, retoques e afins até que a obra esteja concluída. (TELLES, 2004. p. 39)

Embora a especialização não seja estimulada nas metodologias ágeis, ainda assim existe a necessidade de se estabelecer funções para os participantes do projeto. Uma típica equipe de trabalho no XP tem a seguinte configuração (TELLES, 2004):

- **Gerente do projeto:** trata-se do responsável pelos assuntos administrativos do projeto e do relacionamento com o cliente. Duas de suas funções mais importantes incluem o estabelecimento de contato entre a equipe e o cliente e o cuidado para que a equipe fique livre de pressões externas e se dedique integralmente ao desenvolvimento.
- **Coach:** este é o nome do responsável técnico pelo projeto, que deve ser tecnicamente bem preparado e experiente. Se, por exemplo, uma nova tecnologia fica disponível no mercado, é função dele sugerir seu uso no produto em desenvolvimento.
- **Analista de teste:** ajuda o cliente a escrever os testes de aceitação e fornece feedback para a equipe interna de modo que as correções no sistema possam ser feitas. Ao contrário do que é feito nas metodologias tradicionais, no desenvolvimento ágil em geral (e no

Ver anotações

XP em particular), o cliente participa ativamente dos testes no produto.

- **Redator técnico:** ajuda a equipe de desenvolvimento a documentar o sistema, permitindo que os desenvolvedores estejam plenamente focados na construção do programa propriamente dito. À propósito, a metodologia recomenda que o código deve ser claro o suficiente para permitir uma documentação mínima do sistema.
- **Desenvolvedor:** realiza análise, ajuda a criar o projeto e codifica o sistema. No XP, não há divisão entre as especialidades de engenheiro de requisitos, projetista e o desenvolvedor propriamente dito. No modelo tradicional, o desenvolvedor apenas programa, via de regra.

Para atingir seus objetivos, o *Extreme Programming* apoia-se em quatro pilares, comumente chamados de valores.

- O primeiro deles é o **feedback**, cuja aplicação pretende alcançar a troca de informações entre cliente e equipe. Assim, quando o cliente aprende com o sistema que utiliza e reavalia suas necessidades, ele gera feedback para sua equipe de desenvolvimento.
- O segundo valor é chamado de **comunicação** e pretende estabelecer contato proveitoso entre equipe e cliente, evitando que os desenvolvedores realizem o trabalho de forma especulativa.

ASSIMILE

Uma das atitudes mais potencialmente nocivas que uma equipe de desenvolvimento pode adotar é o desenvolvimento especulativo. Isso significa, na prática, criar certa funcionalidade do sistema sem saber ao certo suas definições e seus requisitos, fato comumente associado à falha de comunicação com o cliente. Nesses casos, o desenvolvedor pode não ter entendido corretamente uma necessidade do cliente e, sem a devida segurança, acaba

desenvolvendo um produto com base em suas próprias convicções praticamente. A proximidade com o cliente tende a reduzir bastante esse fenômeno.

- O terceiro valor é a **simplicidade**, cuja formulação serve para orientar a equipe a desenvolver apenas o que for suficiente para atender a necessidade do cliente, sem “sobrecarregar” o sistema com funções quase sempre inúteis.
- Por fim, o último valor estabelece a **coragem** como um dos pilares do XP, pois ela será necessária no impulsionamento da equipe para manter sempre o cliente presente, para propor melhorias no que já foi testado e colocado em funcionamento e, de modo geral, para sempre levar adiante as práticas da metodologia.

Como não poderia deixar de ser, esses princípios refletem-se nas práticas e no processo proposto pelo XP. De forma a destacar suas atividades-chave, Pressman e Maxim (2016) assim dividem o processo do XP:

- **Planejamento**: essa atividade, comumente identificada como jogo do planejamento, tem início com o esclarecimento de requisitos do produto e é executada de modo bem diferente daquele proposto pelo modelo tradicional: aqui, o cliente escreve – de próprio punho – as funcionalidades do produto em uma ficha. A essa ficha dá-se o nome de Estória e cada uma delas é avaliada pela equipe em termos de custo e tempo de execução. A qualquer momento, o cliente pode escrever novas histórias.

EXEMPLIFICANDO

Uma estória escrita pelo cliente deve ser pautada na simplicidade e na objetividade, duas características que serão úteis em sua codificação. Ao descrever uma funcionalidade do sistema, o cliente poderá expressar-se em termos sucintos e usar a própria escrita para isso. Para entendermos

como a criação de estórias funciona na prática, vamos considerar um cenário no qual um cliente deseja a criação de um programa que, com base nas informações armazenadas no sistema integrado de sua empresa, seja capaz de apresentar dados consolidados das suas vendas. Essa funcionalidade do novo sistema pode ser assim expressa em uma estória: “O sistema deverá consolidar as vendas por período, região e grupo de produtos, bem como permitir a exportação desses dados consolidados para uma planilha de dados. Além disso, a impressão de relatórios dessas consolidações deverá estar disponível”. Com a concordância do cliente, a equipe poderá fazer sugestões nessa estória e, por exemplo, dividi-la em outras tarefas para que sua implementação seja facilitada.

- **Projeto:** representações complexas da solução certamente não são características desta tarefa, muito menos a implementação de funcionalidades extras no produto não solicitadas pelo cliente. A intenção é que o projeto sirva como um guia de implementação de cada história e que seja compreendido também pelo cliente. Caso a equipe se depare com um problema cuja representação seja muito difícil, a metodologia recomenda que se construa um protótipo executável dessa parte do projeto, reduzindo, assim, o risco de se construir uma versão final equivocada daquela história.
- **Codificação:** mesmo depois de desenvolvidas as histórias e de ter sido feito o trabalho de projeto, a codificação ainda não é iniciada. Ao invés de procurarem codificar uma versão final do produto, os desenvolvedores criam testes de unidade para cada uma das histórias e só então, após feita e validada essa atividade, o desenvolvimento será orientado a uma versão completa e final.
- **Testes:** os testes de unidade criados durante a codificação devem ser aptos à automatização, de modo a permitir que sejam feitos

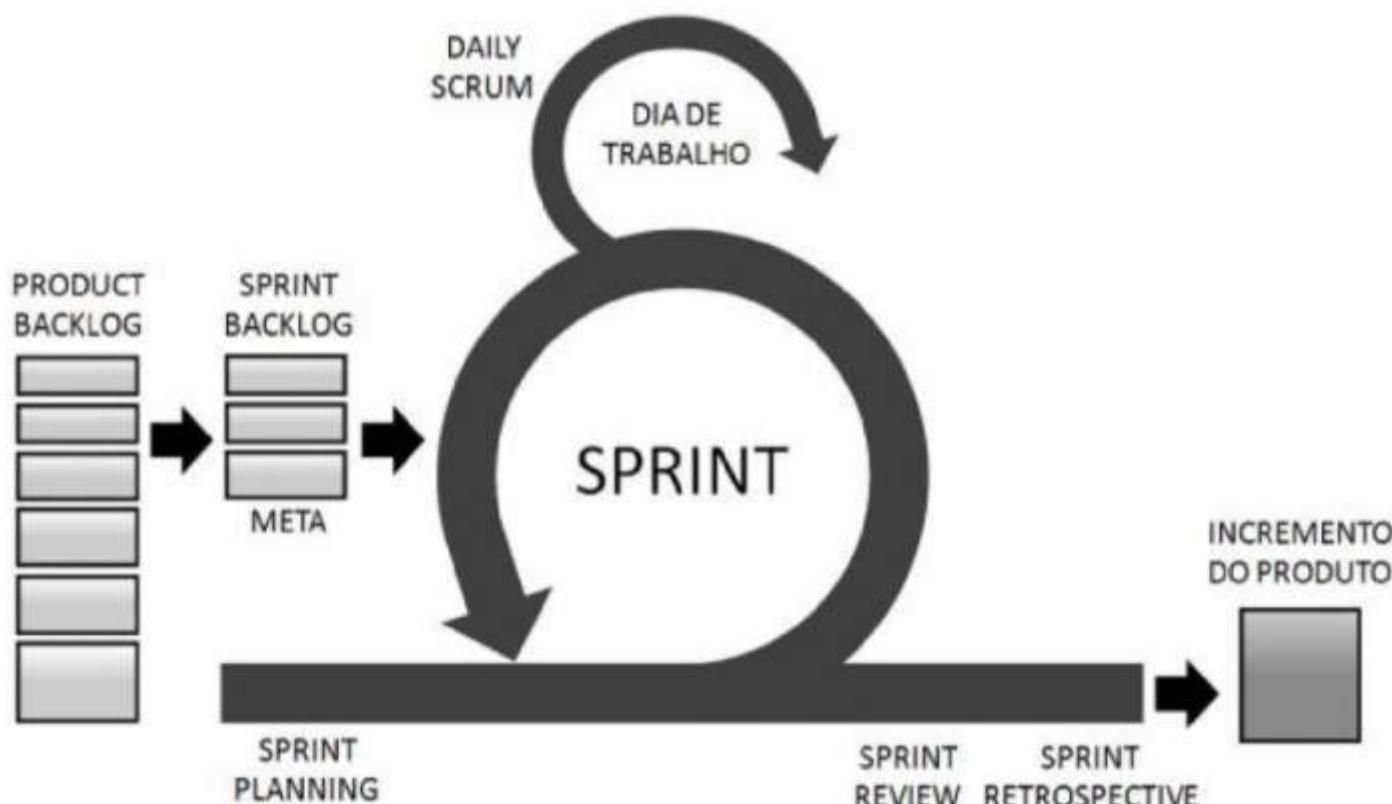
rapidamente e repetidos quando necessário. O conjunto de testes de unidade podem ser usados a qualquer momento para testes de integração e de validação do produto.

Em linhas gerais, é assim que o **Extreme Programming** funciona. No entanto, ele não detém o monopólio das metodologias ágeis.

VISÃO GERAL DO SCRUM

Outro método, também bastante adaptado às demandas atuais de tempo e qualidade, tem sido alternativa para as organizações desenvolvedoras. Trata-se do **Scrum**, um modelo ágil para a gestão de projetos de software, o qual tem, na reunião regular dos seus desenvolvedores para criação de funcionalidades específicas, sua prática mais destacada. Suas práticas guardam semelhança com as próprias do XP, mas possuem nomes e graus de importância diferentes nos dois contextos. Iniciaremos nossa abordagem com o ciclo tradicional do Scrum, ilustrado na Figura 1.3.

Figura 1.3 | O ciclo do Scrum



Fonte: Sabbagh (2013, [s.p.]).

Cada um dos elementos do ciclo é abordado na sequência:

- **Product Backlog:** trata-se da lista que contém todas as funcionalidades desejadas para o produto. O Scrum defende que tal

lista não precisa estar completa logo na primeira vez em que é feita. “Pode-se iniciar com as funcionalidades mais evidentes [...] para depois, à medida que o projeto avançar, tratar novas funcionalidades que forem sendo descobertas” (WAZLAVICK, 2013, p.56).

EXEMPLIFICANDO

Para que possamos compreender a natureza de um Backlog, vamos considerar que está-se iniciando um novo projeto de software e que, antes do efetivo início da criação do produto, um ator do projeto, chamado Product Owner (a descrição desse ator será dada logo a seguir), deva criar uma lista do que será produzido ao longo dessa etapa. Tal documento, chamado Backlog, é uma lista ordenada, ainda não completa, e dinâmica dos itens que o Product Owner acredita que serão produzidos durante o projeto, em seu início (SABBAGH, 2013). Nesse nosso exemplo, o Backlog contém apenas os itens necessários para o começo do desenvolvimento.

Quadro 1.1 | Exemplo de Backlog.lo

Visão do produto: criar um site de vendas que possibilite aos clientes da empresa efetuarem suas encomendas de forma rápida e segura, com a possibilidade de customização dos produtos.

Requisito 1

Requisito 2

Requisito 3

Requisito n

Os requisitos de prioridade menor são colocados ao final da lista, com menor nível de detalhamento. Devemos lembrar, no entanto, que o Backlog é dinâmico e que, por isso, um requisito pode ser movimentado ao longo da lista no decorrer do projeto.

Fonte: adaptado de SABBAGH (2013).

- **Sprint Backlog:** lista de tarefas que a equipe deverá executar naquele Sprint. Tais tarefas são selecionadas do Product Backlog, com base nas prioridades definidas pelo Product Owner.
- **Sprint:** o Scrum divide o processo de efetiva construção do software em ciclos regulares, que variam de duas a quatro semanas (ou em períodos maiores, a depender da complexidade das tarefas). Trata-se do momento em que a equipe se compromete a desenvolver as funcionalidades previamente definidas e colocadas no Sprint Backlog. Se alguma funcionalidade nova for descoberta, ela deverá ser tratada na Sprint seguinte. Cabe ao Product Owner manter o Sprint Backlog atualizado, apontando as tarefas já concluídas e aquelas a serem concluídas.

Embora o modelo Scrum defende que as equipes sejam auto organizadas, ainda assim apresenta três perfis profissionais de relevância:

- **Scrum Master:** trata-se de um facilitador do projeto, um agente com amplo conhecimento do modelo e que preza pela sua manutenção durante todas as etapas do projeto. Deve atuar como moderador ao evitar que a equipe assuma tarefas além da sua capacidade de executá-las.
- **Product Owner:** é a pessoa responsável pelo projeto propriamente dito. Ele tem a missão de indicar os requisitos mais importantes a serem tratados nos sprints.

- **Scrum Team:** é a equipe de desenvolvimento, composta normalmente por um grupo de seis a dez pessoas. A exemplo do *Extreme Programming*, não há divisão entre programador, analista e projetista (WAZLAVICK, 2013).

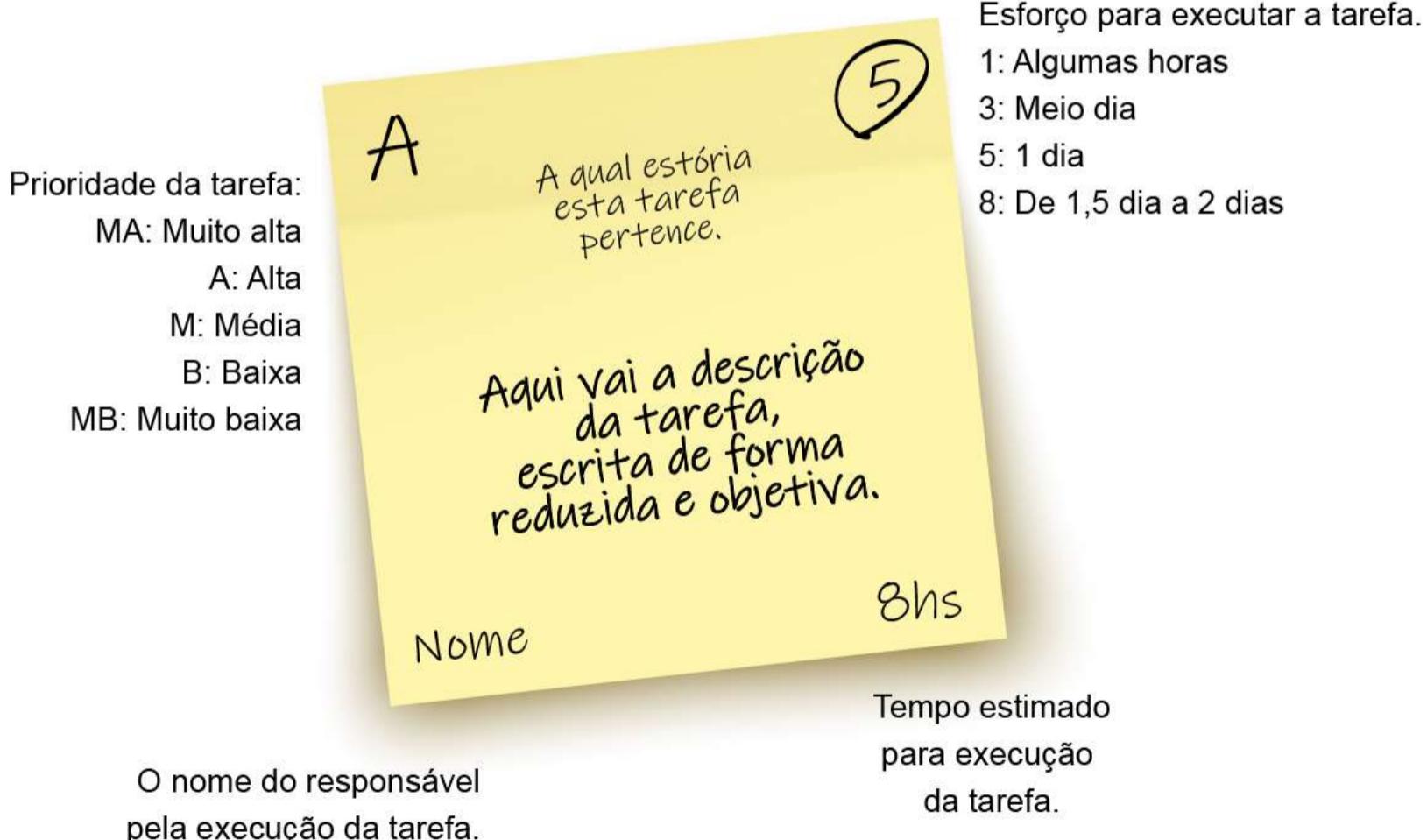
Bem, esse foi o conteúdo de natureza mais teórica que queríamos abordar sobre as metodologias ágeis. No entanto, ainda precisamos tratar do aspecto prático desses modelos e aproveitaremos esta oportunidade para fazê-lo. Conforme já tivemos a oportunidade de discutir, uma das forças que movem as metodologias ágeis é a boa comunicação que deve haver entre os elementos envolvidos em um projeto. É através de boas práticas de comunicação que o cliente poderá se sentir parte integrante (e essencial) do trabalho e poderá expor com segurança o que espera do futuro sistema. Além disso, o bom entrosamento – e a aplicação de técnicas que o aprimoram – é a liga que manterá os membros da equipe mutuamente informados sobre o andamento do projeto.

Imaginemos, então, que você esteja começando seu trabalho em uma desenvolvedora de sistemas, a qual conduz seus projetos de acordo com o Scrum. Em dada ocasião, você acessa uma sala e se depara com um quadro branco afixado na parede, no qual observa um desenho em formato matricial e vários papéis coloridos colados. Um tanto encabulado, pergunta ao seu colega de trabalho do que se trata aquilo e ele diz ser o quadro por meio do qual a equipe mantém o controle das atividades desenvolvidas e a desenvolver do projeto em andamento. Para que você possa reconhecer esse quadro, no caso de essa situação se tornar real, vamos a algumas dicas sobre ele:

- De fato, o **quadro Scrum** é o meio pelo qual a equipe realiza a **gestão visual das atividades do projeto**. Uma análise mais detalhada do guia do framework Scrum revelará que o quadro não faz parte, oficialmente, da metodologia, porém sua adoção foi feita em larga escala pelas equipes e, aparentemente, esse fato não altera sua importância.

- As **divisões do quadro** (daí ele conter desenho com formato matricial) são bem simples: uma coluna identifica a **estória** e as três colunas seguintes representam as tarefas relacionadas a esta estória que estão: **a fazer (to do)**, em **execução (doing)** e **feita (done)**.
- Com essa disposição, cada linha do quadro representa uma estória e suas respectivas tarefas, que são, na verdade, extraídas do backlog do produto e que foram selecionadas para uma determinada Sprint.
- Os papéis coloridos colados no quadro são **post-its**. Eles representam uma determinada característica ou estado daquela tarefa e contêm sua identificação resumida. O exemplo clássico é a do *post-it* vermelho, que pode representar uma tarefa de correção de defeitos em um código. A critério da equipe, uma determinada cor pode identificar um determinado membro da equipe. Entretanto, uma forma mais completa e racional de se usar o *post-it* é a que segue:

Figura 1.4 | Exemplo de controle de atividades por meio de *post-it*



Fonte: elaborada pelo autor.

- A dica final é que tanto o quadro quanto a própria metodologia são adaptáveis à cultura e às necessidades das equipes. Todavia, em sua essência, o quadro ajuda a comunicar a todos o que cada um está

fazendo e em qual estágio da atividade ele está, de forma simples e intuitiva.

Observe, na Figura 1.5, um exemplo de um quadro Scrum:

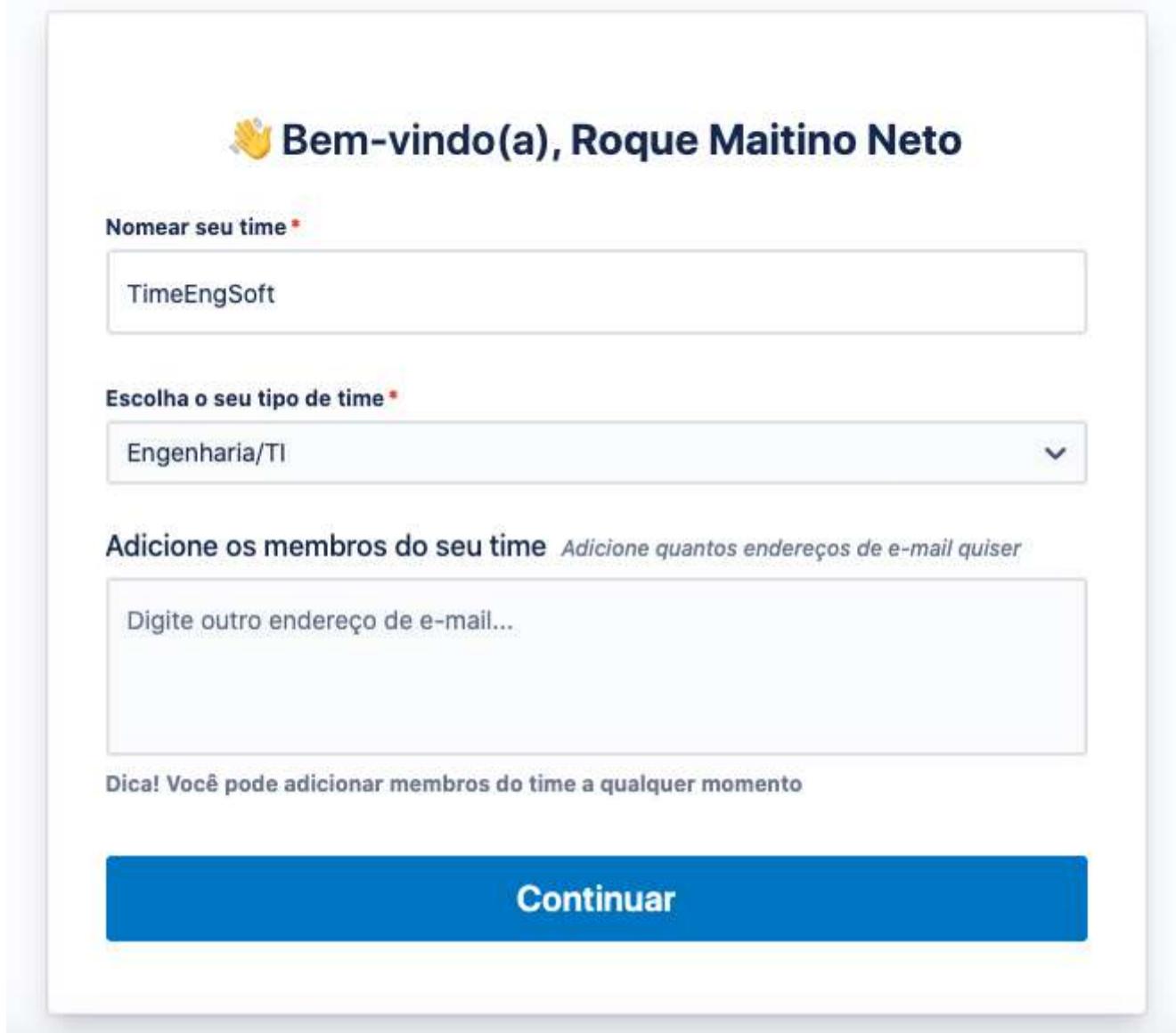
Figura 1.5 | Exemplo de quadro Scrum

PROJETO/EQUIPE: EQUIPE SCRUM MARAVILHOSA					
	Pendência	A fazer	Fazendo	Em revisão/garantia de qualidade	Feito!
História do usuário 1	[square]	[square] [square] [square]	[square]	[square] [square]	[square]
História do usuário 2		[square] [square] [square]	[square] [square]	[square] [square]	
História do usuário 3	[square]	[square] [square]	[square]	[square] [square]	[square]
História do usuário 4		[square]	[square]	[square] [square]	[square] [square]

Fonte: Sutherland (2014, p. 127).

Bem, mas será que podemos contar apenas com pedaços de papel autocolantes para registrar nossas tarefas no Scrum? Embora os post-its sejam úteis para a visualização, digamos, off-line do andamento do projeto, dispomos de ferramentas computacionais colaborativas que facilitam bastante o controle de um projeto, e a primeira que merece menção é o Trello, cujo acesso é gratuito. O início do cadastramento da sua conta acontece quando você fornece seu endereço de e-mail. Em seguida, você já pode dar um nome ao seu time, escolher o tipo do seu time e adicionar membros a ele, conforme exibido na Figura 1.6.

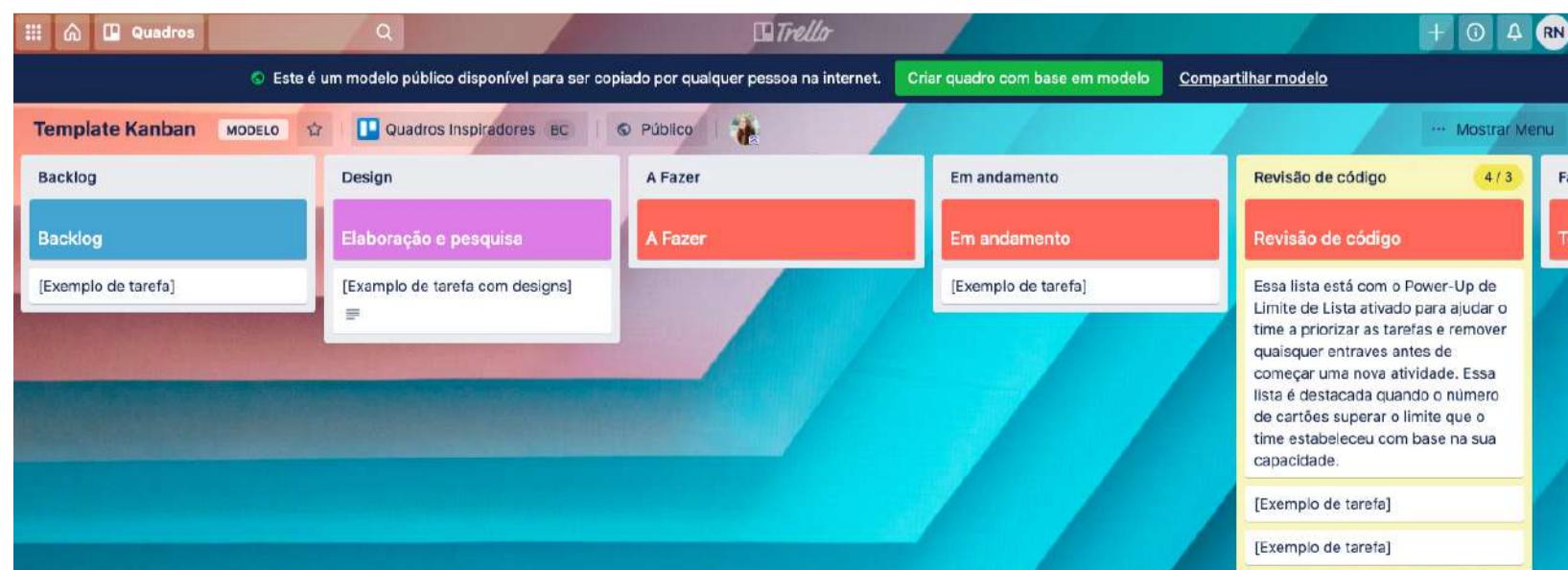
Figura 1.6 | Ambiente de cadastramento de conta no Trello



Fonte: captura de tela do Trello elaborada pelo autor.

Depois de fornecer essas informações, você será levado a um conjunto de opções e, dentre elas, poderá escolher um modelo de quadro em que as tarefas (e demais informações) serão exibidas. Se você escolher o *Template Kanban*, obterá um quadro semelhante ao exibido na Figura 1.7 De acordo com texto da autora do *template*, disponível na mesma página do modelo, ele deve ser usado para manter a equipe com o mesmo nível de informação, para que todos possam seguir o trabalho com fluidez.

Figura 1.7 | Template Kanban no Trello



Fonte: captura de tela do Trello de Alvernaz ([s.d.]).

Basta agora criar um quadro real com base nesse modelo e pronto! A partir daí é só inserir as tarefas e as informações relacionadas a elas para ter um controle eficiente do seu projeto Scrum. A fim de que você possa utilizar corretamente a ferramenta, vale um resumo de cada uma das colunas apresentadas no template:

- **Backlog**: aqui devem ser inseridas e descritas as tarefas do projeto, cada uma em seu cartão. A equipe deve ter a liberdade de inserir aqui tarefas que deverão ser, eventualmente, executadas no futuro, mas que ainda não devem ser movidas para a coluna “A Fazer”.
- **Elaboração e pesquisa**: nessa coluna a equipe deve colocar tarefas que precisam ser mais bem detalhadas e sobre as quais recai necessidade de pesquisa antes que sejam movidas para a próxima etapa. A ferramenta trata essa coluna como apropriada para atividades de projeto ou design, conforme assinalado no título da coluna.
- **A Fazer**: após a tarefa ser devidamente detalhada e um eventual aprofundamento ter sido feito, ela deve ser levada a essa coluna, como sinalização de que está pronta para ser executada. Nesse caso, um ou mais responsáveis devem ser atribuídos a ela e a data de entrega deve ser estipulada.
- **Em andamento**: quando as tarefas estão efetivamente em execução elas devem ocupar esta coluna. Dessa forma, todos os envolvidos poderão conferir o andamento das atividades executadas pelos pares e a ferramenta permite, inclusive, que mensagens sejam deixadas diretamente nas tarefas como forma de promover a comunicação.
- **Revisão de código**: no momento em que a tarefa estiver em vias de ser concluída, ela deve ser movida para esta coluna, a fim de que seja revisada por membro designado da equipe. Embora o título dela seja Revisão de Código, vale aqui qualquer tipo de conferência ou validação.

- **Feito:** assim que a tarefa é concluída, ela é movida para esta coluna, que não está completamente visível na Figura 1.7.

Essa foi, portanto, a exposição de conteúdo relacionado a metodologias ágeis. Tivemos a oportunidade de conhecer conceitos, práticas e, principalmente, de realizar comparações entre o novo e o tradicional. Em sua jornada profissional, você certamente encontrará organizações desenvolvedoras que um dia ousaram transformar seus processos e tornarem-se ágeis, outras que já foram concebidas como tais e, finalmente, outras tantas que preferiram continuar desenvolvendo programas seguindo padrões tradicionais. Em breve você será desafiado a decidir como a sua organização irá atuar e o embasamento aqui adquirido será fundamental. Continue focado nas leituras e nas atividades propostas!

FAÇA VALER A PENA

Questão 1

A ênfase no gerenciamento tradicional de projetos diz respeito à condução de um planejamento detalhado do projeto com ênfase na fixação do escopo, do custo e do cronograma e no gerenciamento desses parâmetros. O gerenciamento tradicional de projetos pode às vezes levar a uma situação em que o plano foi bem-sucedido, mas o cliente não está satisfeito.

Em relação às características dos modelos ágeis, analise as afirmações que seguem:

- I. Estimulam o desenvolvimento incremental e com intervalos curtos de feedback entre equipe e cliente.
- II. Foram criados com base nas ideias da produção linear, desenvolvidas para bens de consumo comuns.
- III. Apresentam determinismo e especialização de funções como marcas próprias.

IV. São mais bem adaptadas às mudanças de requisitos durante o projeto do que os modelos tradicionais.

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

a. I, II, III e IV.

b. I e IV apenas.

c. I, II e IV apenas.

d. III e IV apenas.

e. I, III e IV apenas.

Questão 2

Para serem eficazes, as equipes Scrum devem ter de seis a dez membros. Essa prática pode ser a razão para o equívoco de que o Framework Scrum só pode ser usado para pequenos projetos. No entanto, a equipe pode ser facilmente dimensionada para uso eficaz em grandes projetos, programas e portfólios.

Em relação ao framework Scrum, analise as afirmações que seguem.

I. O artefato conhecido como Product Backlog contém as

F funcionalidades a serem desenvolvidas durante a Sprint. Por ser associado à metodologia tradicional, esse artefato tem caído em desuso.

F II. Scrum Master é outro nome que se dá à equipe de desenvolvimento do Scrum, a qual é concebida para atuar de forma autônoma.

V III. Ao Product Owner compete, entre outras tarefas, a indicação da ordem de criação das funcionalidades do produto.

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

a. I e II apenas.

b. II e III apenas.

c. III apenas.

d. I e III apenas.

e. II apenas.

Questão 3

O Scrum é um dos métodos Agile mais populares. É uma estrutura adaptativa, iterativa, rápida, flexível e eficaz projetada para entregar valor significativo de forma rápida e durante todo o projeto. O Scrum garante transparência na comunicação e cria um ambiente de responsabilidade coletiva e de progresso contínuo.

No contexto do modelo Scrum, preencha as lacunas a seguir.

- I. O _____ contém as funcionalidades que compõem o sistema.
- II. O _____ responde pelo projeto e tem a responsabilidade de indicar os itens que compõem a lista de requisitos.
- III. O _____ é a equipe de desenvolvimento. Nela, não existe necessariamente divisão funcional em papéis tradicionais

Assinale a alternativa cujos termos completam corretamente as lacunas nas frases anteriores.

a. Product Backlog, Product Owner, Scrum Team.

b. Rol de requisitos, Scrum Team, conjunto de desenvolvedores.

c. Sprint Backlog, sponsor, Scrum Team.

d. Levantamento de requisitos, sponsor, Scrum Master.

e. Product Backlog, Scrum Team, Scrum Master.

REFERÊNCIAS

A GUIDE to the Scrum Body of Knowledge (SBOK GUIDE). 3. ed. [S.l.]: SCRUMstudy, 2015. Disponível em <https://apple.co/3nxZA3t>. Acesso em: 18 out. 2020.

ALVERNAZ, A. Template Kanban. **Trello**, [S.I., s.d.]. Disponível em: <https://bit.ly/3gYRRZD>. Acesso em: 12 dez. 2020.

BECK, K. et al. Manifesto para Desenvolvimento Ágil de Software. **Agile Manifesto**, [S.I.], 2001. Disponível em: <https://bit.ly/37tc1lb>. Acesso em: 29 out. 2020.

CRUZ, F. **Scrum e Agile em Projetos**. Guia Completo. 2. ed. Rio de Janeiro: Brasport, 2018.

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

SABBAGH, R. **Gestão Ágil para Projetos de Sucesso**. Edição Eletrônica: Casa do Código, 2013.

SILVA, E. C. **Scrum e FTS**: uma abordagem prática. Rio de Janeiro: Brasport Livros e Multimídia, 2017. Disponível em: <https://bit.ly/38d2kwO>. Acesso em: 29 out. 2020.

SUTHERLAND, J. **A Arte de Fazer o Dobro do Trabalho na Metade do Tempo**. São Paulo: LeYa, 2014.

TELES, V. M. **Extreme Programming**: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec Editora, 2006.

WAZLAWICK, R. S. **Engenharia de software**: conceitos e práticas. Rio de Janeiro: Elsevier, 2013.

FOCO NO MERCADO DE TRABALHO

METODOLOGIAS ÁGEIS

Roque Maitino Neto

Ver anotações

UTILIZAÇÃO DO SCRUM

No Scrum, o trabalho de um projeto é dividido em Sprints, pequenos ciclos de trabalho que geram pequenas entregas. Cada ciclo começa com o Product backlog (necessidades) e na Planning são selecionadas as tarefas da Sprint backlog. Na Daily (reunião diária curta) é discutido o progresso do projeto. Ao final do ciclo há então o incremento do produto.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

SEM MEDO DE ERRAR

- Um Product Backlog equivale a uma lista ordenada de elementos que o Product Owner considera necessários para o projeto do novo

produto. A ordem a ser seguida é a de prioridade desses elementos ou requisitos mais especificamente. No entanto, conforme já mencionado, a priorização não é estática e pode ser alterada segundo o grau de importância de cada requisito, conforme o negócio vai sendo alterado. Assim, uma possível solução para desenvolvimento de um Backlog inicial é a que segue:

Id	Requisito	Prioridade
1	O site deve apresentar a empresa e colocar sua marca em evidência, além de exibir sua missão, visão e valores, logo na página inicial e em posição de destaque.	Alta
2	O site deve apresentar os produtos comercializados pela empresa, com foto principal e breve descrição de cada um deles. No entanto, não há ainda necessidade de desenvolver meio para a realização de compra do produto.	Alta
3	O site deve permitir o cadastramento do usuário para conceder-lhe acesso a conteúdo exclusivo.	Média
4	O site deve requerer dados de contato do usuário para posteriores ações de marketing dirigidas a esse usuário.	Média
5	O site deverá exibir fotos secundárias de cada produto.	Baixa

Observe que os requisitos posicionados nas primeiras linhas possuem prioridade mais alta e descrição ligeiramente mais detalhada. Os requisitos posicionados em linhas mais abaixo apresentam menores graus de prioridade e descrição menos detalhada.

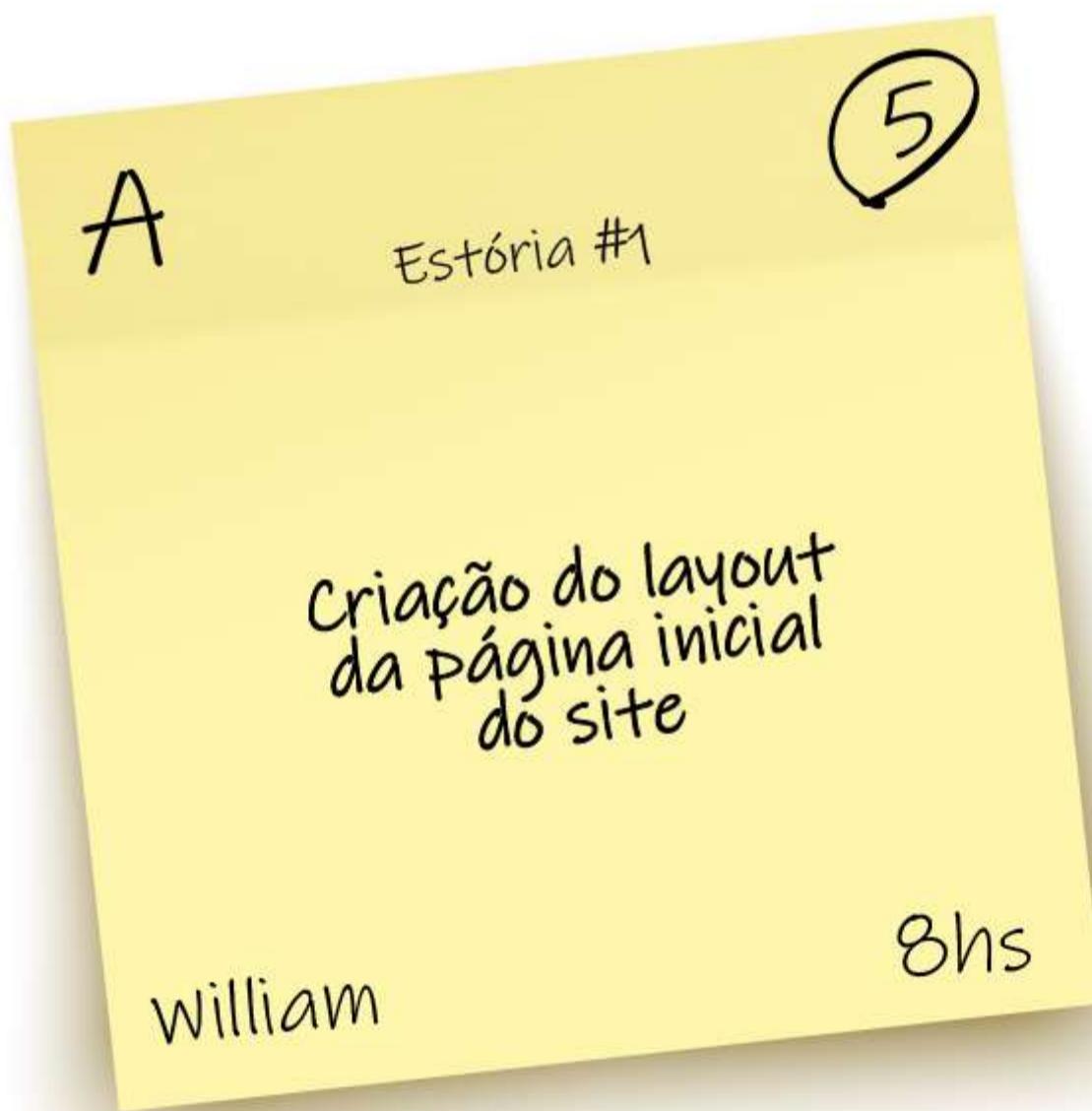
- Estória #1: O site deve apresentar a empresa e colocar sua marca em evidência, além de exibir sua missão, visão e valores, logo na página inicial e em posição de destaque.

Tarefas:

- Criar o layout da página inicial do site.
- Fazer o tratamento das imagens da página inicial do site.
- Inserir texto de missão, visão e valores na página.

- Uma possível configuração do post-it da primeira das três tarefas é a que segue:

Figura 1.8 | Representação de um post-it que contém dados de uma estória



Fonte: elaborada pelo autor.

Se utilizarmos a ferramenta Trello para as finalidades dessa situação-problema, teremos outra possível solução, conforme segue:

Figura 1.9 | Visão geral da função de Kanban da ferramenta Trello

Ver anotações

Fonte: captura de tela do Trello elaborada pelo autor.

Observe na coluna de Backlog o título das tarefas. Na segunda coluna foi colocada a tarefa de criação de modelos de layout de página, a qual requer aprofundamento e pesquisa por parte dos envolvidos. Embora não tenha sido exibida na segunda coluna, a atividade de criação de rotina de cadastramento de usuário já foi movida para a terceira coluna (A Fazer) como sinalização de que ela está pronta para ser executada. Da mesma forma, a tarefa de obter imagens dos produtos foi movida para a coluna “Em andamento” e, nesse caso, já conta com responsáveis e prazos estabelecidos.

AVANÇANDO NA PRÁTICA

APRIMORANDO O CONTATO COM O CLIENTE

Os gestores de uma empresa desenvolvedora de software sentiram a necessidade de aprimorar o modelo que a empresa vem seguindo desde sua criação, de modo que é desejo desses gestores que haja completa integração do cliente ao processo de desenvolvimento. Assim, espera-se aprimorar a comunicação entre os atores do projeto e evitar que a equipe desenvolva as funcionalidades com base apenas no que imaginam terem escutado um dia do cliente. Presunções de entendimento do problema já causaram vários atritos com clientes e a consequente geração de retrabalho.

A metodologia atual, que é baseada no Modelo em Cascata, não prevê regularidade na comunicação com o cliente, cuja participação no projeto fica restrita às reuniões iniciais de requisitos. Sua missão, portanto, é a de sugerir formas facilmente realizáveis de aproximar o cliente do processo, tornando-o corresponsável pelo sucesso da empreitada.

o

RESOLUÇÃO



Um conjunto de providências possíveis para o caso é o que segue:

- Os gestores devem escolher um membro da equipe para promover e manter o contato com cliente. Na medida do possível, esse contato deve ser pessoal e estabelecido no ambiente de desenvolvimento. Um espaço único em que a equipe e o cliente tratem exclusivamente do projeto deve ser reservado. As metodologias ágeis dão o nome de *War Room* (ou sala de guerra) a esse espaço.
- As equipes devem promover revisões sucessivas, em partes pequenas do produto, com a presença do cliente. A ideia de que este deve revisar apenas a versão final do produto não se mostrou efetiva no decorrer dos anos.
- A fim de que o cliente atribua o devido valor àquelas funcionalidades que solicita, ele deve escrever, de próprio punho, o que deseja que o programa resolva.

Com essas providências simples, o cliente deverá entender que é parte importante e ativa do projeto.

Ver anotações

NÃO PODE FALTAR

CONTROLE DE VERSÕES

Roque Maitino Neto

Ver anotações 0

GERENCIAMENTO DA CONFIGURAÇÃO DE SOFTWARE (GCS)

A GCS é um conjunto de práticas que proporciona o controle de todo o processo de desenvolvimento de software, necessário pela complexidade envolvida no processo de desenvolvimento e pela grande quantidade de componentes de um software.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

PRATICAR PARA APRENDER

Desenvolver um sistema definitivamente não é trabalho que se faça sozinho. O grau de exigência mental, a complexidade das construções algorítmicas e a grande quantidade de elementos a serem considerados na construção de um produto de software autorizam (ou quase obrigam) a convocação de mão de obra variada para que o desafio seja levado a bom termo. As consequências dessa união de forças são naturalmente benéficas, mas exigem providências bem coordenadas para o correto gerenciamento da evolução das versões do produto, sob pena de perda de controle das alterações feitas nessas versões.

Por isso, esta seção se dedica a abordar o conteúdo teórico associado ao controle de versões de software, incluindo o papel de um repositório e o necessário controle de compartilhamento de itens. Além disso, tratado funcionamento das principais ferramentas que auxiliam no gerenciamento da configuração de um software.

Chegamos ao desafio final desta unidade e, para bem cumpri-lo, devemos resgatar algumas situações que se apresentaram durante os desafios anteriores. Você está lembrado que, mediante uma situação desfavorável, assumiu a missão de introduzir uma metodologia de desenvolvimento de software na empresa em que havia sido contratado. Depois, com essa metodologia já amadurecida, você tratou de liderar a implantação de um modelo ágil de trabalho. Embora essas ações tenham sido a razão do salto de qualidade na empresa, ainda faltava uma última providência: implantar controle efetivo das versões dos produtos de software desenvolvidos.

Apesar de estarem cientes da importância de um sistema de controle de versões, os membros das equipes e os demais sócios da empresa estavam reticentes quanto a sua aquisição, afinal, bem ou mal, as produções eram razoavelmente bem controladas e todos tinham em mente o que seus colegas estavam desenvolvendo. Além disso, os custos de aquisição e de implantação de uma ferramenta dessas poderiam ser altos, o que diminuiria consideravelmente o benefício de sua adoção.

Assim, o desafio que agora se apresenta a você é o de novamente convencer seus pares quanto à adoção de um recurso, desta vez de um sistema de controle de versões de software. E novamente o meio de que você se utilizará será o da informação, com o adicional de uma exibição prática da ferramenta GitHub de controle de versão.

Para que a exibição seja possível, você deve realizar a instalação e a configuração do GitHub em qualquer computador disponível e apto a recebê-lo. Além disso, o procedimento deverá incluir também a criação do seu primeiro repositório de arquivos. Cada passo dessa sua atividade deverá ser registrado com um texto explicativo e a respectiva tela da etapa da instalação, de modo que seu trabalho possa servir como material de referência caso precise executar as mesmas ações no futuro. Mão à obra!

o

Ver anotações

DICA

O domínio do conteúdo teórico relacionado ao controle de versões, mais a habilidade na utilização de ferramentas que viabilizam esse controle permitirá a você colocar em prática sua capacidade de colaborar em equipe para a construção de um produto de qualidade e bem gerenciado.

Bom estudo e continue com a gente!

CONCEITO-CHAVE

Um software é um produto em constante evolução. Da sua concepção até a entrega (e além dela), ele nasce, cresce e carece de manutenção ao longo de um ciclo de vida bem definido e conhecido. Na condição de um produto dinâmico, seu processo de construção e sua manutenção requerem gerenciamento contínuo em todos os aspectos de sua construção e, mesmo em um estado considerado apto para entrega, um software passa por diversas modificações, e cada uma delas tem como

resultado uma nova versão dele próprio. Por isso, um controle eficiente das revisões é necessário, o que é atingido por meio da utilização de uma ferramenta de controle de versões.

0

GERENCIAMENTO DE CONFIGURAÇÃO DO SOFTWARE

Embora o controle de versões seja o foco desta seção, será necessário contextualizá-lo em um tema mais abrangente, do qual ele faz parte como um elemento muito importante. Esse tema mais abrangente se chama Gerenciamento da Configuração do Software (GCS) que, de modo objetivo, é o meio pelo qual se proporciona controle a todo o processo de desenvolvimento de software (LEON, 2004). Esse controle se torna necessário, entre outros motivos, pela complexidade envolvida no processo de desenvolvimento e pela grande quantidade de componentes de um software.

A motivação maior para a criação e para a estruturação de uma disciplina que cuida da Gerência de Configuração de Software foi a necessidade de controlar as modificações pelas quais inevitavelmente passa um programa, o que é feito com o uso de métodos e de ferramentas e com o intuito de maximizar a produtividade e minimizar os erros cometidos durante a evolução.

A GCS é, portanto, um conjunto de práticas que controlam e notificam as inúmeras correções, extensões e adaptações aplicadas no software durante seu ciclo de vida, com o objetivo de assegurar um processo de desenvolvimento e de evolução organizado e passível de ser rastreado (DANTAS, 2009).

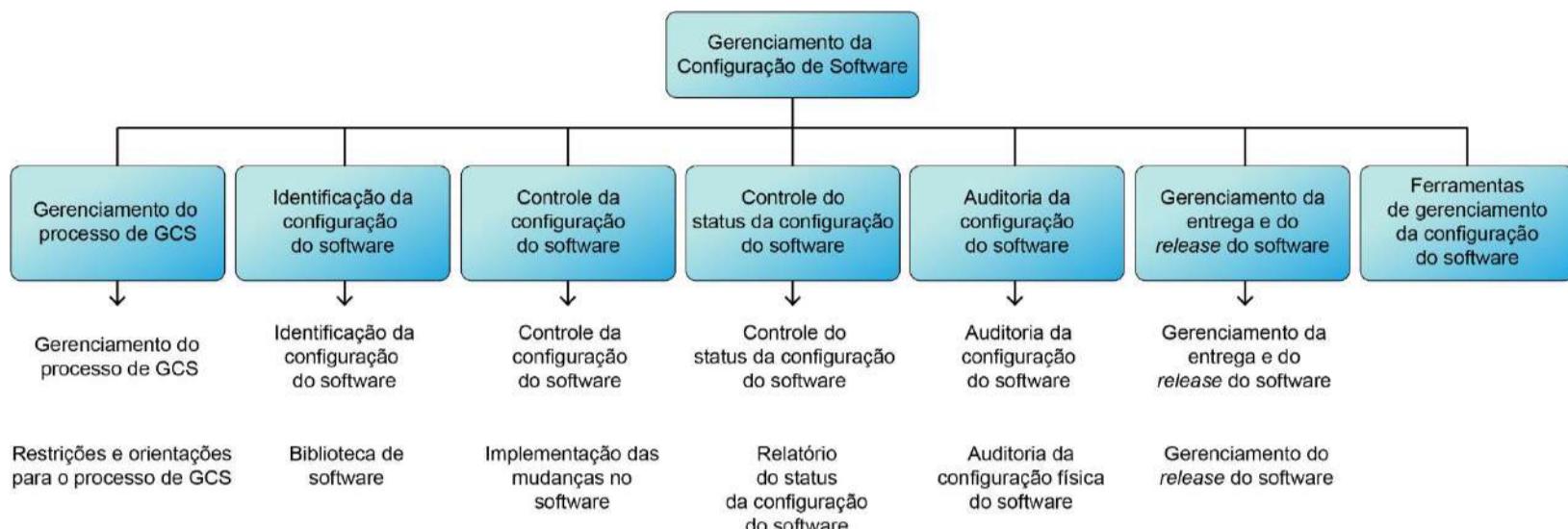
Como o GCS não se resume a uma ação apenas, ele é composto por uma série de atividades bem definidas e estruturadas. De acordo com IEEE (2004), essas atividades incluem: gerenciamento e planejamento do processo de GCS, identificação de configuração do software, controle de

Ver anotações

configuração de software, controle de status de configuração de software, auditoria de configuração de software, gerenciamento da entrega e do lançamento de software e configuração das ferramentas de gerenciamento. A Figura 1.10 ilustra essas atividades em uma estrutura hierárquica.

o

Figura 1.10 | Estrutura de tópicos do Gerenciamento da Configuração do Software



Fonte: adaptada de IEEE (2004).

Na sequência serão discutidas as três primeiras grandes atividades do SCM, colocadas nos retângulos da Figura 1.10, de acordo com IEEE (2004).

- **Gerenciamento do Processo de CGS:** conforme temos mencionado, o GCS controla a evolução e a integridade de um produto e o faz por meio da identificação dos seus elementos, pelo gerenciamento e controle das mudanças aplicadas e pela verificação, registro e relato das informações de configuração. Da perspectiva do engenheiro de software, o SCM facilita o desenvolvimento e as atividades de implementação de mudanças. Uma implementação de SCM bem-sucedida requer planejamento e gerenciamento cuidadosos, o que, por sua vez, requer compreensão do contexto organizacional e das restrições impostas ao projeto e à implementação do processo de SCM.
- **Identificação da configuração do software:** essa atividade identifica os itens a serem controlados, estabelece esquemas de identificação para os itens e suas versões e, por fim, estabelece as

- **Controle de configuração de software:** essa atividade trata do gerenciamento de mudanças durante o ciclo de vida do software. Ela abrange o procedimento para determinar quais mudanças fazer, a autoridade para aprovar certas mudanças, o suporte para a implementação dessas mudanças e o conceito de desvios formais dos requisitos do projeto.

0

Ver anotações

ASSIMILE

A Gerência de Configuração de Software é um processo aplicado a todas as fases que compõem o ciclo de vida de um software, estabelecendo regras formais para identificar e controlar mudanças por meio de um controle sistemático sobre modificações realizadas (CAETANO, 2004).

Mesmo com a evidente necessidade de se gerenciar a configuração de um software, essa prática sofre com alguns mitos, muitos deles com potencial para desencorajar sua adoção nas organizações. Nas próximas linhas trataremos de três desses mitos e daremos bons motivos para que eles sejam desconstruídos, sempre com base na visão de Leon (2004).

- **Mito 1 – Promover o gerenciamento da configuração de um software significa ter mais trabalho e adotar novos procedimentos:** de fato a implantação e o gerenciamento adequados de uma sistemática de GCS não é uma tarefa simples. O período de transição entre um ambiente sem gerenciamento e um cenário de efetiva utilização de uma ferramenta para esse fim costuma ser difícil, já que novas habilidades devem ser desenvolvidas, novos procedimentos devem ser seguidos, entre outros desafios. No entanto, a transição será tão mais simples quanto mais eficiente for o trabalho das equipes de gerenciamento e

de implementação do GCS. Ao utilizarem o sistema, os desenvolvedores e outros atores do processo de criação do software deverão compreender seus potenciais benefícios e o esforço que conseguirão evitar por meio da automação de tarefas e da eliminação de erros. Atualmente, as ferramentas de gerenciamento de configuração de um software automatizam todas as tarefas repetitivas, facilitando assim a atuação do pessoal envolvido com o produto.

- **Mito 2 - O gerenciamento da configuração é de responsabilidade única do pessoal da administração do sistema:** ao contrário do que sugere o mito, a implantação e a condução do GCS é responsabilidade de todas as pessoas envolvidas no processo de desenvolvimento de software embora a principal tarefa do pessoal da administração seja criar um ambiente organizacional no qual o GCS possa prosperar. Uma equipe de GCS precisa de todo o apoio do pessoal da administração para ter condições de implantar o sistema de gerenciamento da configuração aos poucos, pois é esse pessoal que deve monitorar a implementação e a operação do GCS, revisar periodicamente seu progresso e tomar as ações corretivas quando necessário.
- **Mito 3 - O gerenciamento da configuração é apenas para os desenvolvedores:** certamente as equipes de desenvolvedores constituem um dos mais frequentes usuários do GCS. Além disso, são os desenvolvedores que mais se beneficiam de um sistema de gerenciamento corretamente implementado. Problemas como perda de código-fonte, incapacidade de encontrar a última versão de um arquivo e reaparecimento de erros já corrigidos podem ser evitados com um sistema desses. Entretanto, há desenvolvedores que enxergam o GCS como perda de tempo e dele participam apenas por serem obrigados a isso. A potencial hostilidade voltada ao GCS pode ser eliminada se os desenvolvedores forem educados nos princípios da prática e esclarecidos sobre seus benefícios. As

ferramentas atuais de GCS proporcionam um nível fenomenal de automação, o que acaba contribuindo para que outros atores do processo de desenvolvimento acabem se beneficiando delas, incluindo testadores e pessoal de qualidade, manutenção e suporte.

REFLITA

Com que frequência as equipes de desenvolvimento rejeitam ou boicotam um processo de gerenciamento de configuração de software? Bem, a resposta deverá variar conforme a familiaridade da equipe com as ferramentas de GCS, com a disposição a aceitar mudanças, entre outros fatores. Em certas equipes pode preponderar o sentimento de que tudo o que se refere ao software está bem registrado na memória de seus componentes ou que uma mera anotação em algum arquivo servirá como registro das mudanças efetivadas no produto. Com isso, a resistência a colocar em funcionamento um processo de GCS pode ser maior do que a eventual dificuldade técnica de implantá-lo?

CONTROLE DE VERSÕES

Feita a contextualização do Gerenciamento de Configuração do Software com o ambiente em que o controle de versões está inserido, avançamos para a abordagem desse tema. De acordo com Caetano (2004), o controle de versões é o meio pelo qual o GCS controla, de forma consistente, as modificações realizadas em um sistema. Isso ocorre por meio das seguintes funções:

- Recuperar versões anteriores do sistema.
- Auditar as modificações realizadas, levantando quem alterou, quando alterou e o que alterou.
- Automatizar o rastreamento de arquivos.
- Estabelecer meios para obter a situação de um projeto em determinado ponto do tempo.

- Prevenir conflitos entre desenvolvedores.
- Permitir o desenvolvimento paralelo de um ou mais sistemas.

REPOSITÓRIO

Todas essas funções parecem convergir para um elemento bem definido: a centralização dos dados. E se não tivéssemos essa centralização? Bem, o uso descentralizado de um arquivo de código-fonte, por exemplo, permitiria que vários programadores acessassem vários arquivos e cada alteração feita nele não seria refletida nos demais. Assim, o desenvolvimento paralelo seria inviável e o conflito entre desenvolvedores inevitável. O recurso que as ferramentas de controle de versão (abordadas a seguir) usam é o repositório, local em que programas em desenvolvimento, fotos, vídeos e demais arquivos ficam armazenados e podem ser acessados de forma controlada por todos os envolvidos no desenvolvimento do produto.

BASELINES (LINHAS DE BASE)

Um conceito importante no escopo do controle de versão é a *baseline* de software. As *baselines* representam conjuntos de itens de configuração formalmente aprovados, os quais servem de base para as etapas seguintes de desenvolvimento. Quando, no entanto, uma entrega formal é feita ao cliente no final de uma iteração, denominamos tal entrega de release. *Baselines* e releases são identificadas no repositório de programas, na grande maioria das vezes, pelo uso de etiquetas (*tags*) (DANTAS, 2009).

O termo também é usado para se referir a uma versão específica de um item de configuração de software que foi acordado e, em qualquer caso, a *baseline* só pode ser alterada por meio de procedimentos formais de controle de mudanças. Uma *baseline*, junto com todas as alterações aprovadas na linha de base, representa a configuração aprovada atual.

As comumente usadas são as linhas de base funcionais, alocadas, de desenvolvimento e de produto. A linha de base funcional corresponde aos requisitos de sistema revisados. A linha de base alocada corresponde à especificação de requisitos de software revisada e à especificação de requisitos de interface de software. A linha de base de desenvolvimento representa a configuração de um software em evolução, em momentos selecionados, durante o ciclo de vida do software. A autoridade de mudança para essa linha de base normalmente é da organização de desenvolvimento, mas pode ser compartilhada com outras organizações. A linha de base do produto corresponde ao produto de software completo e entregue para integração de sistema (IEEE, 2004).

BRANCHES

A Gerência de Configuração de Software também permite que a implementação de novas funcionalidades por uma equipe seja realizada em paralelo, mas de forma isolada e independente das modificações de outros desenvolvedores. O isolamento é obtido com uso de ramificações (*branches*). As linhas de desenvolvimento (*code-lines*) são designadas para cada projeto e são compartilhadas por vários desenvolvedores. A primeira linha de desenvolvimento definida no projeto é, por convenção, nomeada *mainline*. O ramo é criado no repositório e representa uma linha secundária de desenvolvimento, a qual pode ser unida novamente à linha principal (*mainline*) por meio da operação de junção (merge). Atualmente, a necessidade de atender, ao mesmo tempo, as múltiplas demandas do projeto tornam o uso de ramos um diferencial (DANTAS, 2009).

FERRAMENTAS DE CONTROLE DE VERSÃO

Como era de se esperar, as funções próprias do controle de versões podem (e devem) ser executadas por uma ferramenta computacional, fato que naturalmente apresenta indiscutíveis vantagens em relação a uma eventual execução manual. O mercado disponibiliza ótimas

ferramentas de controle de versão e três delas serão abordadas na sequência. Os critérios para adoção da ferramenta mais adequada deverão variar entre organizações, mas vale o alerta de que nenhuma decisão deve ser tomada com base na convicção de que ela fará as vezes de um compilador, de que substituirá o gerente do projeto, ou de que ela pode se tornar um meio de automatizar testes. Passemos, então, à discussão de duas das mais importantes e utilizadas ferramentas de controle de versão.

Ver anotações

GIT E GITHUB

Nossa primeira providência, ao tratarmos das ferramentas Git e GitHub, será a de diferenciá-las. Apesar de terem nomes bastante parecidos e de serem produtos do mesmo desenvolvedor, suas funções são distintas: o **Git** é uma ferramenta de controle de versão bastante popular entre desenvolvedores e apresenta recursos de colaboração bastante aprimorados, incluindo fóruns de discussão para os projetos em desenvolvimento e para abordagem das alterações em curso. Presente também em outras ferramentas de controle de versão, os *branches* implementados no Git viabilizam o trabalho em paralelo entre membros da equipe e o controle de subprojetos que um desenvolvedor pode manter para experimentos próprios, incluindo correção de bugs e aprimoramento de funções, sem que os arquivos do repositório central sejam alterados e com a possibilidade de que seus experimentos sejam compartilhados (MAILUND, 2017).

O Git é um sistema de controle de versão gratuito e de código aberto, concebido para lidar com todos os projetos, desde os pequenos até aqueles bem grandes e que movimentam praticamente toda as equipes. Ele é bem fácil de ser aprendido e ocupará pouco espaço do servidor. Além disso, seu desempenho é bastante satisfatório. Mas e a segurança? Bem, o modelo de dados que o Git usa garantirá a integridade criptográfica de cada bit dos projetos. Cada arquivo e cada

operação de *commit* passa por verificação de *checksum*, operação também aplicada em sua recuperação. Será impossível extrair qualquer coisa do Git além dos bits exatos que foram adicionados.

O recurso do Git que realmente o diferencia de quase todos os outros sistemas de controle de versão existentes é seu modelo de *branching* (ou ramificação). Com o Git, é possível ter vários *branches* nas máquinas locais, os quais podem ser independentes uns dos outros. As operações de criação, merge e exclusão dos *branches* leva alguns poucos segundos apenas. Com esse diferencial, as equipes poderão trocar de contexto quase que imediatamente, além de criar *branches* para experimentar uma ideia e, mesmo aplicando a operação de *commit*, voltar ao ponto de onde começou. O mesmo cabe para a aplicação de um patch, por exemplo.

O **GitHub** é equivalente ao repositório do Git, mas disponível em uma plataforma mundial e com acesso gratuito aos desenvolvedores que lá desejarem armazenar seus programas e compartilhá-los com outros desenvolvedores. Por meio da página de cadastro (GITHUB, c2020) é possível criar sua conta no GitHub e começar a fazer parte dessa grande comunidade. Bem, mas o que é possível obter com a criação de uma conta no GitHub além de um espaço para compartilhar código? A resposta mais simples vem com apenas uma palavra: visibilidade. Empresas de TI já consideram o repositório do candidato no GitHub como um dos critérios para sua contratação. Tamanha importância justifica nossa incursão por essa ferramenta, começando pelo passo a passo de cadastramento na plataforma.

1. Acesse a página de cadastro (GITHUB, c2020) e preencha os espaços com os dados solicitados.
2. Após a escolha de sua senha e da verificação da conta, a plataforma o levará para uma página em que você poderá escolher, nesta ordem:
 - Seu tipo principal de trabalho

- Seu nível de experiência em programação.
- Qual o uso que você pretende dar ao GitHub.
- Qual seu interesse (para que a ferramenta possa conectá-lo a comunidades com as mesmas afinidades que a sua).

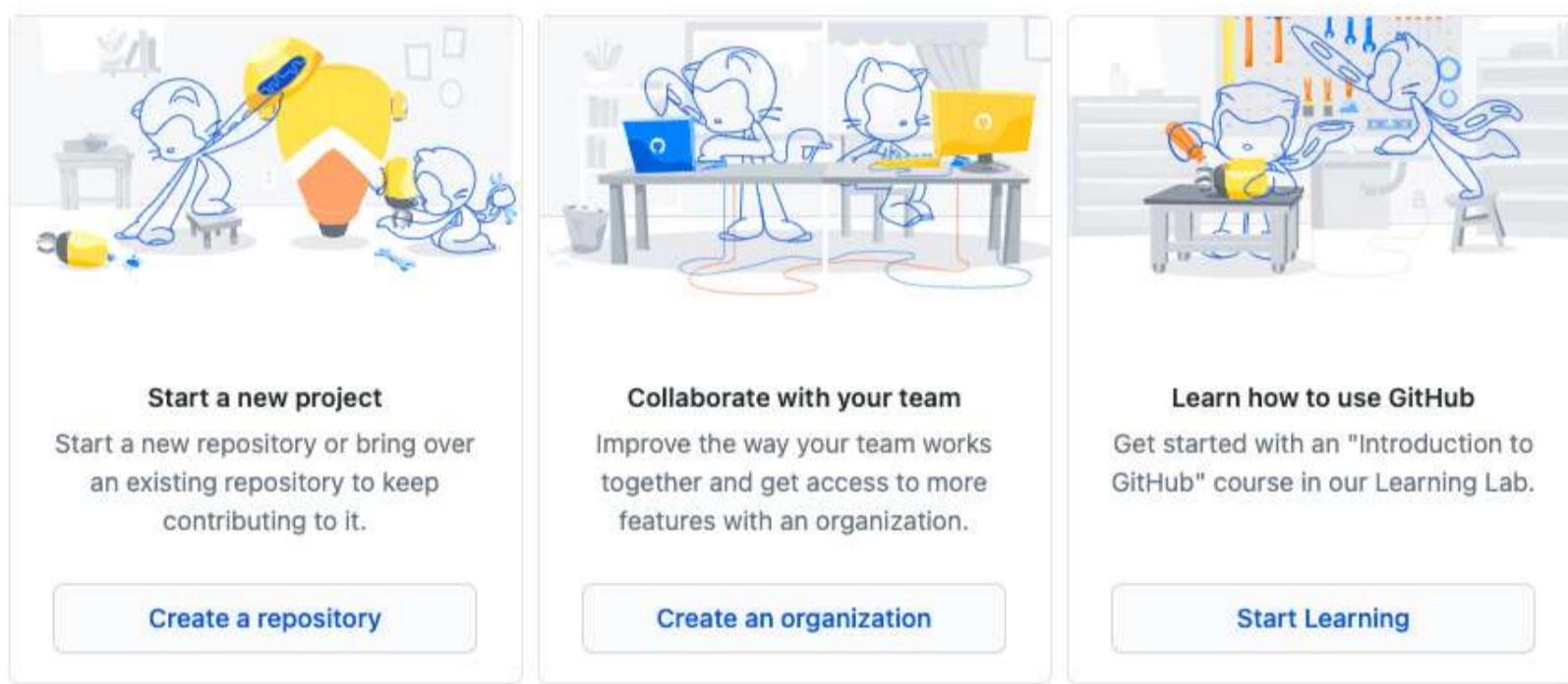
Para essas opções, nossas escolhas foram: estudante, pequena experiência em programação, interesse em aprender o Git e o GitHub e resposta em branco, respectivamente.

3. Após a confirmação do seu endereço de e-mail, você será direcionado à tela em que poderá escolher o que deseja fazer primeiro: criar um novo repositório (ou um novo projeto), criar uma organização ou começar a aprender, conforme mostra a Figura 1.11.

Figura 1.11 | Tela de escolha da ação inicial no GitHub

What do you want to do first?

Every developer needs to configure their environment, so let's get your GitHub experience optimized for you.

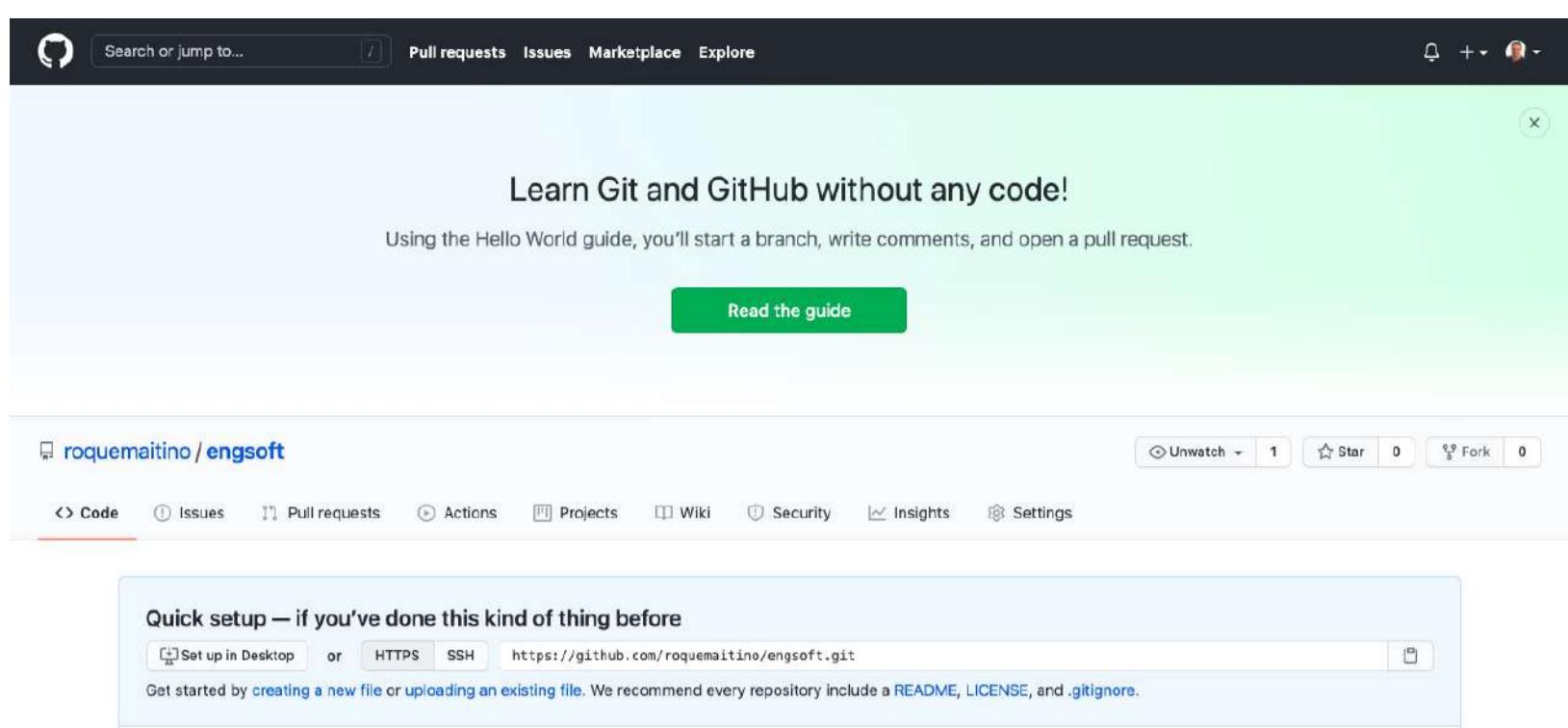


Fonte: captura de tela do GitHub.

4. Ao criar um novo projeto, a primeira ação a ser tomada é a escolha de um nome para o repositório, que estará atrelado ao nome de usuário que você escolheu no passo inicial. O nome escolhido para o repositório foi *engsoft* e a visibilidade escolhida foi a pública. Nenhuma opção de inicialização foi escolhida.

5. Neste ponto, uma tela com uma série de opções será oferecida, conforme ilustra a Figura 1.12.

Figura 1.12 | Visão parcial da tela de opções do GitHub



Fonte: captura de tela do GitHub elaborada pelo autor.

A partir deste ponto, você pode criar seus arquivos de código ou fazer upload de um já existente em sua máquina.

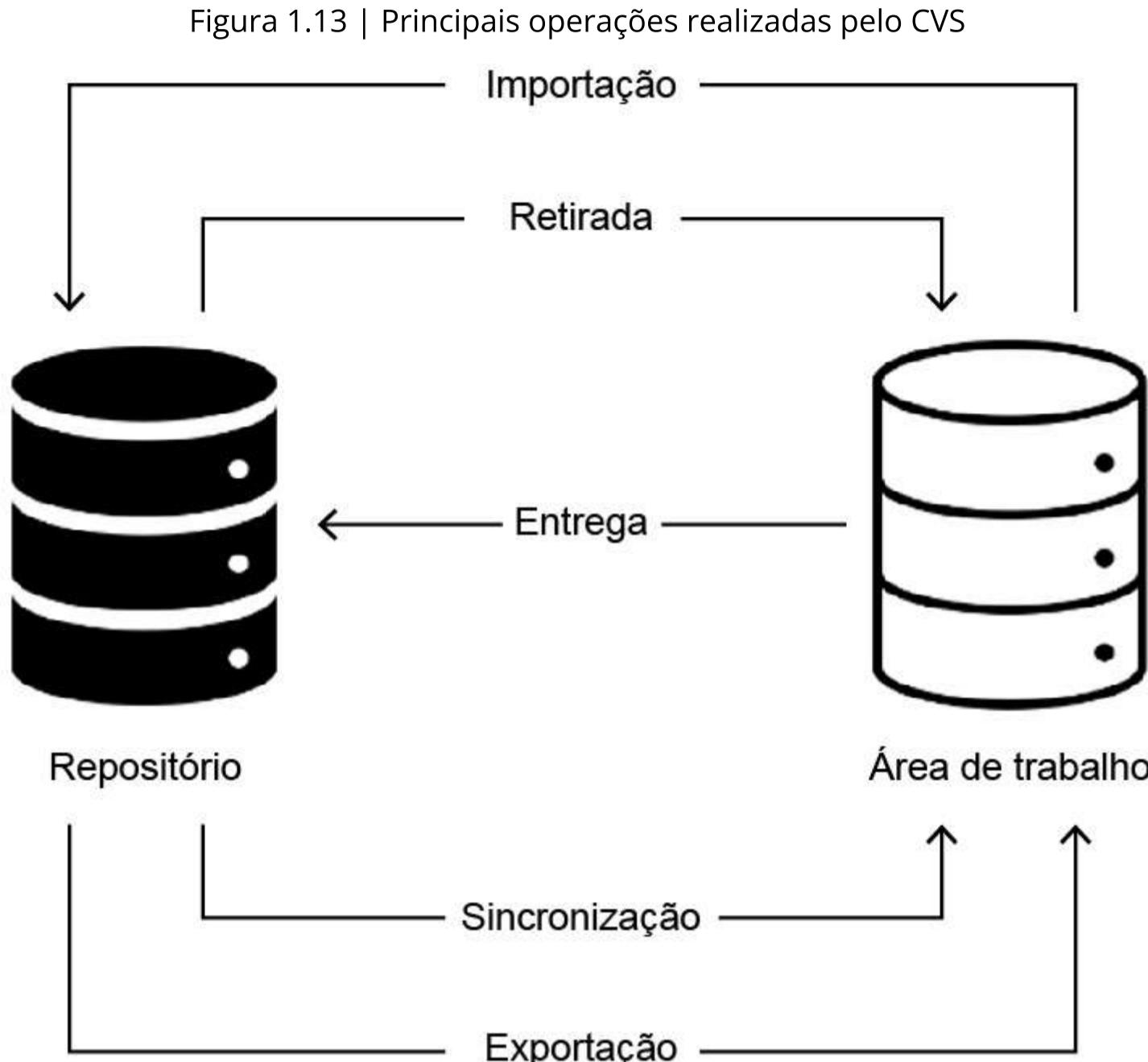
Note o nome do usuário e o nome do repositório (*roquemaitino/engsoft*) na porção central e à esquerda da tela. Para que um arquivo estivesse disponível no repositório, foi feita a escolha de upload de um arquivo existente e, na sequência, acionado o botão de *Commit*. O arquivo escolhido foi *Maior.java*, uma aplicação simples em Java, que exibe o maior valor entre os oito informados pelo usuário via teclado.

Pronto! Ao acessar a página inicial desse perfil (MAITINO, 2020), você poderá ter acesso também à aplicação Java que disponibilizamos e alterar seu código por meio da criação de um branch.

CVS (CONCURRENT VERSION SYSTEM OU SISTEMA DE VERSÕES CONCORRENTES)

Trata-se de uma ferramenta *open source* que implementa as principais funções relacionadas ao processo de controle de versões. O CVS armazena em seu repositório as modificações realizadas num arquivo ao longo do tempo. Cada modificação é identificada por um número chamado revisão. Toda revisão armazena as modificações realizadas,

quem realizou as modificações, quando foram realizadas, entre outras informações (CAETANO, 2004). A Figura 1.13 ilustra as principais funções realizadas pelo CVS.



Fonte: adaptado de Caetano (2004, p. 15).

O repositório CVS – assim como o de outras ferramentas – armazena uma cópia completa de todos os arquivos e diretórios que estão sob controle de versão. Normalmente, o desenvolvedor nunca acessa nenhum dos arquivos no repositório diretamente. Em vez disso, usa comandos CVS para obter sua própria cópia dos arquivos em uma área de trabalho e, em seguida, trabalha nessa cópia. Quando termina um conjunto de alterações, realiza uma operação chamada ***commit*** (ou confirmação) de volta para o repositório. Por isso, o repositório guarda as mudanças feitas pelo desenvolvedor, além de registrar exatamente o que e quando foi alterado e outras informações semelhantes. Observe que o repositório não é um subdiretório da área de trabalho ou vice-versa e que eles devem estar em locais separados (GNU, [s.d.]).

A estrutura geral do repositório é uma árvore de diretórios correspondente aos existentes no diretório de trabalho. Por exemplo, supondo que o repositório esteja em `/usr/local/cvsroot`, uma possível estrutura de diretório é mostrada na Figura 1.14.

Figura 1.14 | Uma possível estrutura de diretórios do repositório CVS

```
/usr
|
+--local
|
|   +--cvsroot
|
|   |   |
|   |   +--CVSROOT
|   |       (administrative files)
|
|   +--gnu
|
|   |   |
|   |   +--diff
|   |       (source code to GNU diff)
|
|   |   +--rcs
|   |       (source code to RCS)
|
|   |   +--cvs
|   |       (source code to CVS)
|
+--yoyodyne
|
|   +--tc
|
|   |   |
|   |   +--man
|
|   |   +--testing
|
+--(other Yoyodyne software)
```

Fonte: GNU ([s.d.]).

Na estrutura de diretórios estão os arquivos de histórico de cada arquivo sob controle de versão. O nome do arquivo de histórico é o nome do arquivo correspondente com ', v' anexado ao final. Os arquivos "`index.php,v`" e "`frontend.c,v`" são exemplos possíveis de arquivos de históricos. Eles guardam, entre outras coisas, informações suficientes para recriar qualquer revisão do arquivo, mais um log de todas as

mensagens de *commit* e o nome do usuário que enviou a revisão. Os arquivos de histórico são conhecidos como arquivos RCS, porque o primeiro programa a armazenar arquivos nesse formato foi um sistema de controle de versão conhecido como RCS (GNU, [s.d.]).

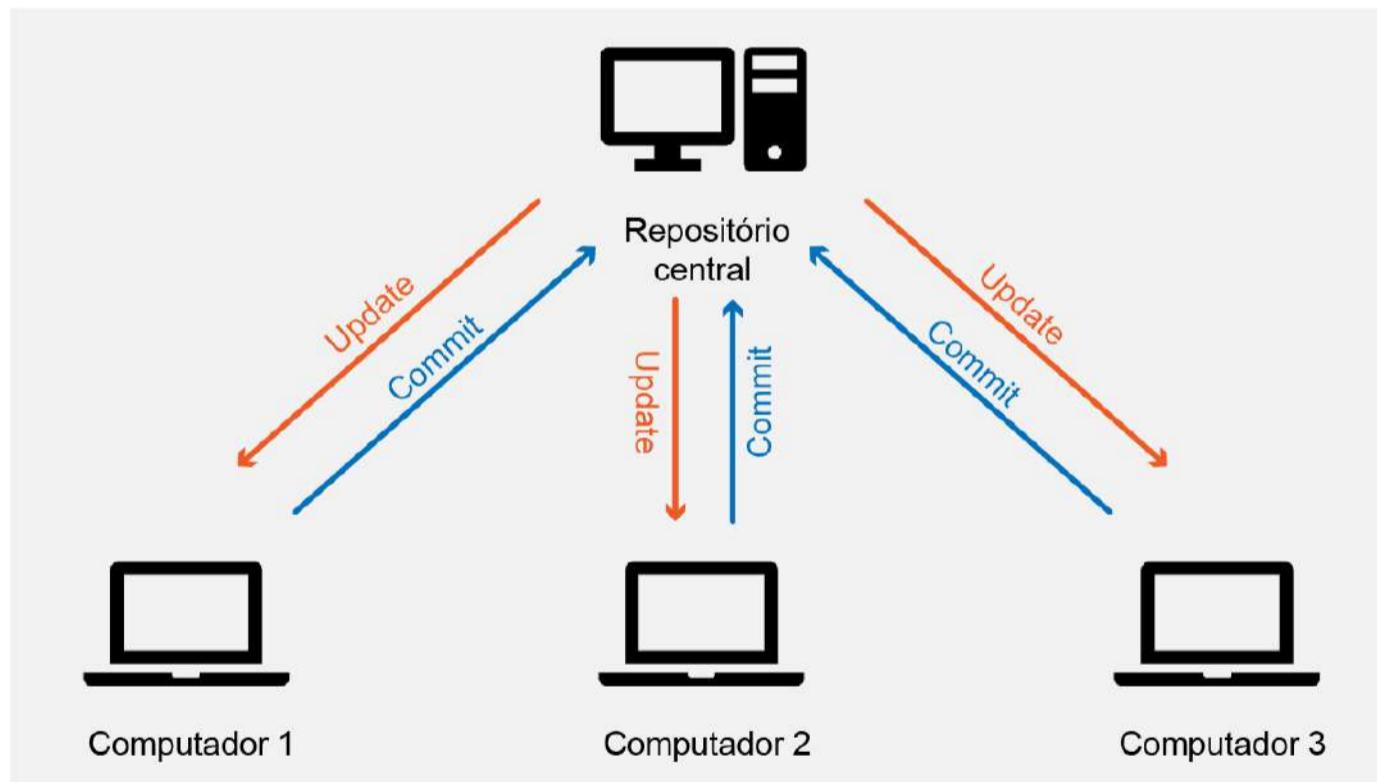
o

EXEMPLIFICANDO

Conforme ilustrado na figura a seguir, vários desenvolvedores podem trabalhar de forma concorrente em um mesmo sistema. Como alternativa, é possível atuar isoladamente em sistemas diferentes. Este exemplo de operação do CVS inclui um repositório central e três usuários, cada um com sua cópia de trabalho. As transições de comandos *update* (ou *check-out*) e *commit* (ou *check-in*) também estão representadas na Figura 1.15.

Ver anotações

Figura 1.15 | Exemplo de funcionamento do CVS



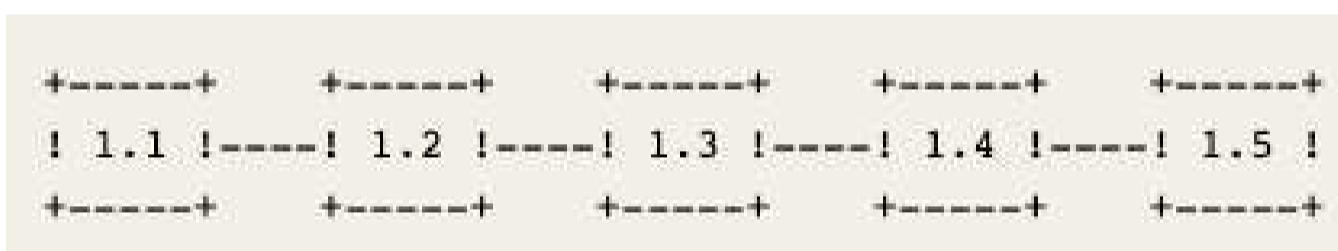
Fonte: elaborada do autor.

Cada alteração feita no programa gera um novo número de versão que o identifica. O CVS atribui automaticamente números como 1.1, 1.2 e assim por diante para as versões geradas. Um arquivo pode ter várias versões e, da mesma forma, um produto de software pode ter várias versões. Esse produto geralmente recebe um número de versão como

“4.1.1”. Cada versão de um arquivo possui um número de revisão exclusivo. Os números de revisão são semelhantes a “1.1”, “1.2”, “1.3.2.2” ou mesmo “1.3.2.2.4.5”.

Um número de revisão sempre tem um número par de inteiros decimais separados por ponto. Por padronização, a revisão 1.1 é a primeira de um arquivo. Cada uma delas recebe sucessivamente um novo número, aumentando o número mais à direita em um. A Figura 1.16 exibe algumas revisões, com as mais recentes à direita. Também é possível terminar com números contendo mais de um ponto, por exemplo “1.3.2.2”.

Figura 1.16 | Representação de revisões mais recentes no número à direita



Fonte: GNU ([s.d.]).

Antes de encerrarmos o conteúdo do CVS, vale tratarmos de mais algumas terminologias relacionadas ao assunto (GNU, [s.d.]).

- **Checkout:** denominação dada à primeira recuperação (ou download) de um módulo do sistema vindo do repositório do CVS.
- **Commit:** trata-se do envio do artefato modificado ao repositório do CVS.
- **Export (ou Exportação):** trata-se da recuperação (ou download) de um módulo inteiro a partir de um repositório, sem os arquivos administrativos CVS. Módulos exportados não ficam sob controle do CVS.
- **Import (ou Importação):** esse termo identifica a criação de um módulo completo no âmbito de um repositório CVS, feita por meio de um upload de uma estrutura de diretórios.

- **Module (ou módulo)**: é a representação de uma hierarquia de diretórios. Um projeto de determinado software efetiva-se como um módulo dentro do repositório.
- **Release**: este termo equivale a um “lançamento”. Um número de release identifica a versão de um produto completo ou inteiro.

ASSIMILE

Release é o termo usado para descrever a denominação atribuída a um conjunto de arquivos para identificar determinado ponto no tempo, sobretudo quando se quer identificar um conjunto de novas características ou correções de um software (CAETANO, 2004).

- **Merge**: refere-se à fusão das diversas modificações feitas por diferentes usuários na cópia local de um mesmo arquivo. Sempre que alguém altera o código, é necessário realizar uma atualização antes da aplicação da operação de *commit*, a fim de que seja feita a fusão das mudanças.

Este foi, portanto, o conteúdo que queríamos compartilhar com você. O entendimento de da importância do gerenciamento da configuração de um software e a familiaridade com os termos e o funcionamento de uma ferramenta de controle de versões são habilidades imprescindíveis para o desenvolvedor inserido em uma equipe de trabalho e ao gestor do software. Esperamos que este conteúdo seja útil a você em breve.

Boa sorte e bons estudos!

SAIBA MAIS

Caro estudante, a Engenharia de Software Sustentável é uma disciplina emergente na interseção de ciência do clima, software, hardware, mercados de eletricidade e design de data center. Os princípios da Engenharia de Software Sustentável são um conjunto básico de competências necessárias para definir, criar e executar aplicativos de

software sustentáveis. Aproveite este módulo para conhecer mais, embarque nessa jornada do conhecimento e bons estudos!

- [Os princípios da Engenharia de Software Sustentável - Learn | Microsoft Docs](#)



FAÇA VALER A PENA

Questão 1

Uma das mais importantes funções de um sistema de controle de versões é a de _____ as alterações feitas no sistema, através do levantamento de quem as efetivou e quando as fez. Outra função bastante conhecida é aquela que permite o desenvolvimento _____ de um sistema ou de vários deles. Por fim, uma ferramenta desse tipo permite também a criação de _____ do mesmo sistema.

Assinale a alternativa com os termos que preenchem corretamente as lacunas do texto.

a. auditar; paralelo; versões.

b. auditar; isolado; versões.

c. desconsiderar; paralelo; documentação.

d. auditar; isolado; versões.

e. afirmar; paralelo; cópias.

Questão 2

Como o GCS não se resume a uma ação apenas, ele é composto por uma série de atividades bem definidas e estruturadas. De acordo com IEEE (2004), essas atividades incluem: o gerenciamento e o planejamento do processo de GCS, a identificação de configuração do software, o controle de configuração de software, o controle de status

de configuração de software, a auditoria de configuração de software, o gerenciamento da entrega e do lançamento de software e a configuração das ferramentas de gerenciamento.

Considerando as características e atividades próprias do Gerenciamento da Configuração do Software, assinale a alternativa que expressa a área ou a atividade com que ele **guarda relacionamento estreito**.

a. Teste de software.

b. **Projeto de software.**

c. Segurança de dados.

d. Garantida da Qualidade do Software.

e. Desenvolvimento profissional.

Questão 3

A qualidade de um projeto de software é diretamente proporcional à qualidade dos processos adotados nas diversas fases do seu ciclo de vida. O controle de versões é visto como uma extensão natural do processo de desenvolvimento, permitindo que se possa paralelizar o desenvolvimento de forma coerente e padronizada, especialmente em se tratando de um conjunto muito grande de desenvolvedores (CAETANO, 2004).

Assinale a alternativa que, resumidamente, descreve o funcionamento **do CVS**, na ordem em que os eventos ocorrem.

a. Transferência dos arquivos da área de trabalho para o servidor, efetivação da operação de commit na área de trabalho, efetivação de alterações no código e efetivação das modificações necessárias.

b. **Importação dos arquivos para o repositório de arquivos no servidor, transferência dos arquivos do servidor para a área de trabalho, realização das modificações necessárias e submissão da versão modificada para o servidor.**

c. Aplicação das ações de copiar e colar na área de trabalho do arquivo a ser alterado, efetivação das modificações necessárias e aplicação das ações de copiar e colar o arquivo alterado no servidor.

d. Download do arquivo do servidor, transferência do arquivo para a área de trabalho, exportação da área de trabalho para o servidor e aplicação das modificações necessárias.

e. Importação dos arquivos para o repositório de arquivos na área de trabalho, realização das modificações necessárias, transferência dos arquivos do servidor para a área de trabalho e submissão da versão modificada para o servidor.

REFERÊNCIAS

CAETANO, C. **CVS** – Controle de Versões e Desenvolvimento Colaborativo de Software. São Paulo: Novatec Editora, 2004.

DANTAS, C. Gerência de Configuração de Software. **DevMedia**, [S.I.], 2009. Disponível em: <https://bit.ly/2LOcXP7>. Acesso em: 26 out. 2020.

GIT FOR WINDOWS. Git for Windows – Version 2.29.2.3. **Git for Windows**, [S.I.], 2020. Disponível em: <https://gitforwindows.org/>. Acesso em: 13 dez. 2020.

GITHUB. Join. **GitHub**, [S.I.], c2020. Disponível em: <https://github.com/join>. Acesso em: 12 dez. 2020.

GNU. The Repository. **GNU**, [S.I., s.d.]. Disponível em: <https://bit.ly/38incTg>. Acesso em: 27 out. 2020.

IEEE Computer Society. **Guide to the Software Engineering Body of Knowledge**. Piscataway: The Institute of Electrical and Electronic Engineers, 2004.

LEON, A. **Software Configuration Management Handbook**. 3. ed. Boston: Artech House, 2015.

LONGEN, A. S. Como Usar Um Git Branch. **Hostinger**, [S.I.], 23 abr. 2019. Disponível em: <https://bit.ly/3r8DGWC>. Acesso em: 28 nov. 2020.

MAILUND, T. **The Beginner's Guide to GitHub**. [S.I.: s.n.], 2017. E-book.

MAITINO, R. Roquemaitino/engsoft. **GitHub**, [S.I.], 2020. Disponível em: <https://bit.ly/3nJHqfr>. Acesso em: 12 dez. 2020.

FOCO NO MERCADO DE TRABALHO

CONTROLE DE VERSÕES

Roque Maitino Neto

Ver anotações

SISTEMA DE CONTROLE DE VERSÕES DE SOFTWARE

Implantação de controle efetivo das versões dos produtos de software desenvolvidos, criação do repositório de arquivos no GitHub.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

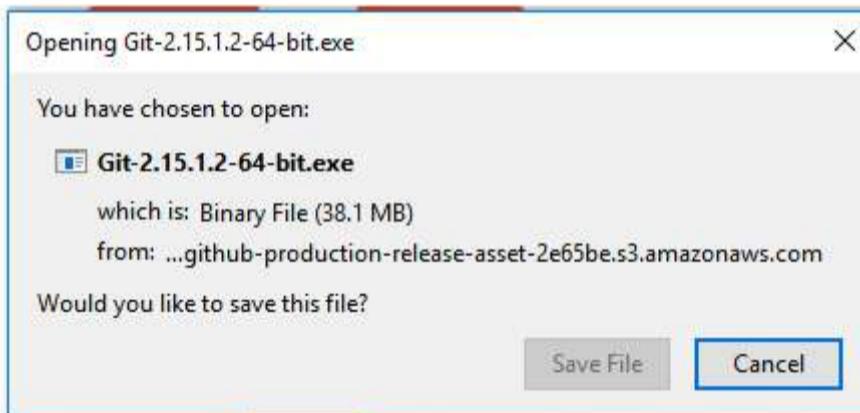
SEM MEDO DE ERRAR

Você já sabe, mas não custa lembrar: a informação é um meio poderoso para atingir o objetivo de convencer pares sobre a importância de se usar certa ferramenta, um sistema ou um processo, entre outros itens de uso comum em uma organização. Melhor ainda se ela estiver

acompanhada por uma demonstração prática desses elementos. Nesse sentido, a missão que se coloca aqui é a de realizar a instalação da ferramenta GitHub e registrá-la com imagens das telas e algum texto que explique cada passo do procedimento. Um possível registro dessa instalação para o Sistema Operacional Windows é o que segue:

1. Download: a primeira ação a ser realizada é o download do Git For Windows (2020). No início da instalação você deverá aceitar o License Agreement da GNU, o que te levará à seleção de componentes da ferramenta.

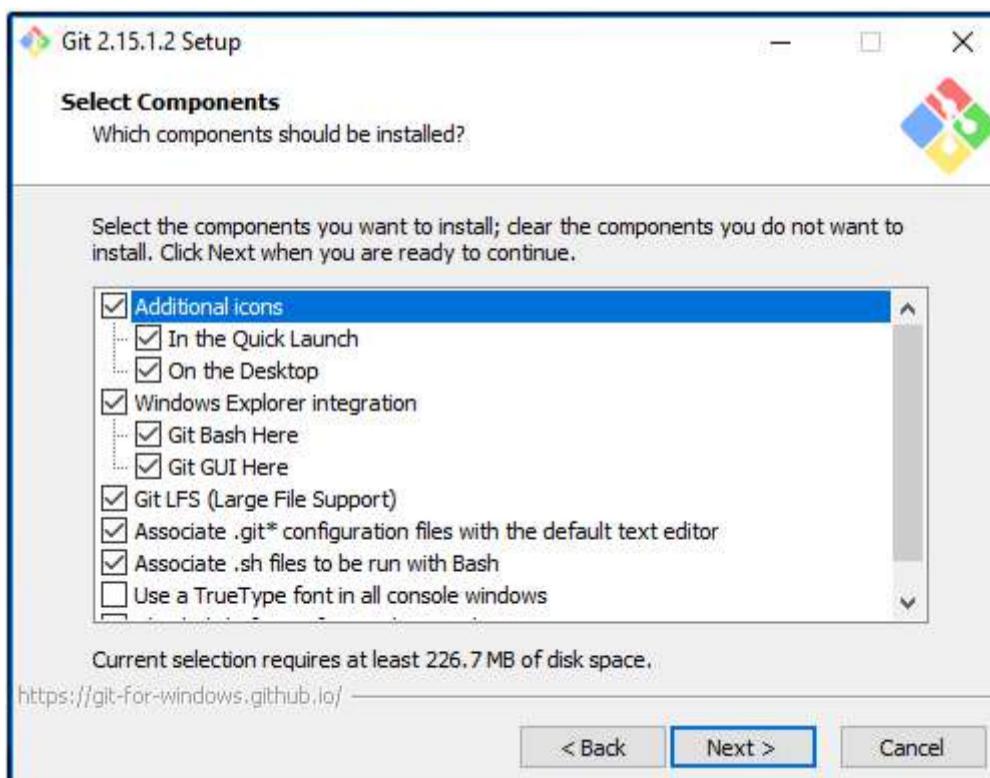
Figura 1.17 | Primeira etapa da instalação do Git



Fonte: captura de tela do executor de software do Windows.

2. Seleção de componentes: nesta etapa você deverá escolher os componentes a serem instalados e algumas associações de arquivos. A sugestão de seleção é a que segue ilustrada na captura de tela.

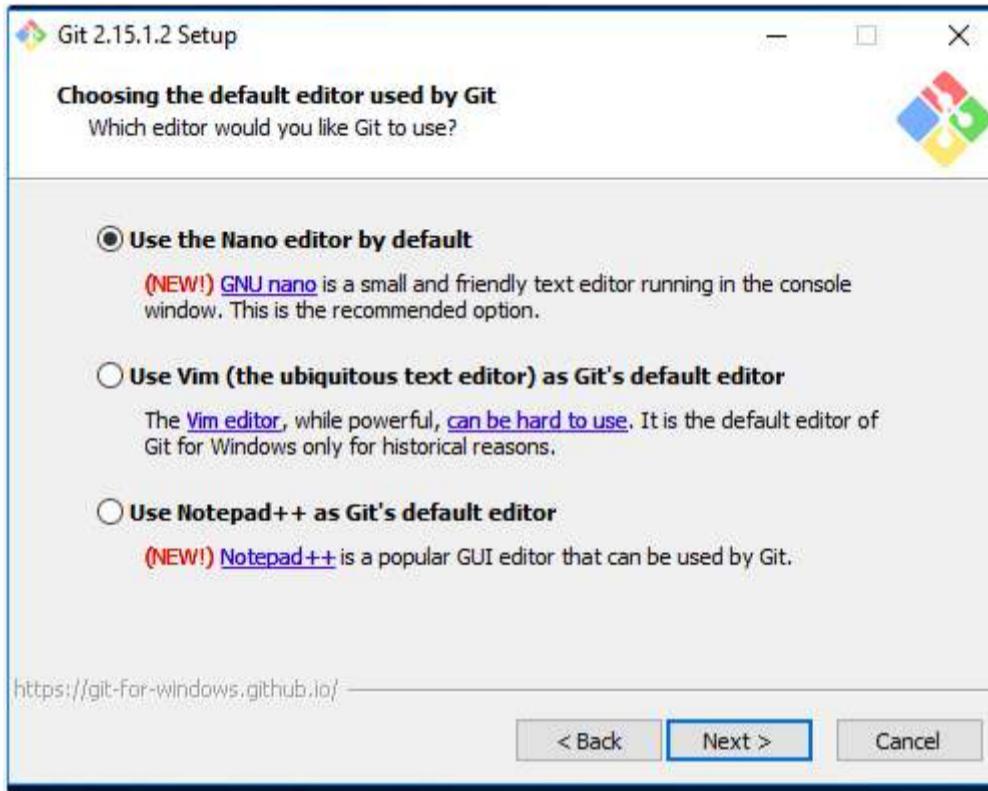
Figura 1.18 | Seleção de componentes do Git a serem instalados



Fonte: captura de tela do Git 2.15.1.2 Setup elaborada pelo autor.

3. Escolha do editor padrão: esta ação irá determinar qual editor de texto será aberto para as ações de *commit*. O editor Nano é uma boa opção.

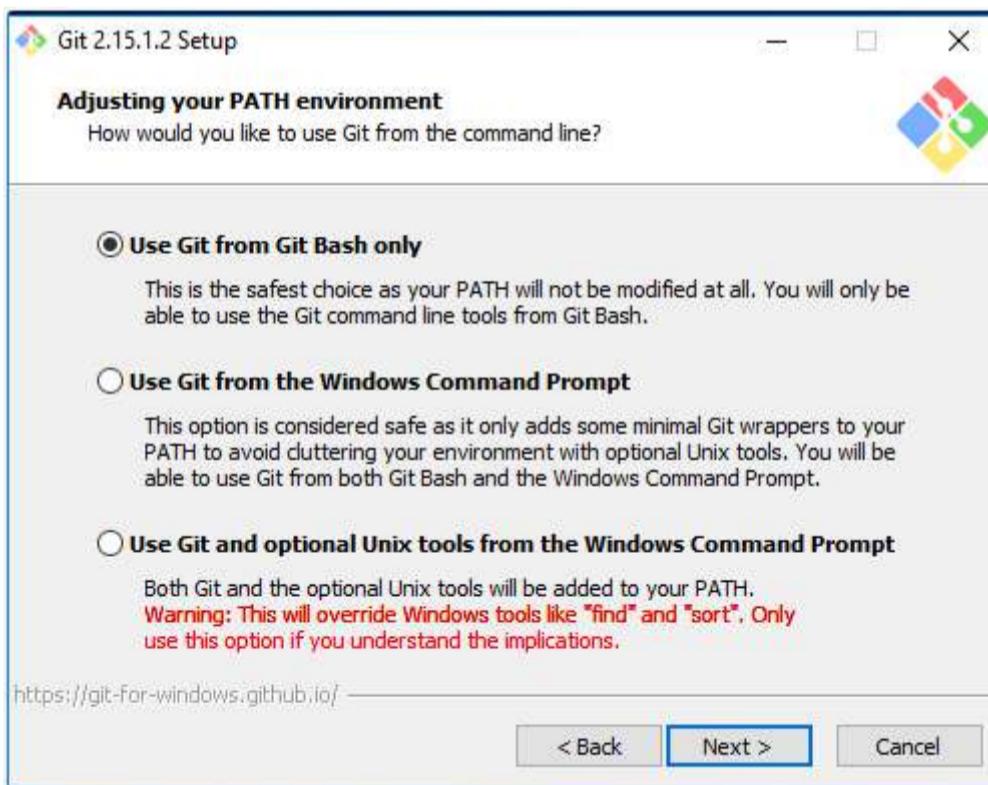
Figura 1.19 | Escolha do editor padrão do Git



Fonte: captura de tela do Git 2.15.1.2 Setup elaborada pelo autor.

4. Escolha do PATH Environment: a sugestão aqui é a utilização da opção “Use Git from Git Bash Only” para fins de escolha do modo de comando do Git.

Figura 1.20 | Ajuste de parâmetro de instalação do Git

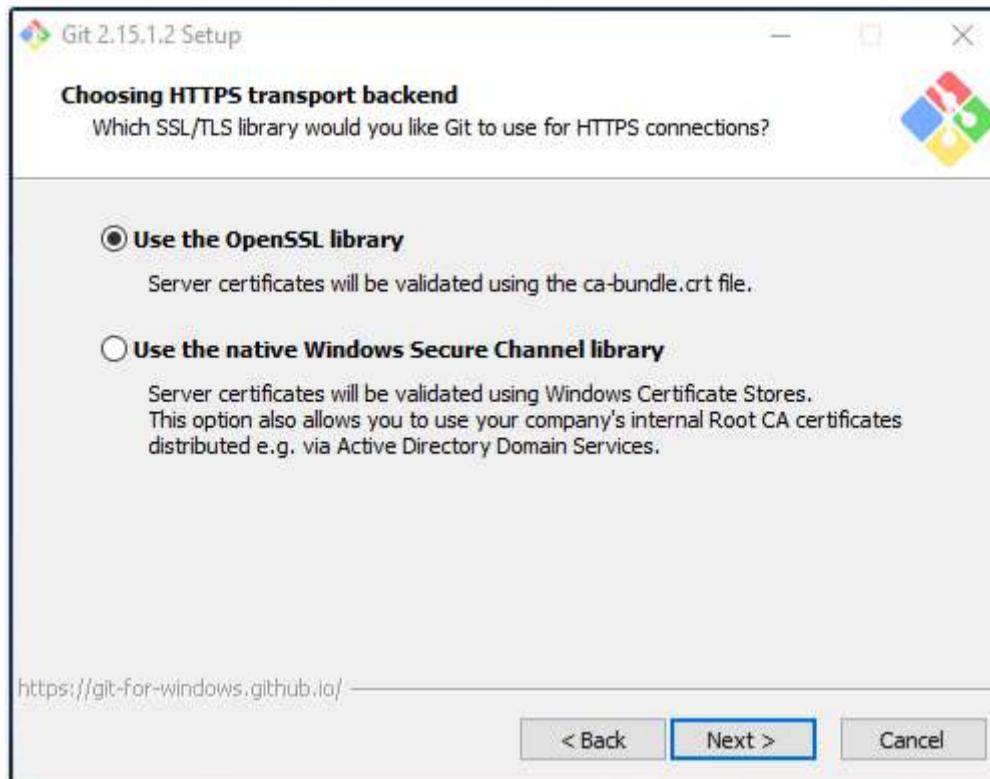


Fonte: captura de tela do Git 2.15.1.2 Setup elaborada pelo autor.

5. Seleção do transporte HTTPS: nesta opção deve ser especificada a biblioteca a ser usada em conexões seguras HTTPS. A sugestão é a

seleção do OpenSSL.

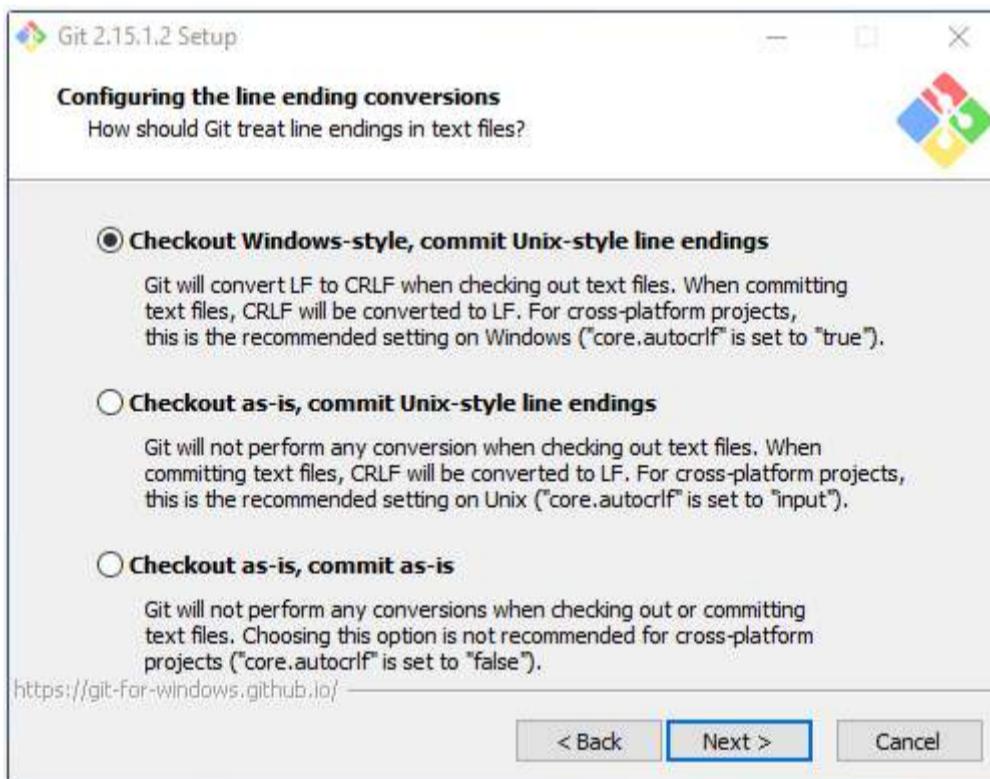
Figura 1.21 | Escolha de parâmetro de conexões HTTPS na instalação do Git



Fonte: captura de tela do Git 2.15.1.2 Setup elaborada pelo autor.

6. Configuração de finais de linha: trata-se da indicação de como o GitHub deverá tratar os finais de linha em arquivos de texto.

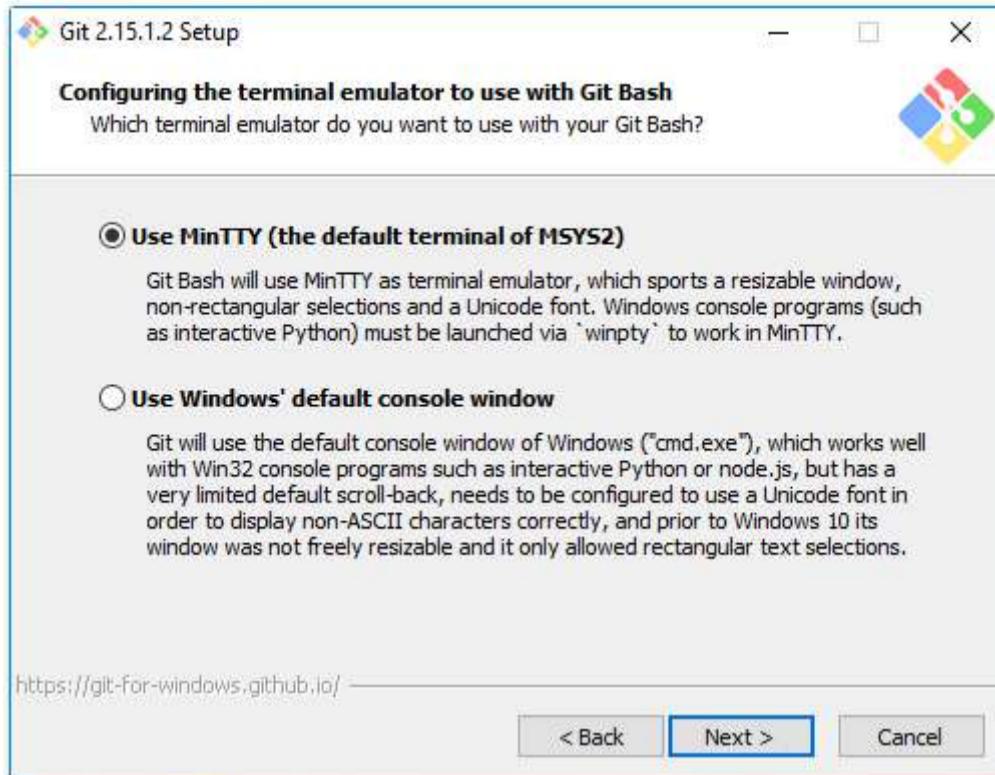
Figura 1.22 | Escolha de tratamento de final de linha na instalação do Git



Fonte: captura de tela do Git 2.15.1.2 Setup elaborada pelo autor.

7. Configuração do emulador de terminal: trata-se da seleção de qual emulador o GitHub deverá usar.

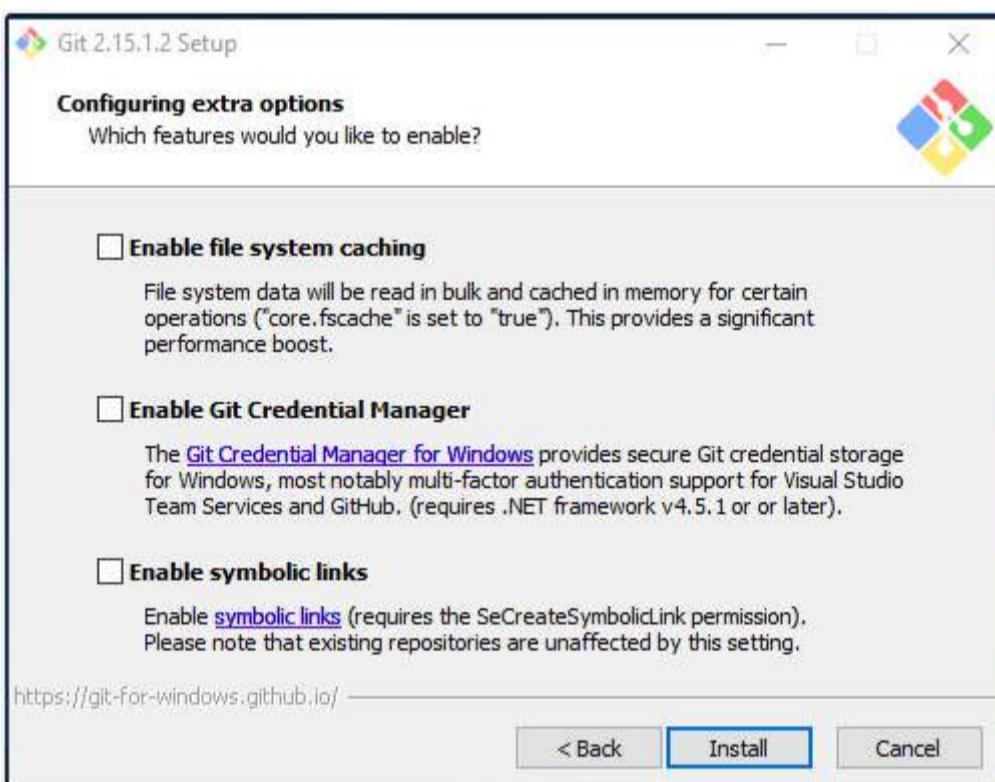
Figura 1.23 | Escolha do emulador de terminal na instalação do Git



Fonte: captura de tela do Git 2.15.1.2 Setup elaborada pelo autor.

8. Opções extras: a sugestão é que nenhuma delas seja selecionada.

Figura 1.24 | Opções extras de instalação do Git



Fonte: captura de tela do Git 2.15.1.2 Setup elaborada pelo autor.

Neste ponto a instalação deverá ser iniciada. Depois de instalada a ferramenta, basta acessá-la pelo Windows, como é feito com qualquer outro programa.

A configuração poderá ser feita pelo Git Bash, ou seja, por meio de linha de comando, basta iniciar o editor e a configuração inicial com os dados que seguem:

Figura 1.25 | Dados para configuração inicial do Git

```
# tells git who you are
$ git config --global user.name "Your Name"

# tells git how to email you
$ git config --global user.email "your.email@domain.com"

# handles end-of-line character differences
$ git config --global core.autocrlf true

# prevents conversion warning messages
$ git config --global core.safecrlf false

# sets Notepad as the default editor
$ git config --global core.editor notepad

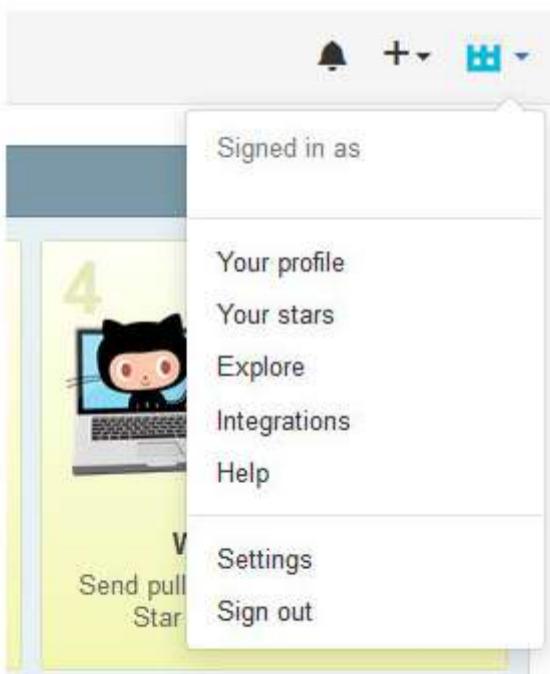
# list all current configuration settings
$ git config --list
```

Fonte: elaborada pelo autor.

No lugar de “*Your Name*”, insira seu nome e faça o mesmo com o comando de configuração de e-mail. O último comando *--list* exibe toda a configuração atual do GitHub. Ele servirá para que sejam conferidos os dados informados.

É chegado o momento de escolher um lugar para o repositório. Para criar um gratuito, acesse a página inicial do GitHub, faça seu cadastramento e escolha o plano gratuito, que garantirá acesso a repositórios públicos ilimitados e gratuitos. Feita essa escolha, seu repositório estará em um endereço como este: <https://github.com/seu-user-name>, sendo que “*seu-user-name*” é o nome que você escolheu no cadastramento. A partir da página de seu usuário será possível acessar uma série de opções, conforme ilustrado na figura que segue:

Figura 1.26 | Itens do menu do Git



Fonte: captura de tela do GitHub.

A conexão ao GitHub é configurada uma única vez e, por meio dela, será possível acessar o repositório a partir do seu computador, usando o protocolo SSH.

NÃO PODE FALTAR

INTRODUÇÃO À QUALIDADE DE SOFTWARE

Sergio Eduardo Nunes

Ver anotações

IMPORTÂNCIA DA QUALIDADE DE SOFTWARE

Os conceitos de qualidade de software estão ligados aos métodos, às ferramentas, aos processos e às medições, os quais guiam a equipe de desenvolvimento de forma mais assertiva e garantem que a entrega seja feita dentro dos padrões de qualidade estabelecidos entre as partes.



Fonte: Shutterstock.

Deseja ouvir este material?

CONVITE AO ESTUDO

Caro aluno, certamente você já deve ter procurado um aplicativo para resolver determinado problema, ou comprado um serviço ou produto, que, quando foi entregue, estava muito aquém do esperado. Pois bem, como usuário você deve ter percebido que sua concepção sobre a empresa, após a decepção, mudou radicalmente. Porém, nesse momento, seu papel não é mais o de usuário, mas o de desenvolvedor, o qual deve utilizar as técnicas de desenvolvimento de software para evitar tais situações.

Uma das técnicas para esse fim está relacionada com a qualidade de software e com suas ferramentas. Essa área do conhecimento da engenharia de software se preocupa em utilizar métodos, processos e normas nas atividades de desenvolvimento de software, a fim de se agregar qualidade nos processos e produtos finais. Para isso, é necessário que se tenha conhecimentos acerca de qualidade de software, qualidade do produto e qualidade dos processos. Como você pode ter percebido, existem diversas competências necessárias para se garantir a qualidade de um desenvolvimento.

Na primeira seção desta unidade de ensino, você compreenderá os conceitos gerais envolvidos na qualidade de software e aprenderá a identificar os problemas relacionados. Após isso, será possível entender como a garantia da qualidade e as métricas de qualidade são utilizadas em projetos de desenvolvimento de software, a fim de se gerar benefícios durante todo o ciclo de vida do projeto.

Na segunda seção, será possível que você compreenda como os modelos de qualidade são utilizados. Em seguida, será discutido como a norma ISO 9126 pode ser utilizada tal qual uma metodologia guia para a qualidade do produto de software. Enfim, você entenderá como a

gestão da qualidade do produto é feita e como as medições, os requisitos e as avaliações da qualidade dos produtos são executadas nos projetos de desenvolvimento.

E na terceira seção desta unidade de ensino, o foco estará nas discussões acerca da qualidade dos processos. Para isso, assuntos como modelos de maturidade, normas ISO na qualidade do processo, melhorias individuais e de equipe são primordiais para que você entenda de que forma essas técnicas são utilizadas nas atividades de desenvolvimento de software. Enfim, você aprenderá sobre a melhoria de processos de software no Brasil.

Todas essas discussões em caráter profissional farão com que você adquira competências para utilizar modelos, métodos, processos e normas a fim de garantir a qualidade nos processos de desenvolvimento de software e em produtos e serviços que atendam a qualidade e os padrões estabelecidos. Isso é de extrema importância para aqueles profissionais diretamente ligados a atividades de desenvolvimento de software, pois grande parte das empresas espera que os profissionais possuam tais competências. Pronto para mais esse desafio?

PRATICAR PARA APRENDER

Caro aluno, os aspectos ligados à qualidade estão na grande maioria de produtos e serviços que você utiliza em sua casa. Mas como isso se relaciona à qualidade de software? Analise a seguinte situação: uma pessoa fez uma filmagem no smartphone dela, mas deseja editar o vídeo, de forma que possa cortar algumas partes e remover o áudio. Após baixar diversos aplicativos e testar muitos deles, o vídeo foi danificado e não é mais possível fazer a sua restauração. Você não acha isso muito grave?

Pois bem, talvez esse vídeo tenha sido algo casual, mas e se fosse uma filmagem de festa, batizado, casamento, formatura, etc.? A perda para o usuário talvez fosse irrecuperável. Esse é apenas um exemplo dos

inúmeros erros que podem ser encontrados em softwares para diversos dispositivos. O que um profissional de tecnologia da área de desenvolvimento pode fazer para que tais situações sejam evitadas? A resposta a essa pergunta será vista nesta seção de estudos.

Você compreenderá a importância da qualidade de software para a garantia de entregas mais assertivas, sendo possível detectar erros, falhas, defeitos e bugs em projetos de software. Com isso, será possível que a garantia da qualidade sirva como guia nos processos de desenvolvimento, possibilitando que sejam utilizadas as métricas para aferir a qualidade dos processos, serviços e produtos, de forma que se compreendam os benefícios gerados por tais ferramentas.

EXEMPLIFICANDO

Profissionalmente, esses conhecimentos permitirão que você adquira competências importantes para serem aplicadas dentro dos projetos de desenvolvimento de software. Como desenvolvedor, você utilizará métodos, modelos e normas para garantir a qualidade dos seus desenvolvimentos. Já como gestor de projetos, deverá utilizar tais conhecimentos e ferramentas a fim de guiar o time de desenvolvimento a realizar as atividades de forma mais assertiva e dentro dos padrões de qualidade esperados.

As web rádios são uma forma interessante de prover serviços idênticos aos de radiodifusão (por antena). Seu meio de transmissão ocorre por uma infraestrutura de redes de computadores e, para isso, é necessário ter um host, no qual uma aplicação é instalada e configurada para que a web rádio possa ter as programações disponíveis aos usuários. Com essa facilidade, diversos seguimentos têm optado por adquirir uma dessas.

O time de futebol da cidade, preocupado com a falta de transmissão de seus jogos na TV aberta ou nas emissoras de rádio, encontrou uma solução interessante na web rádio para dar visibilidade ao time. Os diretores levaram a discussão ao conselho e, após aprovado, o setor de mídias foi autorizado a buscar uma empresa para fazer o desenvolvimento do projeto.

A empresa na qual você trabalha está no mercado de desenvolvimento há mais de dez anos e entre os projetos estão sites institucionais, blogs, sistemas web e rádios web. Nos últimos meses, você tem trabalhado na área de desenvolvimento de *front-end* tanto para sistemas quanto para sites estáticos. O gerente do seu setor vem há tempos introduzindo a ideia de se utilizar algum framework para a garantia da qualidade em projetos de desenvolvimento de software.

Ao saber do projeto da web rádio para o time da cidade, o gerente de projetos da área de desenvolvimento *front-end* procurou se preparar para utilizar uma ferramenta cuja ação, nos processos desse projeto, funcionaria como guia garantidor de qualidade.

Sabendo que você tem estudado acerca da qualidade de software, o gerente fez a seguinte solicitação em seu e-mail:

Caro,

Estamos iniciando o projeto da web rádio e desejamos utilizar ferramentas para a garantia da qualidade. Como você havia comentado que estava estudando a respeito do tema, vamos necessitar de suas competências.

Para isso, gostaria que você desenvolvesse um checklist para front-end com os elementos: HEAD, HTML, WEBFONTS, CSS (Cascading Style Sheets) e SEO (Search Engine Optimization). As entradas que você apontar para cada um dos itens serão utilizadas por toda a equipe como uma ferramenta de qualidade dentro da nossa área. Conto com você!

O gerente de projetos quer inovar dentro da empresa e você tem a oportunidade de alavancar sua carreira junto com ele. Agora é o momento de mostrar os conhecimentos adquiridos acerca de qualidade de software.

DICA

Você ficou curioso para saber como essas técnicas poderão proporcionar benefícios aos seus desenvolvimentos? E também em termos profissionais? Explore todos os conhecimentos discutidos nesta seção e o mais importante: aplique-os. Acredite, você se tornará um grande profissional com tais conhecimentos!

CONCEITO-CHAVE

Caro aluno, certamente alguma vez na vida você já deve ter adquirido um produto ou serviço, o qual, no momento da compra, parecia perfeito, mas, ao utilizá-lo, as coisas acabaram saindo bem diferente do prometido. Como consumidor, isso traz muita frustração. E isso não é uma exclusividade da Tecnologia da Informação. Corriqueiramente, ocorre também na telefonia, na alimentação, no e-commerce, na entrega de encomendas, de roupas, etc.

■ QUALIDADE DE SOFTWARE

No que se diz respeito à desenvolvimento de softwares, a qualidade visa basicamente atender as necessidades e expectativas do cliente, a fim de cumprir os requisitos acordados no início do projeto de desenvolvimento de software. O tema é bem mais abrangente do que essa simples definição e é isso que torna os assuntos relacionados à qualidade de software de extrema importância para os profissionais de T.I.

Segundo Zanin *et al.* (2018), a qualidade de software está ligada aos aspectos de conformidade com os requisitos funcionais e não funcionais encontrados nos desenvolvimentos de softwares. Para compreender melhor, observe a definição dos requisitos:

- **Requisitos funcionais:** especificam uma função que o sistema ou que determinado componente do sistema deve realizar. Essa descrição é feita do ponto de vista do usuário e pode ser determinante para o comportamento do sistema, de modo que precisa ficar bem claro o que deve ocorrer com uma entrada no software e qual a saída gerada. Para melhor exemplificar esse conceito, imagine que o seu cliente descreva que um software deve, após o usuário incluir os produtos no carrinho de compras, apresentar a opção de pagamento em crédito, débito ou boleto bancário.

A nível de desenvolvimento, esse funcionamento é de extrema importância, pois determina quais são as funcionalidades que o software deve apresentar. Por exemplo: inserir, alterar, excluir, listar produtos, etc.

- **Requisitos não funcionais:** esses requisitos podem estar relacionados a necessidades que devem ser atendidas, porém não utilizam funcionalidades. Esse conceito está diretamente ligado à qualidade de software, ou seja, trata das premissas técnicas que o software deve desempenhar. Um exemplo é a solicitação de um cliente para que determinada aplicação web seja responsiva, isto é, que permita ao usuário utilizá-la em um desktop e em um smartphone.

Esses requisitos não funcionais podem ter um detalhamento técnico mais específico, como: o sistema deve se comunicar obrigatoriamente com o SGBD MySQL; a solução desenvolvida tem que ser voltada para o sistema operacional Linux; a consulta de novos clientes deve permitir que seja efetuada *off-line*, entre

diversos outros requisitos não funcionais, os quais podem ocorrer nas atividades de desenvolvimento de software.

Mas, enfim, o que é qualidade?

Diz respeito aos métodos, às ferramentas, às metodologias e aos processos os quais garantirão que determinada entrega seja feita dentro dos padrões de qualidade estabelecidos entre as partes. Todo esse processo se inicia no levantamento de requisitos, quando o cliente passa para a equipe de desenvolvimento todas as suas necessidades.

Para visualizar o objeto, acesse seu material digital.



Compreender os requisitos permite entender de que forma os aspectos relacionados à qualidade de software estão intimamente ligados com os processos de desenvolvimento. Para Pressman (2006), o ciclo de desenvolvimento de um software é dividido em seis partes, conforme pode ser observado na Figura 2.1.

Figura 2.1 | Ciclo de desenvolvimento de software



Fonte: elaborada pelo autor.

Em todo o ciclo de vida do projeto, podem ser utilizadas as ferramentas de garantia da qualidade, ou seja, nos processos de desenvolvimento, teste, validações, correções, enfim, em qualquer parte que venha a compor um projeto de desenvolvimento. É importante lembrar também que essas metodologias de garantia da qualidade necessitam de parâmetros, de métricas, que podem variar conforme métodos, necessidades, recursos, etc.

o

Ver anotações

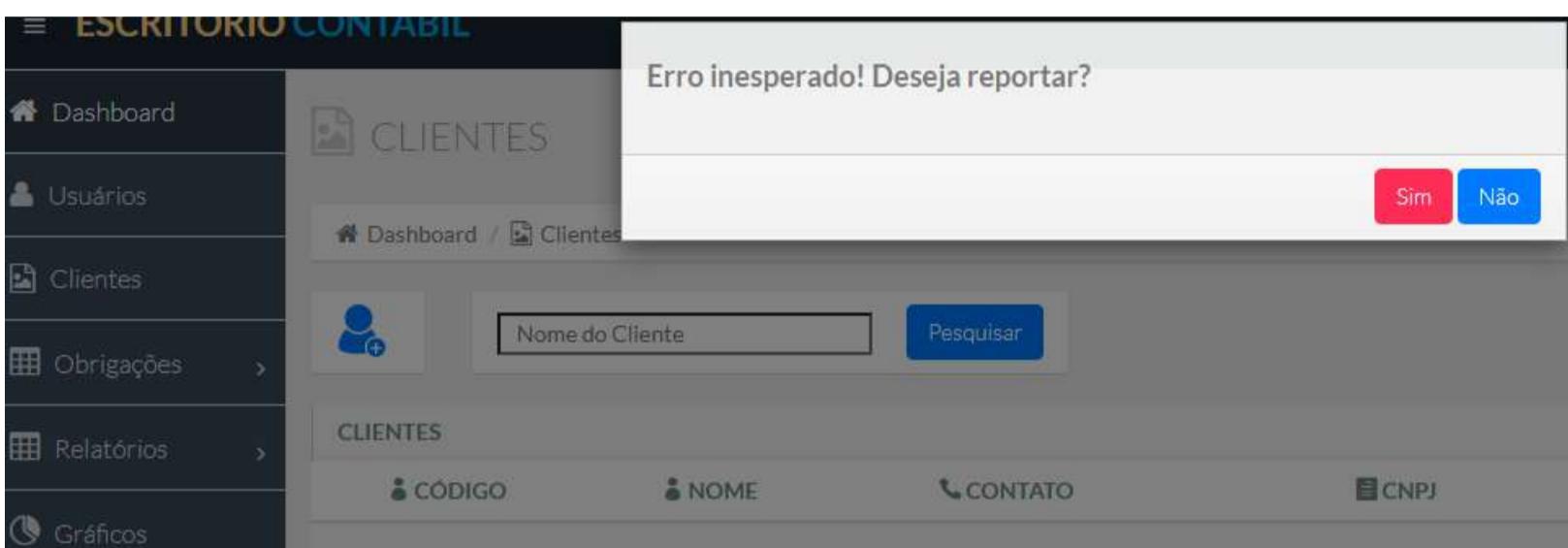
Ainda segundo o autor, o termo qualidade de software pode ser definido como o atendimento das conformidades funcionais com um desempenho esperado pelo patrocinador. Porém, para Sommerville (2011), a qualidade nas atividades de desenvolvimento de software deve ser mais abrangente, devendo ser gerenciada e compreendida em três níveis:

1. **Organizacional:** a preocupação é de um nível mais amplo, no qual o objetivo é o estabelecimento de padrões de trabalho de desenvolvimento de software. Esses frameworks devem agrupar as melhores práticas para que os erros e falhas sejam minimizados.
2. **Projeto:** envolve o desenvolvimento com base em padrões determinados por gestores de projetos. Isso pode variar conforme o projeto, a política da empresa, a utilização de frameworks, entre outras peculiaridades.
3. **Planejamento:** deve haver um plano de qualidade, ou seja, parte da equipe deve ficar responsável pela verificação dos requisitos de qualidade acordados entre as partes. Tanto os processos quanto os produtos desenvolvidos devem ser revisitados afim de se evitar que algo passe despercebido.

Caro aluno, para especificar de maneira mais clara alguns aspectos que podem impactar diretamente no quesito qualidade, observe de que formas eles podem se apresentar.

Segundo Zanin *et al.* (2018), a falha de software pode ser um comportamento inesperado do sistema e pode ser ocasionado por um ou mais erros. Para melhor compreensão de como a falha de software pode afetar um sistema, observe a Figura 2.2.

Figura 2.2 | Falha de software



Fonte: captura de tela de um software em desenvolvimento elaborada pelo autor.

Nesse exemplo, acima do aceite, o sistema retorna uma mensagem de erro, impedindo, assim, que um novo cliente seja cadastrado, pois a falta da confirmação do aceite que impede a finalização do processo. Isso, a nível de usuário, é bem impactante. Imagine que esse sistema seja de uma loja de departamentos, a qual depende desse sistema para gerar as vendas e as notas fiscais. Certamente o prejuízo seria muito grande e preocupante.

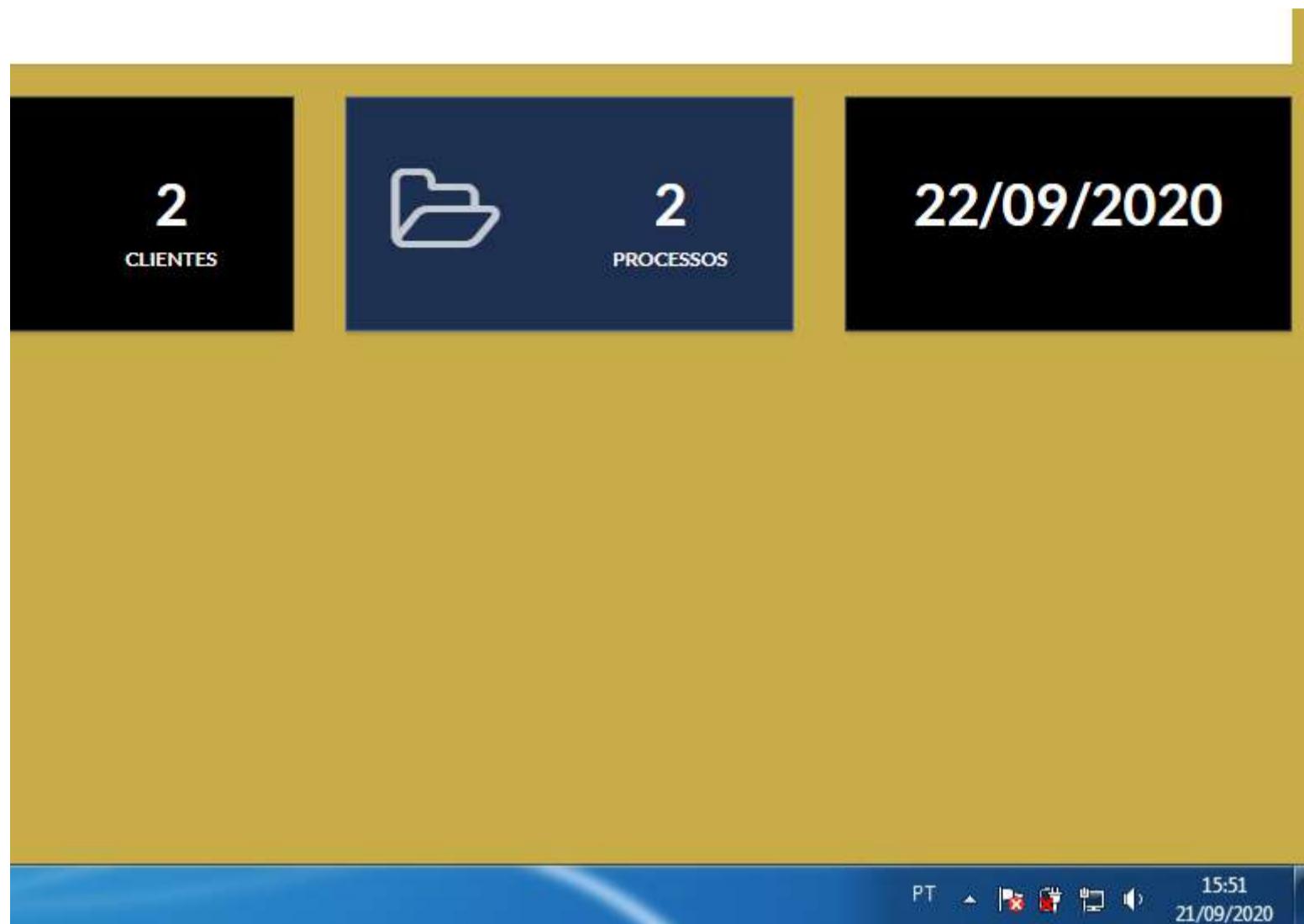
Vale aqui ressaltar que o desenvolvedor é o responsável pela implementação das rotinas que cuidam das tratativas para falhas e erros. Elas podem ser exibidas por meio de uma mensagem ao usuário, podem levar a uma página nova com alguma imagem que remeta a falhas e erros ou podem, ainda, utilizar um report para enviar os problemas a algum repositório, com o intuito de que os desenvolvedores ajustem o sistema.

Segundo Zanin *et al.* (2018), grande parte dos erros de software estão relacionados a execuções incorretas, o que faz com que os resultados gerados não reflitam a verdade. Um exemplo pode ser observado na Figura 2.3.

0

Ver anotações

Figura 2.3 | Erro de software



Fonte: captura de tela de um software em desenvolvimento elaborada pelo autor.

Observe, nesse exemplo, a data do sistema e a data marcada no relógio. O erro de data no sistema pode ter um impacto muito grande na integridade dos seus dados, fazendo com que os lançamentos não sejam confiáveis quanto a sua data. Isso para setores de venda, logísticos, contábeis e fiscais pode ser desastroso.

| DEFEITO DE SOFTWARE

Segundo Zanin *et al.* (2018), o defeito de software se refere a uma implementação incorreta, que ocasiona um erro, uma interrupção de serviço ou, ainda, um mal funcionamento. Um exemplo pode ser um sistema de cadastro que, após o preenchimento de todos os campos obrigatórios, executa o processo de inserção de dados, não retorna erros, porém não faz a inserção das informações no banco de dados.

Quanto às atividades de desenvolvimento, os defeitos são mais difíceis de se encontrar, pois, em alguns casos, erros de lógica, de falha de escrita do código ou de qualquer outra referência não retornam para auxiliar o desenvolvedor a encontrar o problema.

BUGS

Esse termo se popularizou entre os jovens para fazer referência a comportamentos inesperados de softwares. Segundo Zanin *et al.* (2018), um bug de sistema diz respeito a erros e falhas inesperados, que normalmente são de maiores complexidades e demandam mais tempo e conhecimento técnico para que sejam encontrados e solucionados.

EXEMPLIFICANDO

Um bug que tirou o sono de muitos administradores de sistemas foi o bug do milênio. Em 1999, os profissionais estavam preocupados com o comportamento dos sistemas na virada de ano, porque os sistemas antigos faziam a leitura apenas dos dois últimos dígitos do ano. Com isso, em vez de o sistema ir para o ano 2000, poderia voltar para o 1900. Por isso o nome Bug do Milênio.

Esse acontecimento foi veiculado em muitos meios de comunicação, como no site *G1*:

SANDERS, N. Há 20 anos, o 'bug do milênio' e o fim do mundo que não aconteceu. **G1**, São Paulo, 31 dez. 2019.

Caro aluno, agora que você compreendeu como a qualidade de software está profundamente relacionada com os processos de desenvolvimento computacionais, certamente virá aquela dúvida clássica: mas de que forma podemos garantir a qualidade de um software? Para que você possa compreender os aspectos relacionados à garantia da qualidade dos softwares, neste momento, procure ter uma

visão além das funcionalidades, do desempenho, da escalabilidade, etc. Assim, pode-se incluir a qualidade dos processos para desenvolvê-los, testá-los e liberá-los para utilização.

GARANTIA DA QUALIDADE

Sommerville (2011) mostra que a garantia da qualidade é conhecida como *Software Quality Assurance* (SQA). A sua abrangência se estende por todo o ciclo de vida do projeto de desenvolvimento de software e deve:

- Possuir ferramentas e/ou métodos que permitam a análise dos desenvolvimentos e dos testes.
- Efetuar revisões técnicas nos componentes e na funcionalidade, devendo ser feitas em cada uma das fases.
- Controlar a documentação por meio de versionamento.
- Atribuir métodos para se garantir padrões de desenvolvimento e das boas práticas, as quais atendem as necessidades das equipes de desenvolvimento.
- Obter mecanismos de aferição.

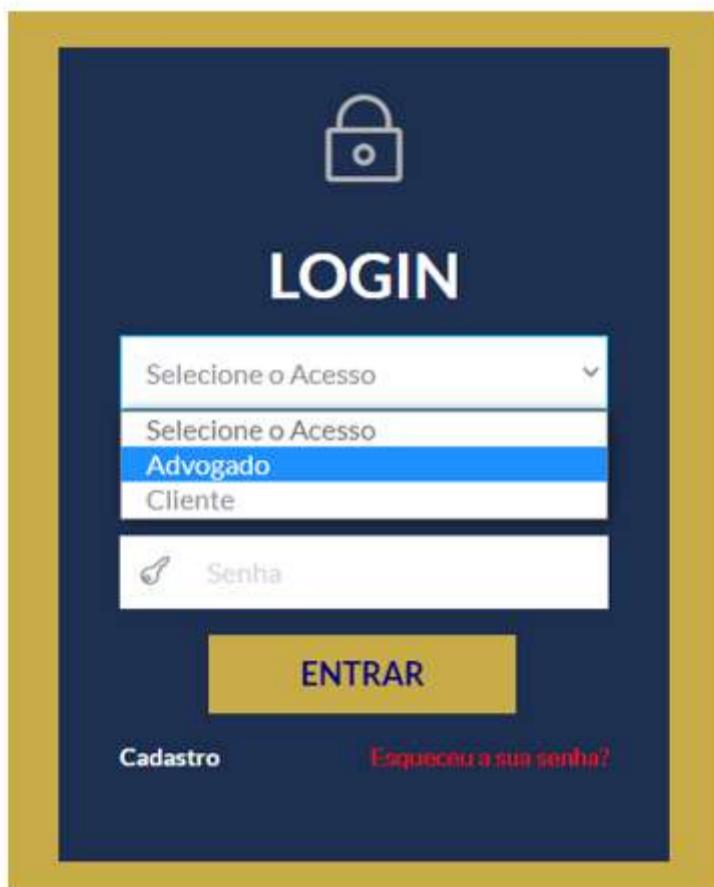
Para Pressman (2006), a garantia da qualidade diz respeito aos procedimentos, métodos e ferramentas utilizados por profissionais de Tecnologia da Informação para se garantir padrões acordados entre as partes durante todo o ciclo de vida do desenvolvimento de um software. É importante lembrar que os padrões de qualidade podem variar conforme o projeto e, por isso, a garantia da qualidade deve ser guiada pelo que foi acordado entre as partes.

Com o intuito de que você possa compreender melhor como a garantia da qualidade é, de fato, operacionalizada em atividades de desenvolvimento em um meio profissional, vamos analisar um case. Imagine que foi solicitado por um escritório de advocacia um programa para gerenciar os processos advocatícios. Em primeiro lugar, foi dito que

tanto os advogados quanto os clientes deverão fazer login no sistema por um único local e de forma transparente. Tempos depois foi apresentada a tela de login ao cliente, conforme pode ser observado na Figura 2.4.

o

Figura 2.4 | Tela de login



Fonte: captura de tela de um software em desenvolvimento elaborada pelo autor.

Repare que o acordo entre as partes era o login transparente. Dessa forma, o menu *dropdown* para escolher o tipo do usuário não deveria ter sido utilizado na tela. A garantia da qualidade deve utilizar as ferramentas para detectar o problema, pois esse ajuste tem um impacto tanto no desenvolvimento do *front-end* quanto no do *back-end*, porque a lógica para se efetuar um login no sistema muda completamente.

Caro aluno, certamente você deve estar pensando que, para determinar se algo está atendendo à qualidade, deve haver métricas. Pois bem, em qualidade de software também existem regras definidas, que fornecem formas de avaliar se os processos e produtos estão dentro da qualidade acordada entre as partes.

Segundo Zanin *et al.* (2018), as diversas engenharias utilizam o processo de medição, para verificar se determinado desenvolvimento está em conformidade e segurança para ser utilizado. Porem, diferentemente

Ver anotações

das demais áreas de conhecimento, a engenharia de software não utiliza leis quantitativas ou medidas absolutas, mas um conjunto de medidas que dão um feedback quanto aos aspectos qualitativos do software.

Boehn, Brown e Lipow (1977) determinam que, para efetuar a medição da qualidade de um software, deve-se determinar quais são as funcionalidades que devem/podem ser medidas e de que forma precisam ser medidas. A fim de orientar os desenvolvedores no alinhamento das métricas, os autores sugerem o esquema demonstrado na Figura 2.5.

Figura 2.5 | Árvore de características de qualidade de software
Para visualizar o objeto, acesse seu material digital.

Fonte: Boehn, Brown e Lipow (1977, p. 593).

Em sua estrutura mais alta, o software deve possuir elementos como usabilidade e manutenibilidade. E, em sua estrutura média, a portabilidade, a confiabilidade, a eficiência, a engenharia humana, a facilidade de teste, a facilidade de entendimento e a facilidade de modificação. Dessa estrutura média, derivam as primitivas (que dão as características dos elementos de nível médio). As primitivas podem ser verificadas por meio de checklists, ou seja, representam a atividade de desenvolvimento em si, que deve ser verificada quanto ao atendimento da garantia da qualidade.

Você deve ter percebido que essa ferramenta é bem complexa, mas observe como ela auxilia na atividade de encontrar problemas relacionados à qualidade de software. Para isso, observe a Figura 2.6.

Figura 2.6 | Cadastro do cliente

A captura de tela mostra duas versões do mesmo sistema de cadastro de usuários. A versão esquerda, para navegador, tem uma interface mais completa com uma barra superior contendo links para 'Usuários', 'Clientes' e 'Produtos'. Abaixo, uma seção 'USUÁRIOS' exibe uma tabela com os dados de dois usuários: 'Serginho' e 'Testando', incluindo campos para nome, e-mail e senha. Um link 'Novo Usuário' está no topo da lista. A versão direita, para mobile, é uma versão simplificada com uma barra superior com os mesmos links. Abaixo, uma seção 'USUÁRIOS' exibe uma tabela com os mesmos dois usuários, mas sem o link 'Novo Usuário'.

Nome	E-mail	Senha
Serginho	teste@teste.com	81dc9bdb52d04dc20036
Testando	teste@teste.com	81dc9bdb52d04dc20036

Nome	E-mail	Senha
Serginho	teste@teste.com	81dc9bdb52d04dc20036
Testando	teste@teste.com	81dc9bdb52d04dc20036

Fonte: captura de tela de um software em desenvolvimento elaborada pelo autor.

Nesse exemplo, o sistema de cadastro está com a funcionalidade de adicionar um novo usuário e exibir os cadastrados no sistema. Do lado esquerdo é apresentada a exibição desse sistema em navegador de desktop, já do lado direito está a apresentação em um smartphone. Com isso, vamos utilizar a árvore de qualidade de software apresentada na Figura 2.5 para analisar esse cenário.

No primeiro caminho: Utilidade geral → Portabilidade → Independente do dispositivo, já se observa que o software apresenta um problema de responsividade, o que demonstra que o sistema não é portável.

Outro exemplo em que a qualidade é atendida pode ser verificado no caminho: Utilidade Geral → Usabilidade → Confiabilidade → Precisão. Isso ocorre, pois, quando uma transação de inserção de dados é feita, a operação se completa corretamente.

ASSIMILE

Checklist é uma ferramenta de grande importância para profissionais de desenvolvimento. Existem softwares que auxiliam o desenvolvedor nessas tarefas. Neles é possível detalhar as atividades do checklist, determinar a quantidade de tempo e ajustar as atividades predecessoras e sucessoras.

Caro aluno, você percebeu como os processos de qualidade de software são de extrema importância para os desenvolvimentos? Com esses estudos, foi possível compreender que, ao se utilizar as técnicas e ferramentas de forma adequada, serão alcançados alguns benefícios. E quais são esses benefícios encontrados nos processos de qualidade de software?

Segundo Sommerville (2011), as vantagens encontradas ao se operacionalizarem os processos de garantia da qualidade dependem de um esforço coletivo, que proporciona a economia de recursos durante todo o ciclo de vida do projeto de desenvolvimento de software. Ainda que de maneira bem específica, os processos de qualidade podem proporcionar algumas vantagens, listadas a seguir:

- **Padronização:** boas práticas passam a ser adotadas pelas equipes de desenvolvimento.
- **Aumento de produtividade:** ao se minimizar os retrabalhos por meio da verificação da qualidade, o tempo de produtividade é otimizado.
- **Satisfação do cliente:** com as ferramentas de verificação da qualidade, a equipe de desenvolvimento realiza as atividades de forma mais assertiva.
- **Economia de recursos:** redução nos custos operacionais como um todo, sendo observados pontos como: tempo, custo com colaborador, custos gerais de operação.
- **Retrabalho:** evita que grandes correções e ajustes sejam necessários, fazendo com que muitas vezes as atividades com dependências funcionais necessitem ser paralisadas.

Muitas vezes as vantagens obtidas com a aplicação dos recursos de qualidade nas atividades de desenvolvimento de software demoram a serem sentidas no dia a dia, uma vez que isso ocorre conforme a utilização das ferramentas aplicadas torna-se uma prática comum e

corriqueira. A partir desse ponto é que, naturalmente, as vantagens começam a fazer parte de todo o ciclo de vida do desenvolvimento de um software.

REFLITA

Dentro das atividades de desenvolvimento de software existem diversos profissionais de diferentes áreas que devem conversar entre si, entre eles há: designers, programadores e DBA (Database Administrator). O bom diálogo entre eles é de extrema importância para que a integração ocorra de forma a atender as necessidades do sistema e com a qualidade desejada.

Caso os diferentes integrantes não tivessem uma proximidade, quais das vantagens em se utilizar os processos de qualidade de software poderiam ser prejudicadas?

SAIBA MAIS

A Sociedade Brasileira de Computação é uma entidade que visa, por meio de eventos, promover o conhecimento através de debates e fóruns. Anualmente ocorre um simpósio totalmente voltado às discussões acerca de qualidade de software.

Caro aluno, os conceitos de qualidade de software estão intrinsecamente ligados às ferramentas, aos métodos e às medições, que guiam a equipe de desenvolvimento de forma mais assertiva. Com isso, torna-se muito importante que o profissional de Tecnologia da Informação, o qual tem, em sua área de atuação, o desenvolvimento de software, comprehenda as características técnicas ligadas à qualidade, as suas falhas e a como conseguir a garantia de qualidade nos processos de desenvolvimento durante o ciclo de vida do projeto. Dessa forma,

podem-se proporcionar vantagens relevantes aos processos de desenvolvimento de software, que são convertidas em aumento de satisfação do cliente.

FAÇA VALER A PENA

Questão 1

Quando um projeto de desenvolvimento de software é iniciado, uma etapa muito importante é realizada: o levantamento de requisitos. Essa fase busca apresentar formas de a equipe de desenvolvimento compreender as necessidades do cliente. Os requisitos podem ser funcionais e não funcionais. Sabendo que os requisitos não funcionais estão diretamente ligados à qualidade do software, o profissional deve saber identificá-los.

Com base no exposto, assinale (F) para requisitos funcionais e (NF) para os requisitos não funcionais.

- () Inserir clientes pessoa física e jurídica.
- () A aplicação deve permitir a instalação nos sistemas operacionais Windows e Linux.
- () Todos os componentes devem ser responsivos, pois serão utilizados em notebooks e smartphones.
- () O sistema deve possuir funcionalidade para pagamento via cartão de débito e crédito e para emissão de nota.
- () O tempo de resposta da lista de produtos disponíveis deve ser just-in-time, de modo que o usuário não fique aguardando o carregamento.

Assinale a alternativa que apresente a sequência correta.

a. F – F – NF – NF – NF.

b. NF – F – NF – F – NF.

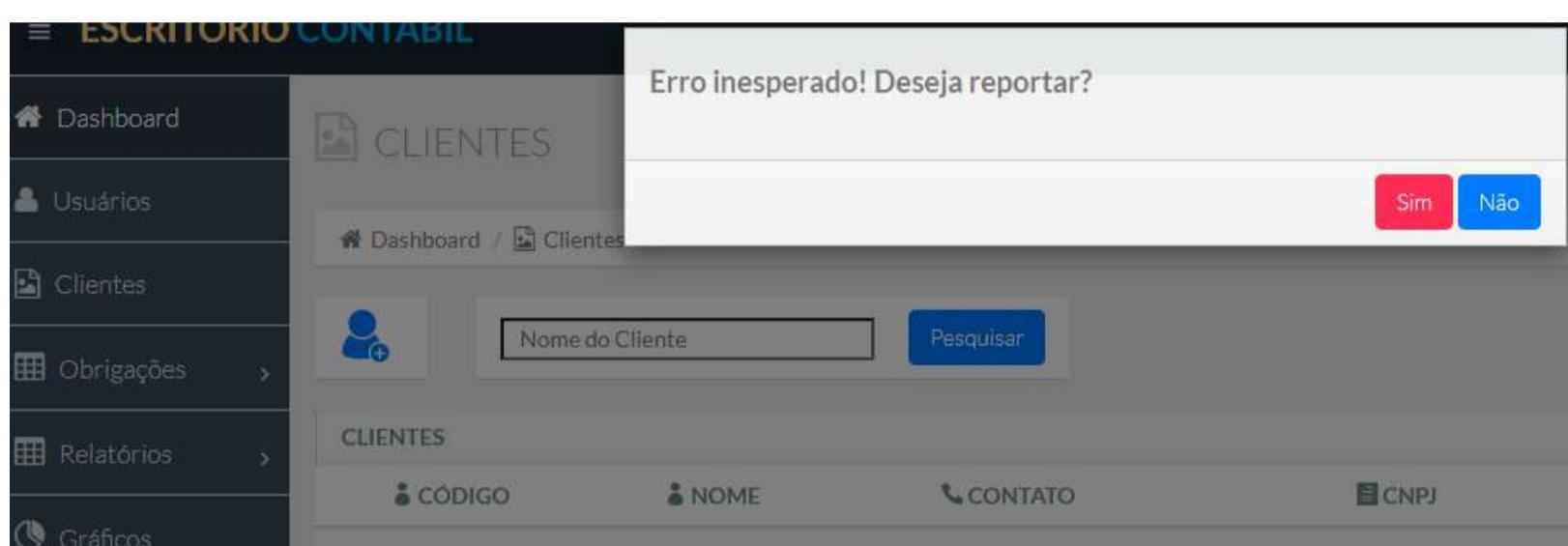
c. F – NF – NF – F – NF.

d. F – NF – F – NF – NF.

Questão 2

O desenvolvimento de softwares é uma atividade extremamente complexa. Por esse motivo, ter ferramentas de qualidade é um grande diferencial para se garantir que o produto final atenda às necessidades e às expectativas do cliente. Porém, em determinadas situações, algumas funcionalidades não executam as operações como o desejado. Observe a figura a seguir:

Figura | Exclusão de registro



Fonte: captura de tela de um software em desenvolvimento elaborada pelo autor.

Nesse exemplo, o cliente seleciona a opção para excluir determinado registro, confirma a exclusão no modal, porém o registro não é excluído do banco de dados e, consequentemente, continua sendo apresentado na lista.

Com base na situação apresentada, assinale a alternativa com o tipo do problema.

a. Falha de software.

b. Erro de software.

c. Defeito de software.

d. Bug.

e. Anomalia.

Questão 3

Toda e qualquer técnica, método, ferramenta ou tecnologia que se utilize em determinada atividade de desenvolvimento de software tem a intenção de fazer com que algumas vantagens sejam alcançadas a fim de se agregar qualidade.

Ao utilizar as ferramentas de garantia da qualidade, a equipe de desenvolvimento visa alcançar algumas vantagens. A partir do apresentado, analise as asserções a seguir e a relação proposta entre elas.

I. Padronização das tarefas, que passam a se tornar boas práticas para a equipe de desenvolvimento.

POIS

II. As atividades tendem a ser repetitivas e repeti-las será uma grande vantagem aos desenvolvedores, à equipe, à empresa e ao cliente.

A seguir, assinale a alternativa correta:

- a. As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- b. A asserção I é uma proposição verdadeira e a asserção II é uma proposição falsa.
- c. As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- d. A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- e. As asserções I e II são proposições falsas.

REFERÊNCIAS

BOEHN, B. W.; BROWNN, J. R.; LIPOW, M. **Characteristics of Software Quality.** [S.I.: s.n.], 1977.

GERÊNCIA e Qualidade de Software – Apresentação da disciplina. [S.I.: s.n], 2018. 1 vídeo (2 min). Publicado pelo canal UNIVESP. Disponível em: <https://bit.ly/38hgONe>. Acesso em: 25 set. 2020.

PRESSMAN, R. S. **Engenharia de Software.** 6. ed. São Paulo: McGraw-Hill, 2006.

SANDERS, N. Há 20 anos, o 'bug do milênio' e o fim do mundo que não aconteceu. **G1**, São Paulo, 31 dez. 2019. Disponível em: <https://glo.bo/3bhqaum>. Acesso em: 19 out. 2020.

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

TONINI, A. C.; CARVALHO, M. M.; SPINOLA, M. M. Contribuição dos modelos de qualidade e maturidade na melhoria dos processos de software. **Produção**, v. 18, n. 2, p. 275-286, 2008. Disponível em: <https://bit.ly/2XdqC4L>. Aceso em: 19 out. 2020.

ZANIN, A. *et al.* Qualidade de Software. Porto Alegre: Sagah, 2018.

FOCO NO MERCADO DE TRABALHO

INTRODUÇÃO À QUALIDADE DE SOFTWARE

Sergio Eduardo Nunes

Ver anotações

O QUE É UM CHECKLIST?

Checklist é uma ferramenta de grande importância para profissionais de desenvolvimento, que auxilia o desenvolvedor em suas tarefas, detalhando e ajustando atividades, e determinando a quantidade de tempo.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

O projeto da web rádio é muito instigante, pois é algo inovador e ao mesmo tempo que permite, no projeto, níveis de detalhamento interessantes para aplicar as ferramentas de garantia de qualidade. Por isso, foi solicitado um checklist em que pudessem ser apontados os seis elementos mais importantes para a empresa.

Dessa forma, para que possamos iniciar de maneira inovadora dentro do setor, o checklist deve estabelecer padrões de desenvolvimento conforme o observado a seguir:

- HEAD:
 - Doctype.
 - Charset.
 - Title.
 - Favicon.
 - CSS link.
- HTML:
 - Header.
 - Section.
 - Footer.
 - Página de erro.
- WEBFONTS:
 - Formato.
 - Tamanho.
 - Cor.
- CSS:
 - Responsividade.
 - Letras.

- Cores de elementos.

- SEO:

- Google analytics.

- Sitemap.xml.

- Robots.txt.

Podem, ainda, ser adicionados mais elementos conforme o projeto, de acordo com o qual alguns elementos poderiam ser obrigatórios e os demais, opcionais. O checklist ainda pode receber mais elementos como IMAGES, padronização de comentários, JAVASCRIPT e até o back-end futuramente.

Conforme observado, ao utilizar esse checklist, o desenvolvedor terá um guia de desenvolvimento das atividades no *front-end*. As vantagens em se utilizar uma ferramenta de qualidade é que os processos serão padronizados pela equipe, haverá diminuição de retrabalho, alinhamento com levantamento de requisitos, entre outros.

AVANÇANDO NA PRÁTICA

CERTIFICAÇÕES EM QUALIDADE DE SOFTWARE

Os colegas de um treinamento que você estava fazendo, no horário do almoço, estavam comentando acerca de certificações em tecnologia da informação. Em dado momento, um dos participantes disse que estava muito interessado em certificações ligadas à qualidade de software, mas que não sabia por onde começar.

Parece que o interesse de um dos colegas era o mesmo da maioria do grupo. Para que fosse possível organizar o caminho para tirar uma certificação, cada um dos interessados ficou com uma das tarefas de levantamento de informações. Você ficou responsável pela pesquisa de, pelo menos, cinco certificações e seus respectivos objetivos.

Esse material, no final da semana, será compilado e, então, será possível que todos os colegas escolham uma certificação e assim formem um grupo de estudos.

RESOLUÇÃO



As certificações em ferramentas de garantia da qualidade são uma forma de provar ao empregador que o profissional possui competências e habilidades para operacionalizar recursos de qualidade de software em atividades de desenvolvimento. Ter um grupo de estudos para auxiliar nessa jornada parece ser uma boa alternativa. Mas, para isso, as atividades devem ser divididas.

Dessa forma, algumas certificações com grande grau de relevância no meio profissional podem ser observadas a seguir:

- **CTFL (*Certified Tester Fundation Level*)**: essa certificação habilita o profissional a realizar testes de software.
- **TMAP (*Test Management Approach*)**: certificação com foco na interação entre o desenvolvimento e os testes.
- **CSQE (*Certified Software Quality Engineer*)**: trata-se de uma certificação da área de desenvolvimento e implementação de qualidade de software.
- **CSQA (*Certified Software Quality Analyst*)**: nessa certificação, o profissional terá uma atuação nos fundamentos da qualidade de software de forma mais analítica. É conhecido como grupo avançado de qualidade.
- **CBTS (*Certificação brasileira de teste de software*)**: é uma certificação que dá ênfase aos padrões de qualidade em software no Brasil.

Essas são algumas certificações que o mercado de trabalho valoriza. Certamente, é de se imaginar que existem muitas outras. Existem algumas que atendem a necessidades bem específicas, sendo

interessante para o momento em que a área de atuação profissional estiver bem consolidada.

NÃO PODE FALTAR

QUALIDADE DE PRODUTO

Sergio Eduardo Nunes

Ver anotações

COMO GARANTIR A QUALIDADE DO PRODUTO DE SOFTWARE?

Através de algumas ferramentas, conhecidas como modelos de qualidade de produto, para a avaliação do produto de software e de normas para garantir a padronização internacional do desenvolvimento de um software.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

PRATICAR PARA APRENDER

Caro aluno, a Engenharia de Software é uma área na qual são discutidos assuntos como gerenciamento, qualidade, processos, entre outros.

Dentro da qualidade existem discussões com especificidades, como é o

caso da qualidade do produto. Com isso, você deve imaginar que essas discussões tendem à área de desenvolvimento de software.

Onde nós podemos ver a qualidade do produto de software no dia a dia? Diversos sistemas, que utilizamos nos computadores, nos smartphones, nos sistemas embarcados, etc., foram validados em todas as suas características. Tendo isso em vista, as normas de qualidade de produto são um guia para que os processos, as avaliações e as métricas, sejam ajustadas conforme as necessidades do projeto.

Inicialmente, serão discutidos os modelos de qualidade de produto de software quanto a suas características, necessidades e aplicações, o que é de grande importância para que você possa, posteriormente, compreender as estruturas das normas.

Em seguida, serão discutidas as normas ISO/IEC (NBR), suas características e subcaracterísticas. E, embora toda essa estrutura seja complexa, você terá exemplos de momentos em que são aplicadas as técnicas durante o ciclo de vida do projeto.

Com isso, o próximo passo é compreender como as métricas são utilizadas para aferir a qualidade do produto de software, sendo possível que você entenda como encontrar os pontos que necessitam de avaliações e quais medidas podem ser utilizadas para tal finalidade.

Finalmente, serão abordados assuntos relacionados à gestão da qualidade. Esse assunto é de extrema importância, uma vez que a operacionalização da qualidade do produto de software necessita de metodologia e/ou técnicas que auxiliem os gestores de projeto nas atividades e que visam proporcionar ao usuário um produto com garantia de qualidade, integridade e segurança.

No tocante à atuação profissional, os assuntos discutidos nesta seção são de extrema importância, pois, em grande parte das empresas de desenvolvimento de software, são utilizados modelos e normas de qualidade como guia de qualidade de produto. Além do mais,

compreender como as métricas são utilizadas para auxiliar na avaliação da qualidade de produto de software é uma atividade necessária na área de desenvolvimento.

A empresa na qual você trabalha acabou se especializando em web rádio após ter desenvolvido um projeto desse tipo de veículo de comunicação para os times que estavam perdendo patrocinadores devido à falta de transmissão dos jogos em diferentes mídias. A fim de aumentar sua visibilidade, o time da cidade solicitou que fosse desenvolvida uma web rádio, que acabou se popularizando e despertando o interesse de outros times espalhados pelo território nacional, os quais enfrentam a mesma dificuldade. Isso fez com que a quantidade de projetos de desenvolvimento de software de web rádio tenha disparado nos últimos meses.

A empresa em que você trabalha apresenta a seguinte metodologia para encontrar novos clientes: o consultor comercial conhece as empresas, fica algumas semanas nelas, identifica os problemas, apresenta-os aos gestores e oferece-lhes soluções computacionais. Quando aprovado pela empresa, entra em cena um segundo time. Este fica responsável pelo levantamento de requisitos, momento em que o gerente de projetos começa a estreitar o relacionamento com os colaboradores da empresa que de alguma forma estão relacionados com os processos os quais serão objeto de desenvolvimento. Esse método fez com que o projeto da web rádio atendesse o time da cidade de forma personalizada.

A equipe de consultores comerciais vem fazendo ações que têm gerado muitos contratos de trabalho junto aos times de futebol. A empresa em que você trabalha fez algumas contratações, porém os prazos e a demanda de trabalho têm ocasionado queda na qualidade do produto de software. O gerente de projetos identificou a ocorrência sistemática de diversas falhas.

Preocupado com isso, o seu gerente de projetos buscou um colega de graduação, que deu a dica de utilizar a estrutura da ISO 9126 como base para a qualidade do produto de software. Após ler um pouco, decidiu que passaria a utilizá-la nos projetos voltados à web rádio.

Por saber que você está estudando essa disciplina na faculdade, ele o chamou para uma reunião. No início dela, foram apresentadas a você todas as situações negativas referentes à qualidade de produto de software com relação aos desenvolvimentos da web rádio pelas quais a empresa estava passando. Num segundo momento, foi-lhe explicado que a ISO 9126 seria utilizada como base para resolver esse problema e, em seguida, perguntaram-lhe como isso poderia ser operacionalizado.

Você, de pronto, sugeriu o desenvolvimento de uma interface web, a qual os desenvolvedores de *front-end*, *back-end*, banco de dados, tester e demais áreas poderiam utilizar para preencher as atividades e registrar um histórico quanto às características das funcionalidades.

A ideia foi genial e, por isso, muito elogiada e aprovada por todos que estavam na reunião. Depois de tantos elogios, o problema é que essa solução proposta foi atribuída a você mesmo, pois os gestores não encontraram nenhum outro colaborador com habilidades e competências para apresentar uma solução interessante.

Com isso, você pode utilizar recursos como slides ou desenvolvimento web (Bootstrap, HTML, CSS ou qualquer outra tecnologia) a fim de criar uma interface para o acompanhamento das atividades de desenvolvimento segundo as características da ISO 9126. Essa é uma ótima oportunidade de mostrar a sua potencialidade aos gestores e alcançar novos objetivos profissionais.

Você deve imaginar a quantidade de conhecimentos importantes na área de qualidade de produto de software. Então vamos lá para mais uma leitura muito interessante acerca de Engenharia de Software.

Caro aluno, você já deve ter baixado algum aplicativo em seu smartphone que prometia fazer determinada função, porém ao utilizá-lo, sua operacionalização era tão complicada que o resultado gerado não chegava nem perto do prometido. Ou seja, o produto final do desenvolvedor do *app* estava longe da qualidade prometida na descrição disponível na loja de aplicativos. A área da Engenharia de Software que trata desses assuntos é conhecida por qualidade de produto de software.

Certamente, quando nos referenciamos ao software como um produto, para alguns profissionais pode soar um pouco estranho. Porém, os assuntos relacionados à qualidade de produtos possuem uma grande abrangência, perpassando discussões como: os modelos de qualidade, as normas ISO de qualidade, a gestão da qualidade do produto, as medições, os requisitos e as avaliações da qualidade. Assim, devido à complexidade apresentada, tais conceitos serão exemplificados.

Segundo Mello (2010), ao contrário dos produtos manufaturados, o desenvolvimento de softwares é projetado. Os processos de desenvolvimento são criativos e técnicos e exigem que os profissionais da área apresentem competências e habilidades para executar suas funções. Já a produção de produtos manufaturados, em sua grande maioria, possui processos produtivos mecanizados, que ocorrem conforme a automação projetada pelo engenheiro. A seguir, conforme Mello (2010), é exemplificada a diferença em termos práticos:

- **Setor de manufatura:** são necessárias matérias-primas, processos de transformação para elas, embalagens, acondicionamento e transporte. E seu sistema produtivo pode ter diversos níveis de complexidade. Exemplo: uma padaria especializada em bolos possui determinadas matérias-primas envolvidas em seus processos produtivos e bolos com diferentes níveis de complexidade. Já uma empresa farmacêutica tem as características de seu setor produtivo construídas a partir de seus diversos insumos e níveis de processos.

- **Setor de desenvolvimento:** em grande parte dos desenvolvimentos, em termos de recursos computacionais, são necessários: computadores, acesso à internet e programas. Porém, para o bom aproveitamento dos recursos, grande parte do projeto vai depender das habilidades, competências e da criatividade dos profissionais que utilizam o desenvolvimento de software para a busca de soluções em diversas áreas do conhecimento.

Você percebeu o quanto é complexo garantir a qualidade do produto de software? Pois bem, para tal existem algumas ferramentas, conhecidas como modelos de qualidade de produto, onde podem ser avaliado o produto de software. Normas foram padronizadas internacionalmente a fim de garantir que, independentemente do local onde se esteja desenvolvendo um software, algumas premissas sejam seguidas.

| ISO 9126 (NBR 13596)

De acordo com Wazlawick (2013), a ISO 9126 (NBR 13596) é a norma que visa avaliar a qualidade, as características e os atributos da qualidade de software. Além disso, a norma tem como objetivo a padronização das atividades e da forma de se avaliar a qualidade do produto, a fim de se gerar feedbacks importantes para a equipe de desenvolvimento de software.

A estrutura da norma ISO 9126 é feita em quatro partes, conforme pode ser observado a seguir:

- **ISO 9126-1 de 2001:** trata das características, subcaracterísticas e métricas da qualidade de produto de software (tema desta seção).
- **ISO 9126-2 de 2003:** trata das métricas externas e do controle de falhas.
- **ISO 9126-3 de 2003:** o seu objetivo é verificar a quantidade de ocorrências de falhas e estimar o tempo de recuperação.

- ISO 9126-4 de 2004: faz as tratativas de *User Experience*, produtividade, eficácia e segurança.

Embora o foco da seção seja a ISO 9126-1, é inevitável que processos de outras ISO não sejam mencionados, conceituados e exemplificados. Isso porque, em termos práticos, as normas são complementares dentro de um projeto de desenvolvimento de software.

Ainda conforme Wazlawick (2013), define-se que a ISO 9126 faz parte da família de normas de qualidade 9000. O foco dela é a qualidade de produto de software. No Brasil a NBR 13596 é equivalente à ISO, porém houve uma substituição da NBR pela ISO/IEC 9126-1. A norma constante na ISO 9126 é composta por características, subcaracterísticas e métricas. As seis características podem ser observadas na Figura 2.7.

Figura 2.7 | Características da ISO 9126



Para melhor compreensão das características e subcaracterísticas da ISO 9126, a seguir vamos ver o detalhamento de cada uma delas segundo Wazlawick (2013).

Funcionalidade

Descreve os atributos das funções contidas nos softwares. Possui as subcaracterísticas:

- **Adequação:** diz respeito ao alinhamento do funcionamento correto da funcionalidade.
- **Acurácia:** preocupa-se com as saídas geradas pelo software, que têm de estar de acordo com as necessidades.
- **Interoperabilidade:** denota a capacidade do software de poder ser utilizado por tecnologias diferentes.
- **Conformidade:** deve estar de acordo com normas, regras e leis específicas.
- **Segurança de acesso:** capacidade de evitar intrusões e incidentes relacionados à segurança da informação.

Para que você compreenda a forma de identificar as funcionalidades dentro de um projeto, analise a seguinte situação: uma empresa precisa de um chat para que os colaboradores se comuniquem dentro da rede local, porém existe a necessidade de se utilizar o protocolo IP nas duas versões: IPv4 e IPv6. Conforme pode ser observado, a funcionalidade de comunicação interna possui características de grande relevância quanto à adequação e à interoperabilidade. Ainda de acordo com Wazlawick (2013):

Confiabilidade

É a capacidade do software de se manter em funcionamento e com o desempenho esperado/estabelecido. Observe as subcaracterísticas:

- **Maturidade:** demonstra a frequência com que ocorrem as falhas de software.
- **Tolerância a falhas:** preocupa-se em verificar, na ocorrência de falhas ou em incidentes relacionados à segurança (falha provocada ou por violação), quais serviços permanecerão acessíveis e com a garantia da integridade.
- **Recuperabilidade:** trata-se do tempo em que, após a ocorrência de determinada falha, a funcionalidade ou sistema estará disponível novamente.

Os aspectos relacionados à confiabilidade estão mais presentes em nossas atividades cotidianas, sendo, portanto, mais fácil de identificá-las. Por exemplo: quando os aplicativos de stream utilizados em smart TVs foram projetados, tornou-se necessário conhecer a sua confiabilidade, o que se deu a partir de questionamentos do tipo: em caso de falha, o filme, ou a série, será interrompido? Se caso ocorrerem, quais serão essas falhas? Qual o tempo para o sistema se reestabelecer?

Usabilidade

Trata-se de uma forma de aferir o nível de experiência do usuário (User Experience) quanto à facilidade de operacionalizar o software. As subcaracterísticas podem ser observadas a seguir:

- **Inteligibilidade:** está relacionada com a lógica de sua aplicabilidade ser intuitiva e de fácil operação.
- **Apreensibilidade:** afere o quanto o usuário se esforçou para aprender a utilizar uma funcionalidade ou o software como um todo.
- **Atratividade:** observa o quanto as interfaces do software despertam a atenção do usuário.

A usabilidade a nível de usuário é uma das características que requer mais atenção. É muito comum alguns usuários instalarem softwares de edição de vídeo e, devido à complexidade para fazer os tratamentos,

acabarem por desistir e buscar uma solução para smartphone. Isso está totalmente relacionado às características de usabilidade do software. Continuando com o que explica Wazlawick (2013):

Eficiência

Refere-se à forma como o software irá se comportar dentro dos parâmetros estabelecidos no projeto. Suas subcaracterísticas são:

- **Tempo:** preocupa-se em medir o tempo de resposta de uma funcionalidade ou ainda a latência e/ou jitter.
- **Recursos:** trata-se de uma forma de aferir o quanto os recursos são utilizados ao se fazer uso de determinada funcionalidade ou do software como um todo.

Manutenibilidade

A preocupação, nessa característica, está na capacidade de modificação, que é possível nos seguintes casos: excluir defeitos e falhas, adicionar novas funcionalidades, adaptar a novas plataformas ou a sistemas operacionais, etc. As subcaracterísticas estão apresentadas a seguir:

- **Modificabilidade:** trata da capacidade de modificar o software por algum motivo ou necessidade.
- **Estabilidade:** preocupa-se em verificar se algumas falhas ocorrerão após as modificações.
- **Escalabilidade:** faz a aferição da capacidade de crescimento do software a fim de se atender uma demanda maior.

A manutenção dos softwares é algo que preocupa gerentes de projetos, pois a falha na projeção das subcaracterísticas impacta diretamente a qualidade dos serviços.

Portabilidade

Em 2020, o governo federal efetuou ajuda financeira à população por meio de um benefício (devido à alta taxa de desemprego causada pela pandemia do novo coronavírus). Porém, para ter acesso a esse

benefício, era necessário fazer a instalação de um aplicativo no smartphone. Ocorre que o aplicativo não havia sido projetado para atender uma parcela tão grande da população e, dessa forma, seu funcionamento ficou muito comprometido. Segundo Wazlawick (2013), a portabilidade torna-se um ponto de grande relevância, com a advento dos dispositivos móveis, e suas características podem ser observadas a seguir.

A portabilidade diz respeito a um conjunto de características que demonstra a capacidade do software ser utilizado em outros sistemas, dispositivos e plataformas.

- **Adaptabilidade:** capacidade de se adaptar a ambientes nos quais não foram projetados.
- **Analisabilidade:** deve-se analisar os impactos positivos e negativos que a utilização do software em outros meios pode ocasionar.
- **Interoperabilidade:** demonstra a capacidade de interação com outros sistemas, muitas vezes desenvolvidos com outras tecnologias e arquiteturas.

Um exemplo clássico e muito atual da portabilidade ocorreu quando os dispositivos do tipo smartphone se popularizaram. Os sites, até então, estavam projetados para serem exibidos em monitores com um tamanho muito maior do que a tela de um smartphone. Isso fez com que os sites tivessem um comportamento muito estranho nesses dispositivos, ou seja, eles não estavam preparados para a portabilidade.

ASSIMILE

Com a popularização dos dispositivos móveis a característica portabilidade, existente na norma ISO 9126, tornou-se mais que uma necessidade, veio a ser uma obrigação. Para isso, algumas linguagens de programação/marcação possuem frameworks que fazem tratativas quanto à responsividade das aplicações.

Um exemplo de linguagem de marcação que provê essa portabilidade é o Bootstrap. Como não existe um site oficial com versão em português para o Bootstrap, utilize o tradutor do navegador ao visitar o site *W3Schools*.

BOOTSTRAP 3 Tutorial. **W3Schools**, [S.I., s.d.].

CONJUNTO DE MÉTRICAS

Caro aluno, caso você não se recorde, no início desta seção, foi dito que a ISO 9126 é composta por características, subcaracterísticas e métricas. De acordo com Wazlawick (2013), a análise das características e das consequentes subcaracterísticas só é possível por meio das métricas, que equalizarão o desenvolvimento. A sua estrutura possui três fases e pode, ainda, ser aplicada em qualquer momento no ciclo de vida do projeto. Observe o conjunto de métricas:

Definição dos requisitos de qualidade

É a definição das características e subcaracterísticas por parte do cliente. Com isso, é possível definir quais serão as métricas utilizadas para a avaliação do produto. Para exemplificar, imagine que uma equipe seja contratada para organizar um campeonato de Counter Strike. O requisito é que a rede suporte vinte e cinco jogadores em cada time (cinquenta ao todo) e uma latência abaixo de 0,350 ms. Ou seja, na contratação foi definido, segundo uma quantidade de jogadores, a latência esperada para se atender a qualidade, por meio da métrica de 0,350 ms.

Preparação da avaliação

Nesse momento devem ser estabelecidos os critérios de avaliação, que podem ser numéricos (com escala de 0 a 10) ou, ainda, como orienta a norma: insatisfatório, razoável, bom e excelente. A avaliação pode ser aplicada nas funcionalidades desenvolvidas, mostrando-se como uma alternativa mais eficiente para aplicação no projeto como um todo.

Para exemplificar a preparação da avaliação, imagine que se tenha um sistema de venda em que o usuário adicione os produtos no carrinho de compras, escolha a forma de pagamento, o local de entrega e encerre o pedido. Quanto à experiência do usuário ao utilizar a funcionalidade de compra, foi elaborado o Quadro 2.1, como demonstrado a seguir.

Quadro 2.1 | Exemplo de avaliação

Parâmetro de rede	Conceito
Sistema de busca do produto	Espaço onde entra o conceito.
Lista de apresentação dos produtos	Espaço onde entra o conceito.
Sistema de seleção do produto	Espaço onde entra o conceito.
Opção de escolha da quantidade	Espaço onde entra o conceito.
Interface do carrinho de compras	Espaço onde entra o conceito.
Sistema de seleção de pagamento	Espaço onde entra o conceito.
Sistema de escolha de entrega	Espaço onde entra o conceito.

Fonte: elaborado pelo autor.

A preparação permite a reflexão sobre os pontos necessários e importantes que devem ser medidos. Além disso, os feedbacks gerados possibilitam verificar se as métricas adotadas para a avaliação dos componentes e funcionalidades estão adequadas.

Avaliação

Trata-se do momento em que os sistemas de avaliação das funcionalidades serão aplicados para posterior análise e aplicação de medidas corretivas.

- **Medida:** são aquelas métricas definidas na preparação da avaliação. É possível, ainda nesse momento, verificar se as métricas estão

adequadas (validação).

- **Pontuação:** verificar se o sistema de pontuação se adequa a uma forma de avaliação justa.

Percebeu como a aplicação da norma propriamente dita é simples? Ela ainda é tida pelos desenvolvedores como flexível e de simples adaptação para qualquer tipo de projeto de desenvolvimento. Sua metodologia é bem definida, fazendo com que os procedimentos sejam fáceis de serem implementados nos desenvolvimentos de software. Para melhor compreensão dos passos a serem seguidos no processo como um todo, observe o diagrama representado na Figura 2.8.

Figura 2.8 | Diagrama de definição das métricas



Fonte: adaptada de Wazlawick (2013).

Vale lembrar que o modelo de processo de avaliação descrito, bem como as tratativas em relação às métricas aqui discutidas, refere-se à ISO 9126.

EXEMPLIFICANDO

Neste momento você deve estar apreensivo devido à quantidade de pontos que devem ser observados para se utilizar os modelos de qualidade de produto de software, não é? Então, imagine que você precise colocar em prática, na empresa em que você trabalha, a ISO 9126. Certamente virá a dúvida: por onde começar?

A equipe deve elaborar um documento (o formato dele pode variar conforme a empresa, não existe um template), no qual o objeto a ser avaliado é pontuado dentro de todas as suas características e subcaracterísticas. Por exemplo: a equipe de desenvolvimento fez uma tela de Fale Conosco. Nesse tipo de

página, normalmente há contatos, localização, textos informativos e uma área para se mandar uma mensagem. A equipe deve utilizar todos os pontos da ISO 9126 para avaliar cada uma das características e subcaracterísticas presentes. Isso pode ser organizado textualmente em quadros, tabelas, formulários, etc.

ISO 9000

Pois bem, você percebeu que os métodos aqui discutidos são todos apoiados em normas. Grande parte dos sistemas de gestão de qualidade são baseados nas normas ISO 9000:2015 (ABNT, 2015), que têm como função:

- Descrever os fundamentos e princípios da gestão da qualidade.
- Compreender os processos de implementação da gestão da qualidade.
- Avaliar a conformidade dos produtos de software desenvolvido.

Segundo Carpinetti e Gerolamo (2019), a ISO 9000 segue oito princípios de gestão da qualidade, os quais têm como objetivo conduzir os gestores na melhoria de desempenho, principalmente em atividades relacionadas a desenvolvimento de software. Observe a seguir a descrição desses princípios:

- **Foco no cliente:** uma abordagem por meio da qual se buscam melhores práticas, a fim de entregar o melhor produto.
- **Liderança:** metodologia e abordagens como forma de liderar.
- **Pessoas:** utilizar formas de as pessoas se comprometerem com os processos e com a qualidade.
- **Processos:** verificar constantemente os processos e repensá-los.
- **Inter-relacionamento:** prover o inter-relacionamento de atividades concorrentes.

- **Melhoria:** buscar a melhoria contínua por meio de metodologias, normas e boas práticas.
- **Decisão:** utilizar os feedbacks gerados a favor da tomada de decisão.
- **Benefícios:** gerar vantagens administrativas e operacionais por meio da adoção de boas práticas.

Para que você possa compreender como os princípios da ISO 9000 estão relacionados às atividades de desenvolvimento no mercado de trabalho, observe o exemplo descrito a seguir. Imagine que se tenham “ilhas” de desenvolvimento dentro de uma organização, o que faz com que os times sejam separados por conhecimento e habilidades. Porém, há um momento dentro do projeto em que as funcionalidades desenvolvidas precisam ser integradas para que o sistema de fato passe a existir.

Como fazer isso?

A ISO 9000 possui tratativas para esses fins. Observe o quanto a norma pode auxiliar nesse exemplo e o quanto trará impactos positivos aos profissionais. A norma apresenta guias quanto à abordagem a ser realizada com a equipe, quanto à liderança que orienta a forma de se garantir a integração das funcionalidades e, sob um aspecto diferente, quanto a um olhar que mapeia os processos a fim de propor melhorias e ajustes. Isso permite um planejamento mais adequado e gera benefícios a curto e longo prazo (conforme o aumento da maturidade).

ISO 9001

Para finalizar, vale a pena citar que a norma 9001:2015 faz uma abordagem com caráter de tarefas administrativas no tocante ao sistema de gestão da qualidade (SGQ). Segundo o catálogo da ISO 9001:2015 (ABNT, 2015), seus objetivos são:

- Fazer o controle documental.
- Efetuar o controle de registro da qualidade.
- Normatizar a auditoria interna.

- Fazer o controle de produtos que não atendam às conformidades.
- Prover ações corretivas.
- Prover ações preventivas.

EXEMPLIFICANDO

O controle documental é uma preocupação tanto de gerentes de projetos quanto de desenvolvedores. Imagine que, após escrever uma funcionalidade de modos de pagamento para compra de passagens aéreas, um outro desenvolvedor subscreva a sua versão no sistema. Todo o seu trabalho estará perdido. Para isso, a ISO 9001:2015 possui algumas tratativas que podem ser facilmente alinhadas por meio de sistemas de versionamento de códigos.

Uma das aplicações web para versionamento que mais se popularizou entre desenvolvedores foi o GitHub.

Caro aluno, ao longo das discussões levantadas nesta seção, você pôde perceber como as normas são um ótimo guia para as atividades de desenvolvimento de software, não é mesmo? Para alguns profissionais, o uso dessas ferramentas parece ter o intuito de agregar mais uma atividade além daquelas que precisam ser executadas dentro de um projeto. Mas o objetivo é, na verdade, o oposto. A intenção é que, pela utilização das normas, as atividades sejam executadas de forma mais assertiva, fazendo com que menos processos e atividades de desenvolvimento precisem ser revisitadas.

REFLITA

Os projetos de desenvolvimento de software são compostos por atividades extremamente importantes, e as normas visam ser um balizador dos processos que devem ser seguidos para atingir determinados objetivos. Como observado, as normas ISO 9126, ISO 9000, ISO 9001, entre

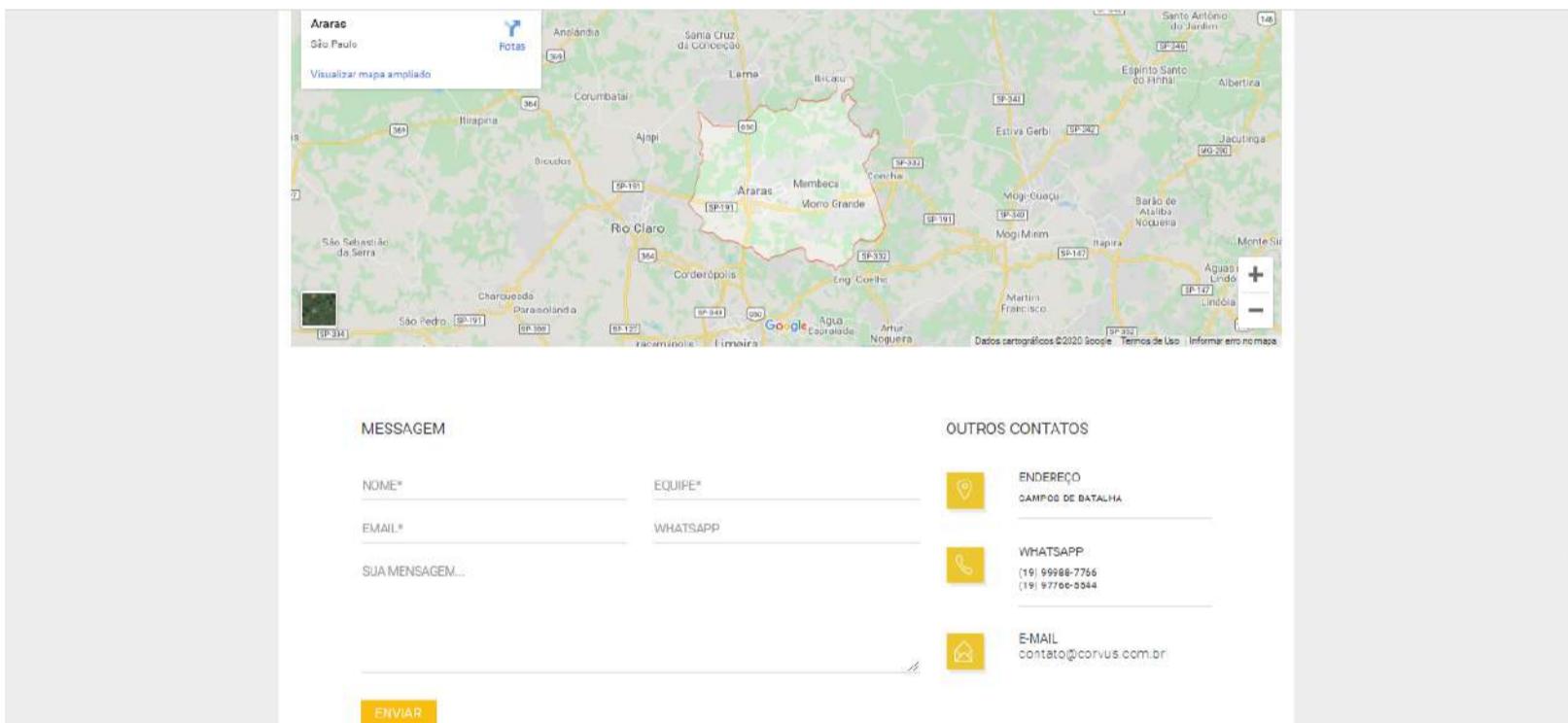
outras, são direcionadoras de qualidade para o produto de software. Porém, faz-se realmente necessário utilizar todos os seus apontamentos? Mesmo que eles engessem a sua operacionalização nos projetos.

FAÇA VALER A PENA

Questão 1

O Bootstrap é um framework que ajusta a responsividade de desenvolvimentos web, como pode ser observado em uma página de formulário de contato demonstrado na figura a seguir.

Figura | Site responsivo



Fonte: captura de tela do Google Maps elaborada pelo autor.

Quando o framework não é utilizado, os efeitos negativos no que se refere à qualidade do produto podem ser observados na figura seguinte:

Figura | Site com falha de responsividade

The screenshot shows a Google Maps interface with a form overlaid. At the top, there are standard map controls like zoom and orientation. Below them is a header bar with the Google logo and links for 'Dados do mapa', 'Termos de Uso', and 'Informar erro no mapa'. The main area contains a form with several fields:

- E*
- PE*
- IL*
- TSAPP
- MENSAGEM...

A yellow progress bar at the bottom indicates the message is being processed.

Fonte: captura de tela do Google Maps elaborada pelo autor.

Por isso a importância de ferramentas de qualidade de produto de software em atividades de desenvolvimento.

Com base no case apresentado, assinale a alternativa que descreva a característica correta, segundo a norma ISO 9126.

a. Funcionalidade.

b. Usabilidade.

c. Eficiência.

d. Portabilidade.

e. Confiabilidade.

Questão 2

As métricas são utilizadas de forma intuitiva em diversas situações no dia a dia, veja alguns exemplos:

"Pode ir almoçar naquele restaurante, pois lá a comida e o atendimento são muito bons".

"A temperatura do nosso filho está 36,6º C. Não está com febre".

"A latência desse jogo está marcando 1,250 ms em média, está muito alta para jogar on-line".

Na qualidade de produto, as métricas são utilizadas como um direcionador da avaliação. Com base nisso, analise as afirmativas a seguir:

- I. Na fase de definição dos requisitos de qualidade, o funcionamento esperado das características e subcaracterísticas são determinadas entre as partes.
- II. Na fase de preparação da avaliação, são feitos testes para verificar se as métricas escolhidas irão gerar resultados confiáveis.
- III. Na fase de avaliação, são determinados os critérios de avaliação, que podem ser numéricos ou conceituais.

Com base nas afirmativas apresentadas, assinale a alternativa correta.

- a. Está correta apenas a alternativa I.
- b. Está correta apenas a alternativa II.
- c. Está correta apenas a alternativa III.
- d. Estão corretas apenas as alternativas I e II.
- e. Estão corretas apenas as alternativas II e III.

Questão 3

Quanto à questão da gestão da qualidade, a ISO 9000 é uma importante ferramenta, principalmente para gerentes de projetos, pois serve como um guia de atividades com oito princípios da gestão da qualidade.

A partir do apresentado, analise as asserções a seguir e a relação proposta entre elas.

- I. O foco da gestão descrita na ISO 9000 deve ser o cliente.

POIS

II. Isso promoverá a melhoria nos processos, permitindo rever a forma de execução das atividades.

A seguir, assinale a alternativa correta:

- a. As asserções I e II são proposições verdadeiras e a II é uma justificativa correta da I.
- b. A asserção I é uma proposição verdadeira e a asserção II é uma proposição falsa.
- c. As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- d. A asserção I é uma proposição falsa e a II é uma proposição verdadeira.
- e. As asserções I e II são proposições falsas.

REFERÊNCIAS

ABNT. **NBR ISO 9000:2000**: Sistemas de gestão da qualidade – Fundamentos e vocabulário. São Paulo: ABNT, 2000. 26p. Disponível em: <https://bit.ly/3981W31>. Acesso em: 3 out. 2020.

ABNT. **NBR ISO 9000:2000**: Sistemas de gestão da qualidade – Fundamentos e vocabulário. São Paulo: ABNT, 2002. 26p. Disponível em: <https://bit.ly/3nob6gS>. Acesso em: 7 out. 2020.

ABNT. **NBR ISO 9000:2000**: Sistemas de gestão da qualidade – Fundamentos e vocabulário. São Paulo: ABNT, 2005. 35p. Disponível em: <https://bit.ly/3s2CuV8>. Acesso em: 7 out. 2020.

ABNT. **NBR ISO 9000:2015**: Sistemas de gestão da qualidade – Fundamentos e vocabulário. São Paulo: ABNT, 2015. 59p. Disponível em: <https://bit.ly/2LrYbNT>. Acesso em: 3 out. 2020.

BOOTSTRAP 3 Tutorial. **W3Schools**, [S.I., s.d.]. Disponível em: <https://bit.ly/2JVNBOM>. Acesso em: 19 out. 2020.

CARPINETTI, L. C. R.; GEROLAMO, M. C. **Gestão da Qualidade ISO 9000:2015**. São Paulo: Atlas, 2019.

MELLO, C. H. P. **Gestão da Qualidade**. São Paulo: Pearson Education do Brasil, 2011.

WAZLAWICK, R. S. **Engenharia de software**: conceitos e prática. Rio de Janeiro: Elsevier, 2013.

FOCO NO MERCADO DE TRABALHO

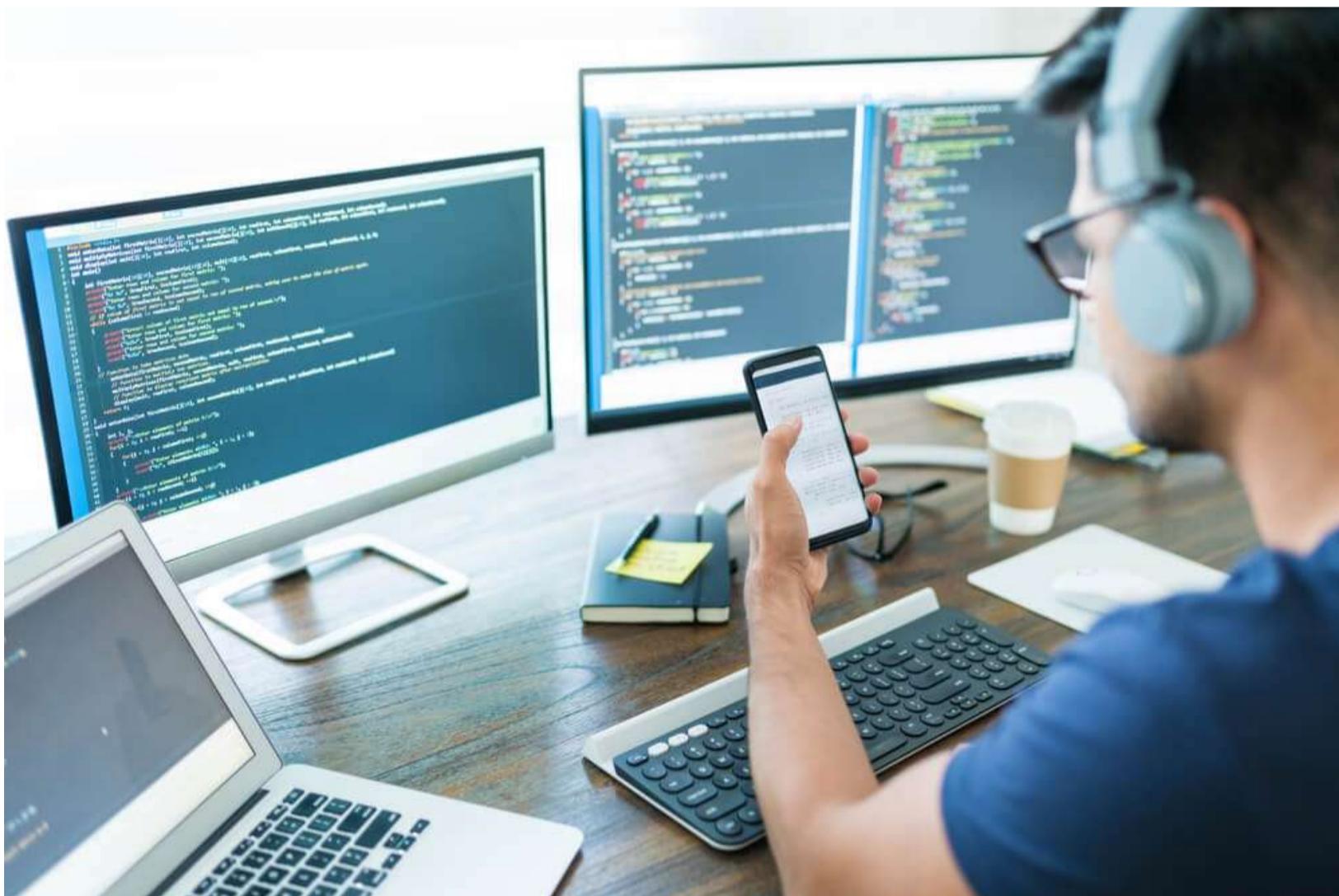
QUALIDADE DE PRODUTO

Sergio Eduardo Nunes

Ver anotações

O QUE ISO 9126?

A ISO 9126 é a norma que visa avaliar a qualidade, as características e os atributos da qualidade de software. Além disso, a norma tem como objetivo a padronização das atividades e da forma de se avaliar a qualidade do produto.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

A empresa em que você trabalha está com uma demanda muito alta de projetos devido ao desenvolvimento da web rádio, que ficou muito famosa entre diversos times de futebol do Brasil. Isso fez com que a demanda de desenvolvimento de novas aplicações para web rádio tivesse um impacto negativo na qualidade do produto de software.

Por esse motivo, os gestores convocaram uma reunião, para a qual você foi convidado a participar. Em dado momento dessa reunião, ao ser questionado, você sugeriu o desenvolvimento de uma interface para o acompanhamento das atividades de desenvolvimento dos web rádios, segundo as características da ISO 9126. Os gestores gostaram muito da ideia e atribuíram a você o desenvolvimento de uma solução para se colocar em prática a ISO 9126 em prol das atividades de desenvolvimento da empresa. Inicialmente, observe a interface de uma das características dessa norma na Figura 2.9.

Figura 2.9 | Interface para funcionalidade

EMPRESA DE DESENVOLVIMENTO DE SOFTWARE

FUNCIONALIDADE

Descrição	<input type="text"/>	
Positivo	<input type="text"/>	
Negativo	<input type="text"/>	
Data Início	<input type="text" value="dd/mm/aaaa"/> <input type="button" value="..."/>	Data Final <input type="text" value="dd/mm/aaaa"/> <input type="button" value="..."/>
Responsável	<input type="text"/>	Avaliação <input type="button" value="..."/>
<input type="button" value="Salvar"/> <input type="button" value="Cancelar"/>		

Fonte: captura de tela de software desenvolvido em PHP elaborada pelo autor.

Conforme pode ser observado, a interface se preocupa com o aspecto de usabilidade (ISO 9126). Além disso, pode-se verificar também que os dois pontos dessa característica foram atendidos:

- **Inteligibilidade:** os campos possuem nomes diretos e existe uma separação por categorias de informação.

- **Apreensibilidade:** foram utilizados elementos como box, drop down e seletor de datas, que tornam a sua operação muito intuitiva.

Para melhor compreensão da interface, serão apresentadas as partes que a compõem. Observe a Figura 2.10.

0

Figura 2.10 | Parte textual para funcionalidade

FUNCIONALIDADE	
Descrição	
Positivo	
Negativo	

Fonte: captura de tela de software desenvolvido em PHP elaborada pelo autor.

Vale ressaltar que, conforme a ISO 9126, no que tange às subcaracterísticas da funcionalidade, é possível observar:

- **Adequação:** o formulário possui os campos necessários, conforme solicitado.
- **Acurácia:** os dados são enviados a uma tabela de uma base de dados.
- **Interoperabilidade:** o formulário é responsivo, atendendo, assim, a diversos dispositivos.
- **Conformidade:** está de acordo com normas, regras e leis específicas, uma vez que não existe o preenchimento de algum dado que exponha o autor.
- **Segurança de acesso:** nesse desenvolvimento em específico, existem mecanismos de proteção de *SQL injection* (injeção de dados via URL no navegador) e validação dos campos no formulário.

Ver anotações

Nesse espaço, o desenvolvedor vai descrever a funcionalidade, os pontos positivos encontrados e os pontos negativos. A ideia é que se utilize esse espaço com uma mensagem clara e dialogada.

O próximo passo pode ser observado na Figura 2.11.

Figura 2.11 | Parte de datas para funcionalidade

Fonte: captura de tela de software desenvolvido em PHP elaborada pelo autor.

Para registrar as datas de início do desenvolvimento e a entrega de determinada funcionalidade, são utilizados os campos demonstrados na Figura 2.12.

Figura 2.12 | Registro de métrica para funcionalidade

Fonte: captura de tela de software desenvolvido em PHP elaborada pelo autor.

Para que seja registrado o responsável, existe um campo, e a funcionalidade deve receber uma atribuição avaliativa. Assim, a proposta da utilização da ISO 9126 não vai tomar muito tempo e será um direcionador das atividades de desenvolvimento tanto para os desenvolvedores quanto para os gerentes de projeto.

AVANÇANDO NA PRÁTICA

METODOLOGIA DE GESTÃO DA QUALIDADE DO PRODUTO

A empresa de desenvolvimento de games para mobile adotou as normas da ISO 9126 para ajustes de qualidade de produto. Isso fez com que, após alguns projetos, os resultados pudessem ser sentidos por

clientes, gestores de projetos e desenvolvedores. Porém, as atividades relacionadas à ISO 9126 têm se tornado uma prática diferente para cada gerente de projetos, causando desconfortos aos desenvolvedores quando são alocados para outros times de desenvolvimento.

Como você é o gerente de projetos com mais tempo de empresa, o gerente geral solicitou um relatório, no qual você deve propor uma solução viável e que, de preferência, não gere custos. Assim, espera-se que haja uma gestão das atividades relacionadas à qualidade do produto durante os projetos de desenvolvimento de software.

o

Ver anotações

RESOLUÇÃO



Ao mesmo tempo que a ISO 9126 ajudou essa empresa, ela tem se tornado um problema quando os colaboradores migram de uma equipe a outra, isso porque cada gerente de projetos tem utilizado uma forma de operacionalizar as normas ISO 9126. Para isso, o gerente geral solicitou que você desenvolvesse um relatório com um sistema de gestão de qualidade de produto. Dessa forma, foi feita a seguinte sugestão:

Vamos adotar a ISO 9000, pois a sua estrutura possui oito princípios de gestão da qualidade, os quais têm como objetivo conduzir os gestores nas atividades de desenvolvimento de software. Perceba como nós conseguiremos a padronização de trabalho entre os gerentes de projetos ao observar os princípios dessa norma: foco no cliente, liderança, pessoas, processos, inter-relacionamento, melhoria, decisão e benefícios.

NÃO PODE FALTAR

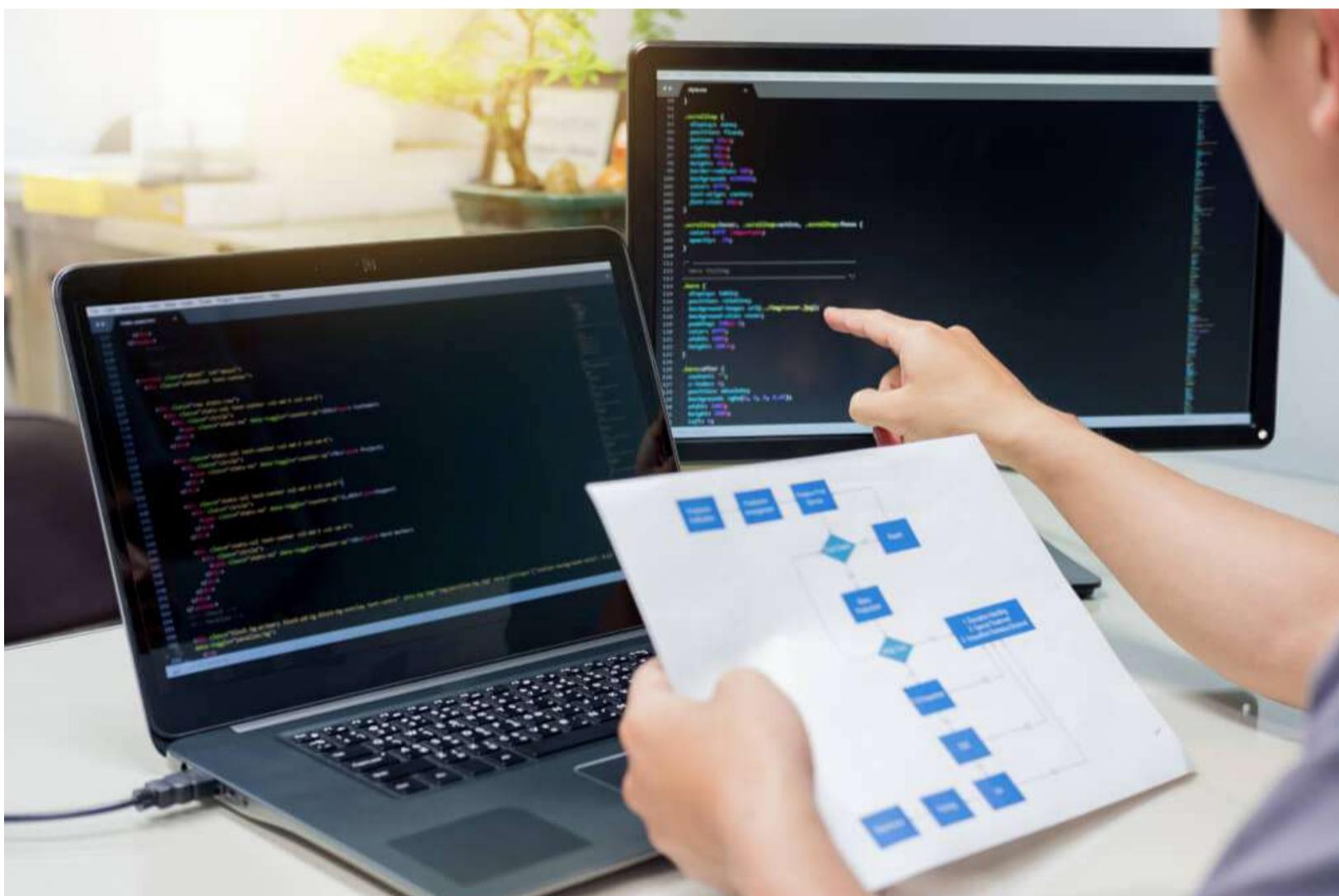
QUALIDADE DE PROCESSO

Sergio Eduardo Nunes

Ver anotações 0

O QUE SÃO MODELOS DE MATURIDADE DE PROCESSOS?

Os modelos de maturidade são ótimas ferramentas que permitem o conhecimento profundo dos processos, da equipe de desenvolvimento, das dificuldades, entre outros.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Caro aluno, muitas vezes nós, enquanto consumidores, estamos acostumados somente a ver o produto final e não os processos envolvidos no desenvolvimento dele. A ideia é a de que, quando você vai a um fast-food, a única parte do processo com a qual você tenha contato seja o momento de fazer e receber o pedido. Seja em um fast-food, seja em um setor industrial ou, no nosso caso, na indústria do software, na grande maioria das vezes, avaliamos se o processo foi bem executado considerando o resultado que é entregue. Mas como são feitas as tratativas referentes à qualidade dos processos de desenvolvimento de software?

Inicialmente, nesta seção serão discutidos os modelos de maturidade CMM e CMMI, os quais são metodologias que visam, por meio de sua estrutura, posicionar a empresa em um nível de maturidade e, através de técnicas, planejar uma evolução em termos organizacionais a fim de promover melhoria em seus processos.

Em seguida, haverá uma discussão acerca da ISO 9001:2015, cujo foco está em utilizar metodologias que permitam aos membros da organização conhecer os processos, os microprocessos e as interações entre os componentes em sua totalidade. Dessa forma, ao se conhecer bem os processos, a chance de erros e falhas é minimizada, proporcionando a melhoria dos processos e consequentes benefícios.

O objetivo do próximo tema é que você compreenda a metodologia para melhoria de processos individuais e de equipe (PSP). Trata-se de técnicas bem fundamentadas e voltadas às pessoas (desenvolvedores), pois quem, de fato, consegue abstrair informações e codificá-las são as pessoas.

Finalmente, será apresentado o modelo de melhoria de processo de software brasileiro (MPS.BR), ferramenta muito interessante, pois permite ao desenvolvedor compreender a potencialidade do método de

forma que possa ser aplicado em organizações brasileiras. Isso vem atender a uma demanda dentro da área de tecnologia da informação, na qual o MPS.BR atende as nossas especificidades.

Em termos profissionais, esses temas são de extrema importância pois são largamente utilizados dentro de organizações que buscam técnicas para atingir resultados expressivos. Dessa maneira, conhecer as técnicas e compreender o momento de aplicá-las pode ser um interessante diferencial competitivo.

A prefeitura de uma cidade litorânea em época de temporada, que compreende os meses de dezembro a fevereiro, contrata alguns profissionais temporários para limpeza, guarda-vidas, serviço de atendimento ao turista, entre outros servidores. Uma função muito importante nas últimas três temporadas foi o serviço de atendimento ao turista. Nessa função, os colaboradores ficavam grande parte do tempo ocupados em localizar os pais de crianças que se perdiam na praia.

Ocorre que, em alguns casos, a falta de um sistema integrado fazia com que esses colaboradores empregassem muito tempo para localizar os pais das crianças. Foi a partir dessa dificuldade que a prefeitura, por meio de uma concorrência pública, fez a contratação de uma empresa para desenvolver o sistema. Porém, entre as cláusulas para conseguir o contrato de desenvolvimento de software está uma classificação de maturidade dos processos a níveis aceitáveis.

Como você já possui vínculo com a prefeitura no que tange à consultoria em tecnologia da informação, no começo da semana, o prefeito da cidade solicitou-lhe um relatório técnico no qual apontasse o nível de maturidade em que a empresa contratada se encontra, para que, estando o nível de acordo, o contrato de prestação de serviço pudesse ser firmado.

A estrutura da empresa conta com parte familiar, que são os colaboradores administrativos, e parte de contratação em regime CLT, que são o gerente de projetos e demais desenvolvedores. Embora tenha

uma estrutura modesta, o portfólio da empresa é interessante, pois muitas prefeituras já contrataram seus serviços de desenvolvimento.

A metodologia segue o esquema: a demanda é originada pelo gerente de projetos e os desenvolvedores realizam o trabalho. O processo de integração das partes desenvolvidas por diferentes profissionais geralmente é um pouco demorado, pois é necessário um esforço coletivo para correção de falhas e erros, além da realização de complementos e ajustes severos às vezes.

Dentro do contexto da empresa, você deverá elaborar um relatório, para o prefeito, que descreva o nível de maturidade em que a empresa está posicionada atualmente. Para isso, faz-se necessária a aplicação de um modelo de maturidade. Como sempre, o chefe do executivo espera de você um relatório com a qualidade de sempre. Mão à obra!

DICA

Percebeu como conhecer os processos é muito importante? Porém, não é possível adquirir esses conhecimentos sem as técnicas adequadas. Assim sendo, vamos dar continuidade ao seu crescimento profissional? Bons estudos!

CONCEITO-CHAVE

Caro aluno, quando você está utilizando o aplicativo para consultar sua área acadêmica, buscando informações como notas, dados pessoais, financeiros, entre outros, talvez não reflita sobre a quantidade de processos necessários para que se possa utilizar uma aplicação com confiabilidade, eficiência e segurança. Para isso, são utilizadas normas, métodos, ferramentas e metodologias que garantem a qualidade dos processos de software.

Sommerville (2015) define que, nas atividades de desenvolvimento de software, são necessários processos como uma forma de se organizar, gerenciar e compreender a ordem em que as atividades são executadas. Com o intuito de que você compreenda melhor os processos, vamos

tomar como exemplo um projeto de desenvolvimento da página Home de um site. Nesse exemplo poderiam ser observados os processos: design de layout, tratamento de imagens, responsividade, entre outros. Além disso, é possível notar que cada etapa do desenvolvimento pode ser orientada por uma norma a fim de que atinja os objetivos desejados.

E por que a melhoria dos processos é importante para as atividades de desenvolvimento de software? Num primeiro momento pode parecer que a melhoria de processos só pode ser aplicada em atividades de linhas de produção, mas não é bem assim. Segundo Sommerville (2015), ao se utilizar ferramentas de qualidade de processo, a intenção é a de corrigir erros e falhas, padronizar atividades, alinhar o trabalho com a equipe de desenvolvimento, entre outros. Antes de se utilizar alguma metodologia de qualidade de processos, é necessário fazer o mapeamento deles.

Sommerville (2015) afirma que o mapeamento de processos é uma ferramenta gerencial que visa identificar a sequência na qual as atividades são executadas. Em termos práticos, o mapeamento pode gerar mais benefícios, como:

- **Compreensão dos processos:** permite entender todas as partes que os compõem.
- **Análise dos processos:** ao mapeá-los, é possível fazer uma reflexão sobre as atividades que os compõem.
- **Melhoria dos processos:** enxergando detalhadamente as atividades que os compõem, eles podem ser otimizados, ajustados ou substituídos, a fim de que sejam melhorados.
- **Padronização dos processos:** as atividades que possuem níveis de qualidade ajustados com as políticas da empresa dentro dos processos podem vir a se tornar padrão.
- **Documentação dos processos:** ao mapear as atividades que compõem os processos, é possível documentá-los, se assim a equipe

não tiver feito nas atividades que precedem a fase de desenvolvimento do projeto.

Percebeu que antes de qualquer coisa, devem-se mapear os processos para que se compreendam detalhadamente todas as atividades de desenvolvimento? Até aqui tudo certo. Mas com que nível de detalhamento o mapeamento deve ser feito? Segundo Sommerville (2015), existem três níveis de detalhamento dos processos, conforme pode ser observado:

- **Nível 1 – Descritivo:** por meio de uma descrição básica e abrangente dos processos, busca-se o alinhamento deles com as partes envolvidas no projeto de desenvolvimento.
- **Nível 2 – Analítico:** trata-se de uma fase técnica, na qual são detalhadas as atividades de desenvolvimentos, pontos de atenção e o tratamento de exceções.
- **Nível 3 – Executável:** é uma visão mais próxima aos dados, à aplicação em si. Aqui a intenção é detalhar as funcionalidades, os serviços e as saídas.

Com isso, após fazer o mapeamento dos processos, seja por meio de softwares, seja por meio de descrições (textos, tabelas ou quadros), pode-se utilizar ferramentas como: modelos, normas e metodologias, os quais compõem as formas que as equipes podem utilizar para garantir a qualidade de processos de desenvolvimento de software. Dentre essas ferramentas, abordaremos inicialmente os Modelos de Maturidade CMM e CMMI.

■ MODELOS DE MATURIDADE CMM E CMMI

Caro aluno, as discussões acerca de qualidade de processos têm uma abrangência significativa, isso porque os processos de desenvolvimento podem ser vistos sob diversos aspectos. Para isso, inicialmente, você

será conduzido às explanações acerca do grau de maturidade dos processos por meio de um modelo conhecido como **CMM** (*Capability Maturity Model*).

0

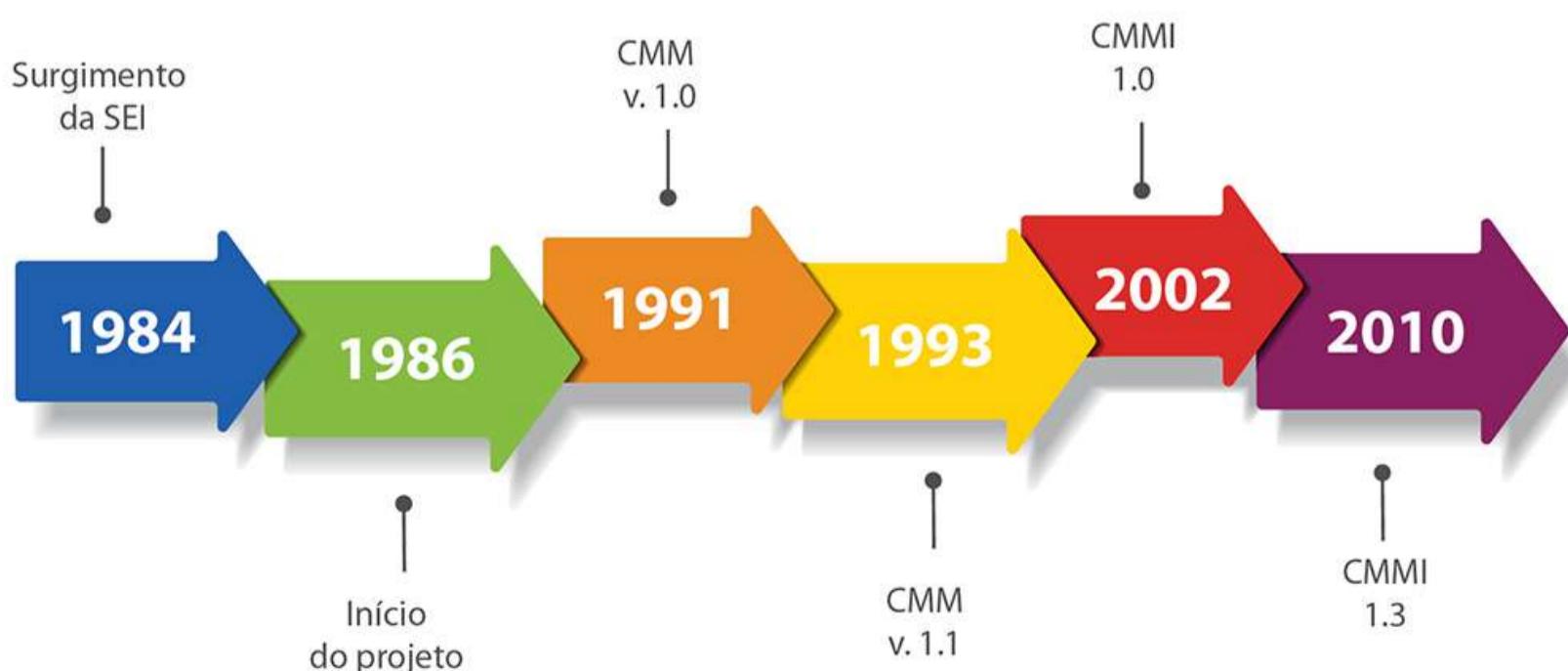
ASSIMILE

Os modelos de maturidade são ótimas ferramentas que permitem o conhecimento profundo dos processos, da equipe de desenvolvimento, das dificuldades, entre outros. Colocar tais metodologias em prática exige comprometimento e persistência, pois, em muitos casos, a resistência de colaboradores que nunca tiveram contato com as normas e diretrizes podem criar certo bloqueio em sua adoção.

Ver anotações

Segundo Couto (2007), o modelo CMM foi elaborado pelo Instituto de Engenharia de Software (conhecido por SEI – *Software Engineering Institute*) da Universidade Carnegie Mellon, nos Estados Unidos da América. Para compreensão da linha do tempo do CMM, observe a Figura 2.13.

Figura 2.13 | Linha do tempo CMM/CMMI



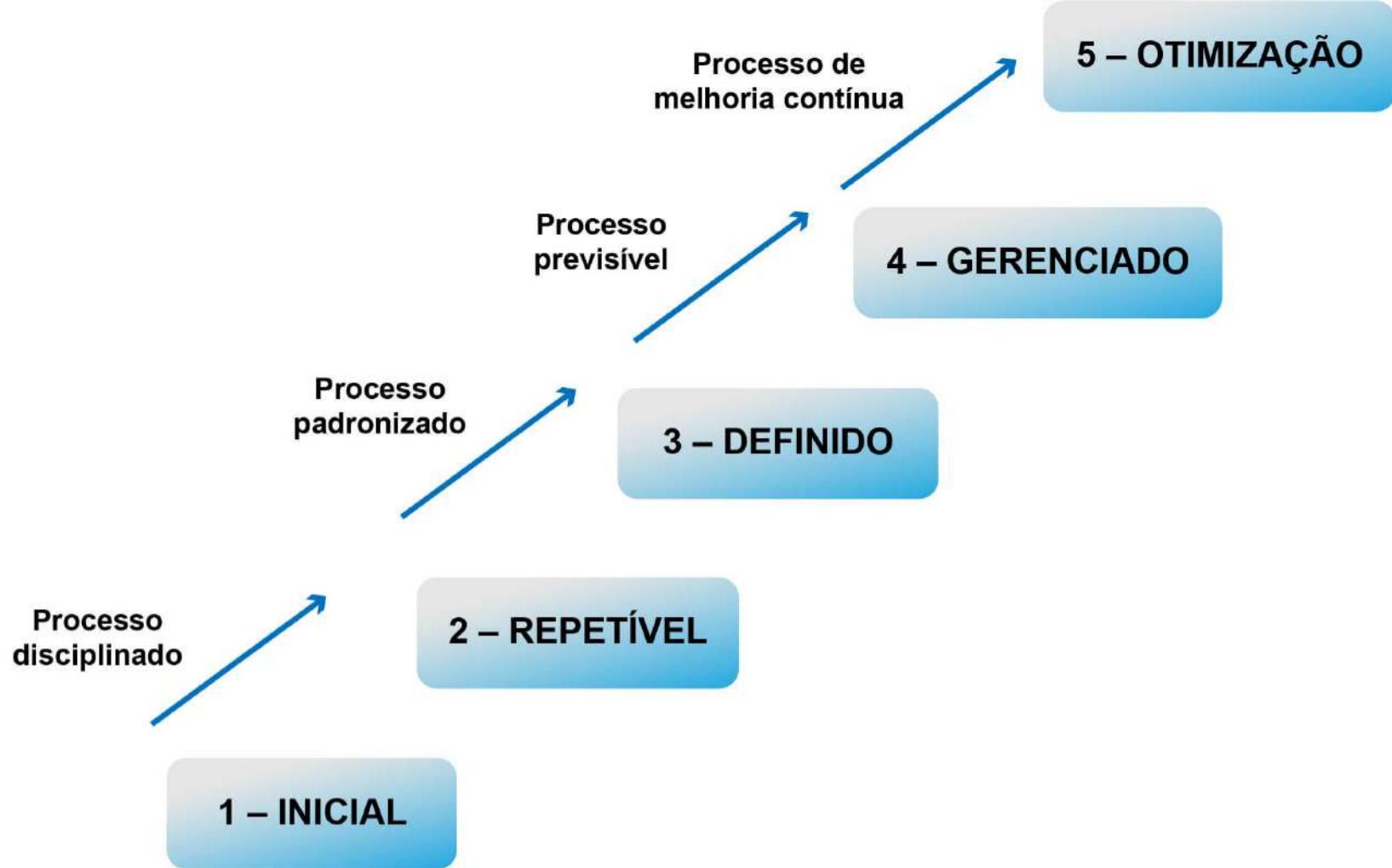
Fonte: adaptada de Couto (2007).

Antes de partirmos para a demonstração dos modelos de maturidade, vamos responder a uma indagação que você certamente está se fazendo: para que serve isso? Pois bem, imagine que um mesmo projeto é entregue para um grupo de programadores e para uma empresa de desenvolvimento de software. Ambos terão a mesma quantidade de profissionais, recursos computacionais e prazo. A partir disso, você pode imaginar que a entrega da empresa provavelmente será melhor. Por quê? Simplesmente porque os seus processos estão bem definidos, testados e possuem maturidade.

Percebeu como os processos estão muito presentes na Engenharia de Software? Os improvisos, nas atividades de desenvolvimento, não podem ocorrer, pois os resultados seriam desastrosos. É por isso que existem os modelos de maturidade de processos.

Segundo Couto (2007), os processos de melhoria se preocupam muito mais com a evolução dos processos que compõem as atividades de desenvolvimento do que com a agregação de novos processos ou inovações. O CMM utiliza, em sua estrutura, cinco níveis de maturidade, dentro de uma base hierárquica. Para melhor compreensão do modelo, observe a Figura 2.14.

Figura 2.14 | Modelo CMM



Fonte: adaptada de Couto (2007).

Com base no modelo apresentado, observe o detalhamento de cada um dos níveis:

- **Nível 1 (Inicial)**: não existe controle de processos, a equipe é guiada pelas atividades e entregas. O cliente faz a avaliação na entrega e, quando necessário, ocorrem os ajustes.
- **Nível 2 (Repetitivo)**: nessa fase os controles de processos básicos já são utilizados nos desenvolvimentos. Porém, o mais relevante é que os processos bem-sucedidos passam a ser replicados em outros projetos.
- **Nível 3 (Definido)**: após repetir boas práticas nos projetos, esses procedimentos são estabelecidos como padrão bem definido nas atividades de desenvolvimento. Com o tempo, passa a ser cultural entre os colaboradores.
- **Nível 4 (Gerenciado)**: uma vez que se esteja no nível 3, momento em que já existe uma definição dos processos, é possível utilizar ferramentas de medição de estatística para efetuar o gerenciamento e o controle dos processos.

- **Nível 5 (Otimização):** conforme o conhecimento acerca dos processos vai aumentando e, consequentemente, o nível de maturidade, é possível repensar alguns processos, permitindo a otimização destes.

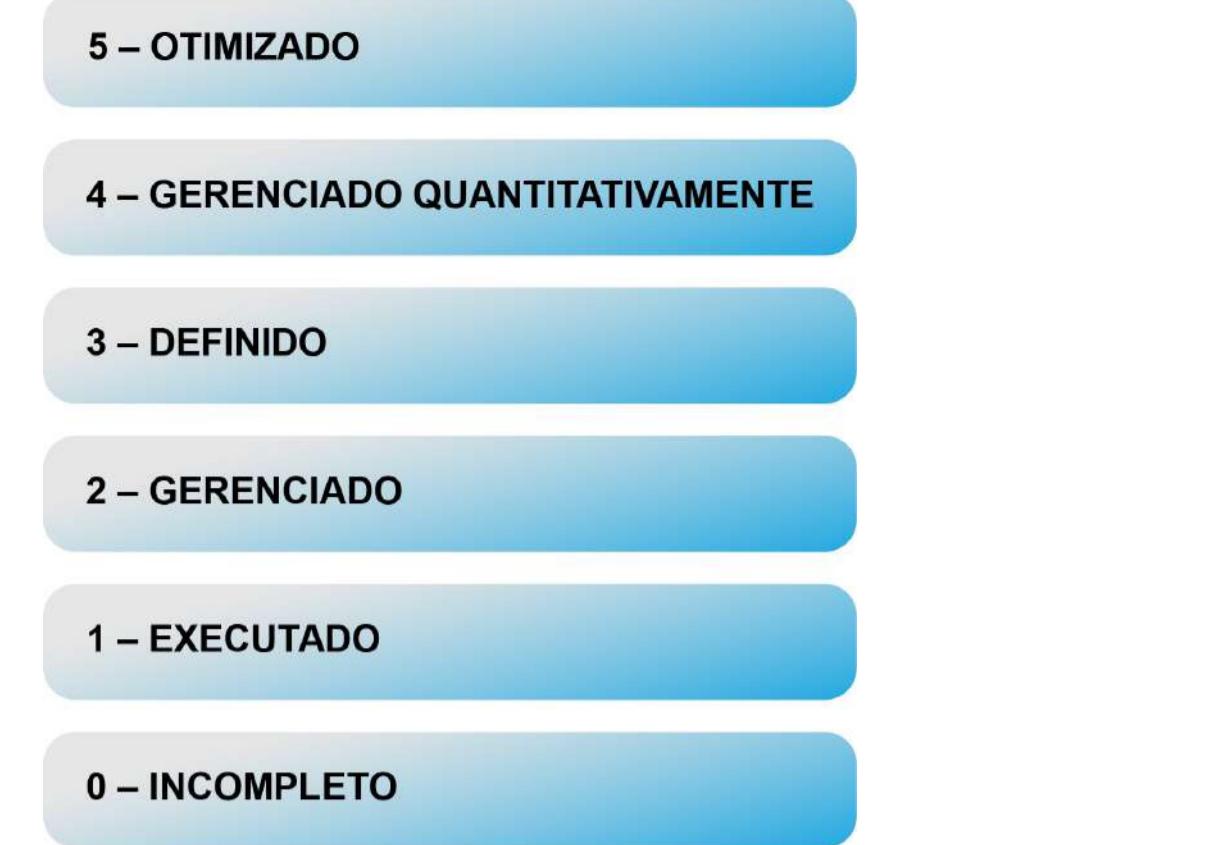
Ainda conforme define Couto (2007), os níveis de maturidade apresentam uma estrutura evolutiva, por meio da qual se busca atingir a qualidade esperada em determinado nível, para que, assim, seja possível passar ao próximo estágio e aumentar a capacidade dos processos dentro da organização.

Ainda dentro do tópico sobre modelos relacionados ao processo de desenvolvimento, porém agora com o olhar voltado para a capacidade dos processos, abordaremos o framework **CMMI** (*Capability Maturity Model Integration*). Também desenvolvido pela SEI, em 2012 foi lançada a sua última versão (1.3). O principal objetivo dessa ferramenta é analisar a capacidade da maturidade do processo de software.

Segundo Koscienski e Soares (2006), o CMMI, quando inserido na área de desenvolvimento de software, utiliza a capacidade e a maturidade para atingir metas previamente estipuladas, com nível de qualidade acordado entre as partes. Seu maior objetivo é a produção de software com o maior nível de qualidade e a menor taxa de erros.

A estrutura do CCMI é dívida em cinco níveis, os quais são utilizados para determinar a capacidade e a maturidade dos processos. Para melhor compreensão do modelo, observe a Figura 2.15.

Figura 2.15 | Modelo CMMI



Fonte: adaptada de Koscienski e Soares (2006).

Com base no modelo apresentado, observe o detalhamento de cada um dos níveis:

- **Nível 0 (Incompleto):** os processos não são executados em sua totalidade ou são parcialmente executados.
- **Nível 1 (Executado):** os processos conseguem satisfazer metas específicas, e a organização como um todo reconhece que existem processos definidos.
- **Nível 2 (Gerenciado):** nessa fase, além dos processos serem executados, existe também um monitoramento e um planejamento deles.
- **Nível 3 (Definido):** o processo serve como boa prática e se torna padrão dentro da organização.
- **Nível 4 (Gerenciado):** são utilizadas medições e técnicas de estatística para análise dos processos, o que gera, consequentemente, melhorias.
- **Nível 5 (Otimizado):** o foco é a melhoria contínua para a busca de melhores resultados.

Para que você compreenda o CMMI em uma situação profissional de desenvolvimento de software, observe o seguinte exemplo: o desenvolvimento do sistema web de determinada organização se dá em três partes: *front end*, *back end* e banco de dados. Ocorre que muitas vezes o tipo de dado tratado pela equipe de *back end* não está alinhado com o desenvolvedor de banco de dados. Ainda, ocorrem situações em que é necessária a validação dos campos tanto pelo *front end* (por meio do Bootstrap) quanto pelo desenvolvedor *back end*.

Percebeu como nesse caso, os processos de software não estão definidos? Claramente a equipe está no nível 1 do CMMI. Por meio de técnicas específicas, de planejamento e da utilização das diretrizes do CMMI, a equipe poderá visar ao próximo nível, otimizando e profissionalizando os seus processos de desenvolvimento de software.

| NORMAS ISO DE QUALIDADE DE PROCESSOS

Provavelmente você deve ter percebido, ao longo das discussões, que existem diversos modelos, normas e metodologias para todo o tipo de solução. Com a qualidade de processos não é diferente. Dito isso, você poderá compreender como a norma ISO 9001 pode auxiliar os desenvolvedores de software na busca da qualidade dos seus processos.

Segundo Valls (2003), a ISO 9001:2015 possui uma característica fundamental quanto à visão sistêmica que deve reger as atividades de desenvolvimento. Ou seja, os profissionais devem conhecer os processos, e esse conhecimento deve ser bem consistente entre os membros da organização como um todo.

Valls (2003) defende ainda que a ISO 9001 é um sistema de qualidade que visa à garantia de otimização dos processos. Essa norma também é conhecida nos meios profissionais como SQA (Sistema de Gestão da

Qualidade). Para os gestores de projetos de desenvolvimento de software é um poderoso instrumento de correção e de melhoria de processos.

Conforme definido por Koscienski e Soares (2006), a ISO 9001 é estruturada por meio de oito princípios da qualidade, conforme pode ser observado a seguir:

- **Foco no cliente:** o pilar principal de qualquer organização deve ser o cliente, de modo que a norma orienta a busca constante das necessidades e expectativas dele. Exemplo: o cliente deseja uma aplicação mobile, porém a empresa apresenta uma “opção” de aplicação web, que é acessada por meio do navegador. Embora tenha sido apresentada uma solução (que poderia até resolver o problema do cliente), o que foi solicitado foi para uma aplicação mobile.
- **Liderança:** na norma existem indicativos para uma liderança que busque, com disciplina, empenho, engajamento e dedicação, os melhores resultados para a organização. Exemplo: a equipe está envolvida em um projeto cujo cliente solicita novas funcionalidades e modificações (sendo que isso estava previsto no contrato). Retrabalho pode gerar desconforto na equipe. Por meio da norma, o gestor pode reverter esse quadro.
- **Envolvimento das pessoas:** os desenvolvedores devem ter ciência do seu papel no projeto. Além disso, devem ocorrer treinamentos, workshops e outras atividades que otimizem o seu desempenho. Exemplo: um projeto necessita consumir uma API cujo funcionamento é pouco conhecido. A organização pode promover cursos Hands on de forma que os desenvolvedores possam atender ao projeto sem que ocorram improvisos.
- **Abordagem do processo:** formalizar os processos que devem ser utilizados no desenvolvimento. Gerenciar a sua correta utilização, promovendo ajustes quando necessário. Exemplo: determinado

desenvolvedor acha que a construção de uma função sairá melhor fazendo do jeito dele do que utilizando o processo estabelecido na organização. Nesse caso, são necessárias algumas intervenções para ajuste de conduta.

- **Abordagem sistêmica para a gestão:** os processos devem ser visualizados como partes que compõem um sistema. Dessa forma, as pessoas envolvidas nos processos conseguem entender melhor o seu papel no projeto. Exemplo: imagine que exista uma equipe desenvolvendo modos de pagamento para um e-commerce. Os desenvolvedores necessitam conhecer os processos que antecedem o pagamento (escolha do produto, cesta de compras, cadastro do cliente, etc) e, ainda, os que são posteriores (retorno de confirmação de pagamento, acompanhamento de envio, etc).
- **Melhoria contínua:** nesse princípio, os colaboradores compreendem o detalhamento dos processos e promovem melhorias. Exemplo: determinada API, que constrói gráficos com consulta a banco de dados relacional, apresenta um atraso. Por meio de testes, a equipe descobriu que, ao se utilizar banco de dados não relacional, existe uma diminuição no tempo de consulta.
- **Abordagem para tomada de decisão:** os indicadores de qualidade devem permitir uma análise e oportunizar melhorias. Exemplo: ao se utilizar um software para contagem de linhas de códigos produzidas, percebeu-se que um intervalo de cinco minutos a cada meia hora gera um aumento de produtividade.
- **Benefícios:** deve haver uma relação de parceria entre as partes, de forma que possam gerar benefícios nos processos de desenvolvimento. Exemplo: uma forma de estreitar laços e conhecer os processos do cliente é alocar, por um período, na empresa, colaboradores responsáveis por mapeamento de processos e levantamento de requisitos.

Ainda de acordo com Valls (2003), ao se implantar o modelo ISO 9001:2015, deve-se: estabelecer, manter e buscar melhorias para a gestão da qualidade com ênfase nos processos e nas interações entre os microprocessos. Dessa forma, fica evidente o objetivo de se conhecer a fundo cada parte dos processos e a interação entre eles, para que não ocorra a dissociação entre os processos.

Você percebeu como a compreensão da ligação entre as partes facilita o entendimento dos processos? Além da compreensão desse aspecto, Valls (2003) afirma que, para a melhoria dos processos, as organizações devem compreender claramente os pontos a seguir:

- **Entradas e saídas:** todo software possui entradas e saídas. Estas devem estar bem claras desde a fase de levantamento dos requisitos, pois isso possibilita o aumento da maturidade desse ponto.
- **Sequência dos processos:** os processos, quando muito bem claros e estabelecidos, permitem à equipe compreender como será a sequência de trabalho. Esta deve apresentar uma lógica bem estruturada.
- **Interação dos processos:** o conhecimento dos processos permite a compreensão da forma e do momento em que ocorrerão as interações entre eles, permitindo, assim, o estabelecimento de pontos de atenção dentro do projeto.
- **Os recursos disponíveis:** não se trata apenas de recursos computacionais. Os recursos mais escassos normalmente estão ligados a prazo e a mão de obra especializada.
- **As responsabilidades:** cada colaborador dentro da equipe deve ter em mente sua atribuição dentro do projeto, além de conhecer os pares de interação e reconhecer as lideranças.
- **Os riscos:** conhecer os processos mais a fundo permite aos gestores de projetos de desenvolvimento a identificação dos riscos e dos

pontos de atenção, o que reflete em ações preventivas para o cumprimento da qualidade e dos prazos.

O processo de implantação da ISO 9001 é relativamente simples e pode ser feito nas empresas independentemente do seu porte. Porém, obter o reconhecimento da aplicação 100% correta por meio da certificação ISO 9001 exige que a empresa já possua um bom nível de maturidade em seus processos de desenvolvimento de software, além de um conhecimento profundo das partes que compõem as suas normas e diretrizes.

Ver anotações

MELHORIA DE PROCESSOS INDIVIDUAIS E DE EQUIPE (PSP)

Em uma coisa a maioria de nós concorda: os softwares são desenvolvidos por pessoas. Indivíduos que, em algum momento da vida, dedicaram-se a aprender a desenvolver sistemas por meio de alguma tecnologia e hoje abstraem informações das pessoas e transformam essas ideias em códigos. É por esse motivo, justamente, que surge a ferramenta PSP, a qual visa às pessoas que desenvolvem os sistemas.

Segundo Koscienski e Soares (2006), a sigla **PSP** significa *Personal Software Process*, cujo objetivo é promover o desenvolvimento de software com enfoque na habilidade individual dos colaboradores. Segundo o PSP, o conhecimento, a avaliação e a melhoria contínua no processo individual de desenvolvimento de software deve observar os erros e as falhas cometidas para que possam ser corrigidos e aprendidos pelo desenvolvedor.

Para exemplificar uma das formas de se operacionalizar o PSP em um projeto, pode-se utilizar a contagem de erros em código de desenvolvimento. Uma das maneiras clássicas utilizadas para esse fim é conhecida por Kloc, que tem como objetivo contar a quantidade de

erros a cada mil linhas de código. Ainda segundo Koscienski e Soares (2006), o PSP é composto por quatro níveis de competência conforme pode ser observado a seguir:

- **PSP 0 – Processo atual:** aqui deve ser estabelecida a base da competência, que inclui a determinação dos métodos de medição, o que medir e a que momento (desenvolvimento, compilação e teste), permitindo, assim, o desenvolvimento de relatórios para análise de falhas. Nessa fase ainda deve-se desenvolver o PIP (*Process Improvement Proposal*), o qual, por meio de um formulário, deve registrar os problemas dos processos e sugestões de melhorias.
- **PSP 1 – Estimativa de tamanho:** nessa fase existem apenas dois objetivos: o planejamento do tempo e das tarefas. Trata-se de uma fase de planejamento pessoal, na qual se deve ter uma estimativa de quantas tarefas estão atribuídas ao colaborador e o tempo de desenvolvimento delas. É claro que essas medidas variam conforme o nível de competências e habilidades do desenvolvedor.
- **PSP 2 – Revisão de código:** o enfoque dessa fase é o processo de administração da qualidade pessoal. Nela a visão técnica da tecnologia, por meio da qual está se desenvolvendo, deve observar a aplicação dos processos estabelecidos e os resultados positivos e negativos.
- **PSP 3 – Desenvolvimento cíclico:** trata-se de um processo pessoal cíclico, no qual a correta aplicação dos processos, que atendem as demandas e a qualidade, seja processada em cascata, a fim de se obterem os melhores resultados.

Koscienski e Soares (2006) afirmam que as ferramentas de aferição, em conjunto com os tratamentos, fornecem curvas estatísticas que possibilitam uma análise mais profunda do desenvolvedor e ainda projeção de sua curva de evolução profissional. Porém, essa mesma

técnica pode diagnosticar membros de equipe que, embora apliquem ações corretivas, insistem em não alinhar os processos de desenvolvimento.

REFLITA

As normas, referências, guias e metodologias fornecem um referencial para se atingir um objetivo e normalmente são documentos completos que permitem muitos olhares para um mesmo problema, fazendo com se tenha uma base técnica muito interessante.

De fato, as atividades são desenvolvidas por pessoas, mas será que apenas uma metodologia pode ser capaz de promover mudanças?

MELHORIA DE PROCESSOS DE SOFTWARE BRASILEIRO (MPS.BR)

No Brasil, nós possuímos uma ferramenta que possibilita a melhoria dos processos de softwares brasileiros. Porém, isso não significa que as suas normas sejam melhores ou piores que outras, pois estas seguem padrões internacionais de qualidade de processos. Aqui no País, a entidade que faz a gestão do **MPS.BR** é a **Softex**, responsável por apoiar a cultura da qualidade de software e por contribuir com a melhoria contínua dos processos/produto de software.

EXEMPLIFICANDO

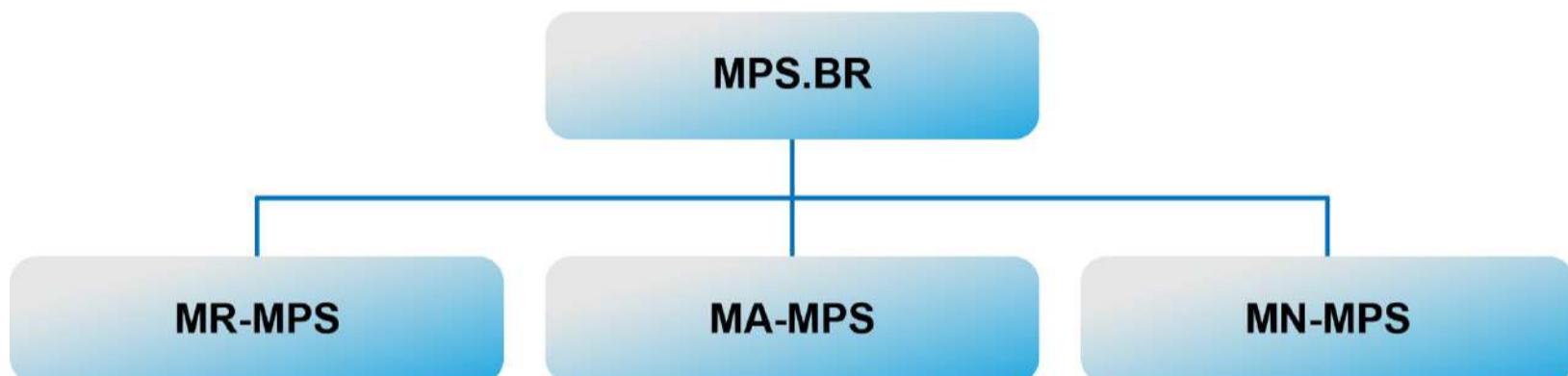
Uma dúvida muito comum no MPS.BR é referente aos processos de implementação. A Softex, gestora do MPS.BR, disponibiliza um tutorial com os passos para uma empresa ser avaliada quanto ao nível de maturidade na utilização do modelo. Os passos necessários podem ser observados a seguir:

1. Habilitar um representante da empresa na equipe de avaliação.

2. Implementação do modelo MPS.BR.
3. Contratação da instituição avaliadora.
4. Avaliação oficial.
5. Publicação do resultado oficial (SOFTEX, [s.d.]).

Segundo Rocha (2005), a construção das técnicas constituintes ao MPS.BR é composta pelas NBR ISO/IEC, conforme pode ser observado no esquema representado na Figura 2.16.

Figura 2.16 | Esquema MPS.BR



Fonte: adaptada de Rocha (2005).

Para a compreensão das siglas que compõem o MPS.BR, acompanhe os tópicos a seguir:

- **Modelo de Referência (MR):** contém informações a forma como a organização deve conduzir os seus processos para atingir os resultados. Ainda é possível, nesse mesmo documento, determinar o nível de maturidade e da capacidade dos processos descritos na NBR ISO/IEC 12207.
- **Modelo de Avaliação (MA):** trata-se do processo no qual são determinados os parâmetros e requisitos para se aferir a qualidade do desenvolvimento. É baseado na norma ISO/IEC 15504.
- **Modelo de Negócio (MN):** determina o nível de maturidade dos processos, que podem ser: (A) Otimizado, (B) Gerenciado, (C) Definido, (D) Largamente definido, (E) Parcialmente definido, (F) Gerenciado e (G) Parcialmente gerenciado.

Vale ressaltar que cada modelo apresenta o seu formulário específico:

MR (Guia Geral e Guia de Aquisição), **MA** (Guia de Avaliação) e **MN**

(Documentos do projeto). Segundo Rocha (2005), a construção das técnicas constituintes ao MPS.BR é composta pelas NBR ISO/IEC:

- NBR ISO/IEC 12207 para processo de ciclo de vida de software.
- ISO/IEC 15504 para avaliação de processo.
- ISO/IEC 15504-5 para modelo de avaliação de processo de software.

Na prática, como o MBS.BR deve ser iniciado no nível G, para o qual o manual determina 28 GPR, que são os objetivos que a equipe deve alcançar para atingir de maturidade. Observe a seguir as três primeiras GPR do nível G:

“

GPR 1. O escopo do trabalho para o projeto é definido;
GPR 2. As tarefas e os produtos de trabalho do projeto são dimensionados utilizando métodos apropriados;
GPR 3. O modelo e as fases do ciclo de vida do projeto são definidos;
— (MPS.BR, 2012, p.)

Os GPRs devem ser documentados, gerenciados e acompanhados. O modelo de documento é sugerido pelo manual, porém ele mesmo deixa flexível para que as empresas adicionem ou retirem quais campos desejar.

Caro aluno, você percebeu ao longo do texto como os métodos, normas e modelos de qualidade de processo são importantes para a área de desenvolvimento de software? Por esse motivo, é importante que esses pontos abordados sejam compreendidos e aplicados em projetos de desenvolvimento. Fica evidente que, ao utilizar tais técnicas, os resultados gerados, conforme o nível de maturidade aumenta, e as melhorias tendem a ser constantes na maioria dos processos.

SAIBA MAIS

Normas como ISO 9000, 9001, etc. possuem certificações.

São provas feitas em unidades certificadoras que

comprovam, por meio de testes, que o profissional possui

competências e habilidades para utilizar determinada tecnologia.

PESQUISE MAIS

Em diversas normas, métodos e metodologias, são utilizados níveis para classificar a maturidade de uma equipe/organização. Entre esses níveis, normalmente existe um em que o nível de maturidade é gerenciado e em que são utilizados medições e tratamentos estatísticos para uma análise dos processos.

A UNIVESP disponibiliza, em seu canal do YouTube, uma aula que trata exatamente desse tema. Não deixe de conferir!

CONTROLE Estatístico de Processo – Aula 01 – Introdução ao Controle de Qualidade. São Paulo: Univesp, 2017. 1 vídeo (21 min). Publicado pelo canal UNIVESP.

FAÇA VALER A PENA

Questão 1

Uma empresa estava envolvida no processo de desenvolvimento de um software para monitoramento de redes de computadores. O objetivo desse software era efetuar a medição de jitter, latência e vazão dos pacotes.

O gerente de projetos, por entender quão complexo é esse desenvolvimento, pensa ser necessário efetuar algumas medições, que passarão por tratamentos estatísticos, a fim de se prever erros, falhas e prazos.

Com base no modelo de classificação CMM, assinale a alternativa que descreva corretamente o nível da empresa de desenvolvimento de software quanto ao seu nível de maturidade.

a. Nível 1.

b. Nível 2.

c. Nível 3.

d. Nível 4.

e. Nível 5.

Questão 2

A empresa júnior de uma faculdade faz desenvolvimentos diversos, porém, nos últimos tempos, tem surgido uma grande demanda por aplicações web. Desde a sua concepção, os veteranos adotaram o PSP para acompanhar a evolução pessoal dos colaboradores.

Assim como descreve Koscienski e Soares (2006), existem nele quatro níveis: PSP 0, PSP 1, PSP 2 e PSP 3, os quais são aplicados diariamente por meio de formulários e outras ferramentas de análise. O ambiente de trabalho é muito interessante, porque, por exemplo, às 15h os scripts são enviados ao gestor de projetos e os desenvolvedores podem utilizar a sala de descanso/jogos/leitura.

Às 15h ocorre uma das fases do PSP. Com base no exposto, assinale a alternativa com a fase PSP correta.

a. PSP 0.

b. PSP 1.

c. PSP 2.

d. PSP 3.

e. PSP 4.

Questão 3

Uma equipe de desenvolvedores se reuniu para atender uma demanda de aplicativos para iOS. Os projetos têm chegado diariamente e a equipe percebeu que está muito próxima aos limites de projetos em desenvolvimento. Porém, um dos líderes dos times de desenvolvimento percebeu que cada equipe utiliza os processos adotados desde a

concepção da organização e isso estava consumindo muito tempo, principalmente em projetos em que os trabalhos eram segmentados pelas equipes.

Caso seja utilizado o MN-MPS.BR, como a empresa poderia ser nivelada quanto a sua maturidade?

a. Parcialmente definido.

b. Parcialmente gerenciado.

c. Largamente definido.

d. Gerenciado.

e. Otimizado.

REFERÊNCIAS

CONTROLE Estatístico de Processo – Aula 01 – Introdução ao Controle de Qualidade. São Paulo: Univesp, 2017. 1 vídeo (21 min). Publicado pelo canal UNIVESP. Disponível em: <https://bit.ly/2Xg7WkH>. Acesso em: 19 out. 2020.

COUTO, A. B. **CMMI**: integração dos modelos de capacitação e maturidade de sistemas. Rio de Janeiro: Ciência Moderna, 2007.

KOSCIANSKI, A.; SOARES, M. dos S. **Qualidade de software**: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. São Paulo: Novatec, 2006.

MPS.BR – Melhoria de Processo do Software Brasileiro. Guia Geral MPS de Software. Brasília: Softex, 2012. Disponível em: <https://bit.ly/3oluRa5>. Acesso em: 15 nov. 2020.

SOFTEX. MPS BR. **Softex**, Brasília, [s.d.]. Disponível em: <https://softex.br/mpsbr/>. Acesso em: 16 out. 2020.

SOMMERVILLE, Ian. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2015.

VALLS, V. M. A documentação na ISO 9001:2000. **Banas qualidade**, São Paulo, v. 12, n. 133, p. 100-105, jun. 2003.

FOCO NO MERCADO DE TRABALHO

QUALIDADE DE PROCESSO

Sergio Eduardo Nunes

Ver anotações

CLASSIFICAÇÃO DE MATURIDADE DOS PROCESSOS A NÍVEIS ACEITÁVEIS

O CMM utiliza, em sua estrutura, cinco níveis de maturidade, dentro de uma base hierárquica, visando posicionar a empresa em um nível, a fim de que, após isso, a organização possa evoluir nos níveis dos seus processos.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

SEM MEDO DE ERRAR

Os vínculos com a prefeitura para emissão de relatórios sobre projetos voltados à tecnologia da informação já renderam muitos contratos a você. Os relatórios são sempre muito técnicos e exigem conhecimentos específicos para que sejam elaborados e para que atendam às demandas da prefeitura com a qualidade esperada por tantos gestores que passaram pelo cargo do executivo da cidade. Tendo isso em vista, foi elaborado o seguinte relatório:

Este relatório tem o intuito de esclarecer os níveis CMM e posicionar a empresa que venceu a licitação quanto ao seu nível. Existem alguns detalhes técnicos importantes, os quais o senhor deve compreender para que fique claro como a empresa, que está no processo de contratação, foi posicionada. A metodologia adotada para elaborar o relatório com o nível de maturidade é o CMM.

A estrutura do CMM, visa posicionar a empresa em um nível, a fim de que, após isso, a organização possa evoluir nos níveis dos seus processos. Para tal, observe os detalhes dos níveis descritos a seguir:

- **Nível 1 (Inicial):** não existe controle de processos.
- **Nível 2 (Repetitivo):** os processos básicos já são utilizados nos desenvolvimentos.
- **Nível 3 (Definido):** os bons processos são estabelecidos como padrão.
- **Nível 4 (Gerenciado):** são utilizadas ferramentas de medição para melhorar os processos.
- **Nível 5 (Otimização):** busca-se a otimização dos processos.

Observe que essa empresa apresenta algumas características bem definidas, conforme pode ser observado a seguir:

- No setor administrativo a empresa tem uma estrutura familiar.
- Os desenvolvedores são contratados pelo regime CLT.

- O portfólio da empresa é interessante, com muitos contratos nas prefeituras.
- A metodologia segue um esquema por meio do qual é originada uma demanda pelo gerente de projetos.
- O processo de integração das partes não é definido, sendo passível de falhas, erros, complementos e ajustes.

Esses pontos de atenção fazem com que a empresa se encaixe exatamente no Nível 1 do CMM, ou seja, no qual não existe definição de processos. É válido lembrar que não cabe a esse relatório julgar a capacidade da organização, mas apenas posicionar a empresa quanto ao nível de maturidade no qual se encontra atualmente.

AVANÇANDO NA PRÁTICA

FORMULÁRIO PSP

Uma empresa de desenvolvimento de software, para prover educação, recebeu uma demanda na qual serão utilizadas tecnologias interativas, como realidade virtual ou aumentada. O projeto em si é muito interessante e inovador, porém o maior entrave está na qualificação técnica da equipe, sobre a qual, na verdade, não se tem nada documentado. Ou seja, de fato, a empresa não conhece a própria equipe. Isso é muito negativo, pois impacta diretamente no setor comercial, que não tem um direcionamento quanto aos contratos que podem ser fechados, segundo o nível de conhecimento profissional de cada um dos colaboradores.

Pensando nessa situação, o gerente de projetos enviou para você um link no qual havia informações sobre as características de uma técnica conhecida por PSP, sobre a qual você já havia estudado durante a graduação. Junto com o link, o gerente de projetos fez a seguinte solicitação: “por gentileza, com base nas premissas do PSP, desenvolva um formulário para que possamos classificar os desenvolvedores quanto ao seu nível”.

Com isso, você deve aprofundar os conhecimentos acerca do PSP e elaborar um formulário, cujo objetivo é conhecer o nível de conhecimento e de maturidade dos nossos processos. Isso permitirá ao gerente de projetos tomar uma decisão acerca do fechamento de contrato do projeto educacional com realidade virtual e aumentada.

o

Ver anotações

RESOLUÇÃO



A ideia do relatório não é apenas auxiliar nesse novo contrato, mas também utilizar o PSP como base para realmente conhecer os membros da equipe de desenvolvimento. Para isso, a proposta do formulário pode ser observada a seguir:

Quadro 2.2 | Proposta de formulário PSP

Característica	Insatisfatório	Regular	Bom
Atendimento do escopo			
Qualidade e inovação na tecnologia utilizada			
Compatibilidade das entregas com o demais			
Cumprimento dos prazos			
Cumprimento dos ajustes e revisões			
Participação nas reuniões			
Disponibilidade para a equipe			
Proatividade			

Fonte: elaborado pelo autor.

Esse modelo é flexível e permite a inserção de novas informações ou ainda a retirada de algumas que não satisfaçam a necessidade do projeto.

NÃO PODE FALTAR

CONCEITOS DE TESTES DE SOFTWARE

Roque Maitino Neto

Ver anotações

O QUE SÃO TESTES DE SOFTWARE?

Os testes são processos que representam uma sequência de ações executadas com o objetivo de encontrar problemas no software, o que aumenta a percepção da qualidade geral dele e garante que o usuário final tenha um produto que atenda às suas necessidades (PINHEIRO, 2015).



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Prezada aluna, caro aluno, iniciamos aqui mais uma unidade do nosso conteúdo de Engenharia de Software e aproveitamos a oportunidade para desejar, desde já, que você tenha um excelente aproveitamento dos temas que nela serão abordados.

Não é de hoje que a elevada qualidade de um produto, além de alguns outros fatores de natureza mais subjetiva, é o que conta para fazer dele um produto de sucesso em seu segmento. Um carro, um aparelho de televisão ou um telefone celular, por exemplo, tendem a seguir como produtos comercialmente viáveis se atenderem a certos requisitos de qualidade identificados com os seus propósitos e com o nicho a que se destinam. Naturalmente essa premissa também continuará sendo verdadeira se usarmos um software como exemplo e, nesse contexto específico, as providências para se garantir bons níveis de qualidade vêm sendo aprimoradas, especialmente aquela que é o tema central desta unidade: o teste de software.

Durante os três encontros desta unidade, teremos a oportunidade de conhecer detalhes sobre o processo de teste. Na primeira seção, trataremos do processo comum através do qual os testes são planejados, executados e têm seus resultados analisados. Nesse conjunto de informações, serão inseridos, também, conceitos de casos de testes e de depuração, além da abordagem de validação e de verificação, importantes providências de qualidade relacionadas aos testes. Já na segunda seção, os testes serão classificados e discutidos segundo uma tipificação própria, que leva em conta – entre outros fatores – o ambiente em que o software é executado e o paradigma de programação usado em sua construção. Esses tipos incluirão testes de funcionalidade, testes estruturais, voltados a aplicações móveis, e a aplicações orientadas a objetos.

Por fim, na última seção, nosso foco estará voltado aos testes automatizados e às ferramentas que viabilizam essa prática. Além disso, trataremos de aspectos de gerenciamento de testes e do

desenvolvimento orientado a testes, que constitui um meio bastante ágil de se testar funcionalidades do sistema. Esperamos que seu aproveitamento nesta unidade seja pleno e que o conteúdo nela desenvolvido possa ser aproveitado para aumentar sua capacidade de aplicar testes com excelência e para torná-lo apto a compreender a real importância dessa prática em um projeto de software.

Bom estudo!

PRATICAR PARA APRENDER

Consideremos um sistema minuciosamente planejado, construído com base na metodologia mais moderna que existe e codificado por desenvolvedores experientes e de alto nível. Imagine agora que, diante desse cenário de excelência técnica – e da expectativa de já contar com um produto de alta qualidade –, o líder do projeto tenha deixado a realização dos testes a critério apenas dos desenvolvedores, sem se importar em definir um plano para sua elaboração. Bem, o que viria a seguir seria, provavelmente, um retumbante insucesso no projeto e a possibilidade de a equipe arcar com grande quantidade de retrabalho.

Para que uma situação como essa não faça parte da sua realidade profissional, esta seção se incumbirá de abordar elementos de planos de testes e, nesse contexto, discutirá o papel dos casos de testes e da depuração de código como ferramentas essenciais em seu cumprimento. Nossa jornada, no entanto, se inicia com a conceituação de validação, de verificação e de alguns aspectos elementares de teste de software, na condição de elemento constituinte do conjunto de providências a serem tomadas para a garantia da qualidade de um produto de software.

Uma empresa de desenvolvimento de software em que você trabalha vem aplicando testes com sucesso em seus produtos e tem obtido satisfatório nível de qualidade em suas entregas. No entanto, os gestores da empresa detectaram aumento na quantidade de manutenções corretivas em seus programas, motivado sobretudo por

problemas de baixa complexidade no código e, teoricamente, de resolução simples. Os gestores imaginam que, atentos aos defeitos potencialmente mais danosos, os testadores não têm dado a devida relevância aos problemas simples.

Na busca por soluções para esses casos, os gestores colocaram a depuração do código como elemento fundamental em qualquer procedimento de teste e forneceram diretrizes para ampliar a aplicação desse recurso nos casos em que ele já era utilizado. Em uma de suas primeiras experiências na utilização sistemática da depuração, a equipe da qual você faz parte detectou a ocorrência de alguns defeitos no código que lhe foi confiado. Embora esses defeitos tenham se manifestado através da ferramenta, a equipe não sabia, de imediato, como localizá-las no código. A solução do caso passava, então, pela aplicação do processo de depuração, conforme designado pelos gestores. Em contato com o desenvolvedor, você tomou conhecimento que ele suspeitava de um certo trecho do código em que o problema poderia estar localizado. Expresso em pseudocódigo, esse trecho era o seguinte:

Utilizando,

```
início
    inteiro i, v, f=0;
    faça {
        escreva ("Informe um valor entre 1 e 10");
        leia (valor);
        enquanto (valor < 1 e valor > 10)

        para (i=1, i<=valor, i++) f=f*i;
        imprima f.
    fim.
```

Para solucionar este caso, as atribuições dadas a você foram as seguintes:

- Transformar o pseudocódigo em um código-fonte, na linguagem de programação à sua escolha.
- Como as linguagens mais recentes colocam à disposição do desenvolvedor um ambiente integrado de desenvolvimento, você terá acesso ao modo de depuração oferecido por esse ambiente. Encontre-o e familiarize-se com ele.
- Submeta o código-fonte ao processo de depuração, incluindo as seguintes ações:
 - a. Execução do código passo a passo.
 - b. Inspeção de, ao menos, uma variável do programa.
- Ao final, descreva o(s) defeito(s) encontrado(s) no código por meio do uso do recurso da depuração.

Bom trabalho!

DICA

Assim, dominar os conceitos relacionados à validação e à verificação de um software, conhecer as etapas de um processo de teste e saber aplicar os critérios para determinar os melhores casos de teste possíveis para a ocasião serão fundamentais nesta atividade. E lembre-se: quanto mais recursos forem investidos nos testes de seu software, menor o potencial retrabalho a ser executado e maiores as chances de que seu cliente o recomende para trabalhos futuros. Boa sorte e siga conosco!

CONCEITO-CHAVE

Já vai longe o tempo em que um sistema computacional tinha sua execução restrita apenas a um computador de mesa e servia a propósitos típicos do meio corporativo, como controle de vendas e de estoques. Atualmente, há uma grande variedade de equipamentos que dependem de sistemas para executarem suas funções de modo muito

mais flexível e confiável do que seria sem uma aplicação computacional. Exemplos não faltam e nossos carros e televisores são apenas dois deles. Há programas sendo executados em todos os lugares e controlando dispositivos indispensáveis em nosso cotidiano. Mas o que aconteceria se esses programas falhassem?

Para que essa pergunta não precise ser respondida da pior forma, a Engenharia de Software desenvolveu mecanismos para que os produtos de software – tantos os comuns quanto os que dão “inteligência” aos nossos equipamentos – criados pelos desenvolvedores passassem por processos que atestassem sua aptidão para executar suas funções de forma adequada e com elevados níveis de qualidade. Esses processos incluem a verificação, a validação e os testes de software e, se por meio de uma análise menos atenta eles podem parecer idênticos, uma conceituação apropriada servirá para esclarecer seus conceitos e diferenciá-los.

GERENCIAMENTO DA QUALIDADE DO SOFTWARE

No entanto, antes de individualizarmos esses termos, vale a pena posicioná-los em um contexto mais amplo, relacionado à qualidade de um produto, e conhecido por Gerenciamento da Qualidade do Software (ou *Software Quality Management*). Nesse contexto, a verificação e a validação são colocadas como ações intimamente relacionadas, destinadas à averiguação da conformidade do produto com as necessidades do cliente e comumente referenciadas em conjunto, sob a sigla V&V.

De acordo com uma importante publicação da área de Engenharia de Software (IEEE, 2014), o objetivo da V&V é ajudar a organização que desenvolve software a incorporar qualidade ao sistema durante o ciclo de vida, por meio de avaliações objetivas de produtos e de processos. Tais avaliações demonstram se os requisitos são corretos, completos,

precisos, consistentes e testáveis. Os processos de V&V determinam se os produtos desenvolvidos em certa atividade estão em conformidade com os requisitos dela e se o produto satisfaz seu uso pretendido.

As atividades de V&V, portanto, não são aplicadas unicamente a um programa ou função, mas a qualquer artefato que seja criado como resultado de determinada etapa do ciclo de vida de um produto. Os requisitos, o projeto e a implementação do produto são artefatos que podem (e devem) passar por verificações e validações.

Nesse sentido, a IEEE (2014) estabelece que a **verificação** é uma tentativa de garantir a correta construção do produto, com vistas a atender as especificações impostas aos produtos de saída em atividades anteriores. Já a **validação** é uma tentativa de garantir que o produto certo seja construído, ou seja, que ele atenda a sua finalidade específica.

Schach (2009) relaciona os conceitos de verificação e de validação a momentos específicos do ciclo de vida de um produto de software. Nesses termos a verificação se refere ao processo de determinar se um fluxo de trabalho foi executado corretamente ou não, e ela ocorre ao final de cada fluxo de trabalho. Já a validação é o processo intensivo de avaliação que ocorre imediatamente antes de o produto ser entregue ao cliente; seu propósito é o determinar se o produto como um todo satisfaz ou não às suas especificações.

Embora sejam expressões parecidas e estejam inseridas em um contexto único, verificação e validação não são a mesma coisa. Santos e Oliveira (2017) resumem assim os termos: verificação refere-se à garantia das especificações do software em uma determinada fase do desenvolvimento, enquanto a validação se refere à garantia do produto de software como um todo. A validação é uma fase mais geral, na qual o produto criado é confrontado com as expectativas do cliente.

Observadas sob uma perspectiva teórica, expressões como “incorporar qualidade ao sistema” e “verificar se uma atividade está em conformidade com os requisitos” podem transmitir uma falsa sensação

de simplicidade procedural. Entretanto, a aplicação de procedimentos de verificação e validação requerem planejamento cuidadoso e precisão na execução. O objetivo do planejamento de V&V é garantir que cada recurso, função e responsabilidade sejam claramente atribuídos.

Nessa fase de planejamento, devem ser especificados os recursos, suas funções e atividades, bem como as técnicas e ferramentas a serem usadas. A compreensão dos diferentes propósitos de cada atividade de V&V ajuda no planejamento cuidadoso das técnicas e dos recursos necessários para cumprir seus propósitos. O planejamento também deve abordar a gestão, a comunicação, as políticas e os procedimentos das atividades de V&V.

o

Ver anotações

ASSIMILE

A verificação consiste em analisar o software para ver se ele está sendo construído de acordo com o que foi especificado.

A validação consiste em analisar o software construído para ver se ele atende às verdadeiras necessidades dos interessados. Assim, a pergunta-chave para a validação é “Estamos fazendo a coisa certa?”, enquanto que a pergunta-chave para a verificação é “Estamos fazendo a coisa do jeito certo”? (WAZLAWICK, 2013).

Conforme mencionamos, a verificação e a validação estão incluídas em um escopo mais abrangente e estão vinculadas à Garantia da Qualidade do Software (*Software Quality Assurance* ou SQA). Na visão de Pressman e Maxim (2016), a verificação e a validação incluem grande variedade de atividades de SQA, quais sejam as revisões técnicas, auditorias de qualidade, monitoramento do desempenho, simulação, estudo de viabilidade, teste de usabilidade e testes de aceitação e de instalação. Os autores complementam que, embora a aplicação de teste tenha um papel extremamente importante em V&V, muitas outras atividades são necessárias.

À propósito, a menção aos testes vem a calhar. Como podemos posicioná-lo nesse contexto? O teste é a última frente de preservação da qualidade, mas não pode ser entendido como a garantia total de que a produção entregue pelas equipes está livre de defeitos. O teste proporciona o último elemento a partir do qual a qualidade pode ser estimada e, de forma mais pragmática, os defeitos podem ser encontrados. A qualidade é incorporada ao software por meio da correta aplicação das técnicas da Engenharia de Software e é confirmada durante o teste (PRESSMAN; MAXIM, 2016).

•
Ver anotações

| TESTES DE SOFTWARE

Um teste não é um procedimento isolado que pode ser concluído por um único membro da equipe. Embora seja comum tratá-lo por “teste”, sua execução depende de um conjunto de ações e procedimentos executados por vários elementos da equipe de desenvolvimento. Por isso, melhor seria chamá-lo de processo de teste, já que formalmente ele representa uma sequência de ações executadas com o objetivo de encontrar problemas no software, o que aumenta a percepção da qualidade geral dele e garante que o usuário final tenha um produto que atenda às suas necessidades (PINHEIRO, 2015).

É necessário, no entanto, destacar que o objetivo do teste não é o de garantir que um programa seja absolutamente livre de defeitos, mas o de encontrar problemas no software. Por mais que essa premissa nos soe estranha (e um pouco frustrante) ela deriva do fato de que nenhum teste é capaz de assegurar 100% de ausência de defeitos em um sistema. Logo, se o processo de teste não revelar defeitos, há que se aprimorar o processo de forma geral. Não se pode considerar que o sistema não possui problemas se o teste não os revelar.

Mas, afinal, o que é teste de software? Como ele se efetiva? Há um plano a ser executado?

IEEE (2014) nos oferece o seguinte conceito: o teste de software consiste na verificação dinâmica de que um programa, de fato, fornece comportamentos esperados em um conjunto finito de casos de teste adequadamente selecionados do domínio de execução geralmente infinito.

É natural que uma definição retirada de uma publicação de caráter estritamente técnico peça uma explicação que a aproxime da percepção comum que temos a respeito de teste.

A expressão “verificação dinâmica” indica que aplicar um teste significa, necessariamente, executar o programa. Além disso, essa execução deve ser baseada em entradas previamente selecionadas. Conforme estudaremos em detalhes na sequência, casos de teste são entradas fornecidas ao programa e às respectivas saídas esperadas. Na teoria, as entradas possíveis para um programa são infinitas, daí a necessidade de restringi-las em um “conjunto finito de casos de teste” escolhidos criteriosamente.

EXEMPLIFICANDO

Antes de seguirmos em frente em nosso conteúdo, vale a pena fazermos um exercício de imaginação que colocará você diante da necessidade imperiosa de se testar um software (ou função dele) antes de exibi-lo a alguém ou de colocá-lo em funcionamento. Você assumiu a missão de criar uma nova funcionalidade para o sistema que já se encontrava em operação, com a promessa de facilitar o controle do estoque dos produtos controlados pela empresa em que atua.

Na urgência de apresentar a nova funcionalidade, você reuniu os diretores e envolvidos com os estoques e, no primeiro acesso à função, um erro se manifestou no programa e sua execução foi interrompida. Feita a correção ali mesmo durante a demonstração, você resolveu confrontar

a quantidade física de um certo item de estoque com a quantidade fornecida pelo sistema. Novamente uma falha se manifestou e uma resposta inesperada foi obtida. Dois tipos de problemas se tornaram visíveis na apresentação e a percepção de qualidade em relação à sua função sofreu abalo, para dizer o mínimo. Com os testes adequados, tal constrangimento teria sido facilmente evitado.

o

Ver anotações

Bem, mas como os testes são feitos? Há uma ferramenta computacional que os execute? Nas seções futuras desta unidade, teremos a oportunidade de abordar esses assuntos com mais riqueza de detalhes, mas convém termos agora uma rápida visão de um programa sendo testado. Para que isso seja possível, algumas premissas devem ser apresentadas:

- O programa em teste foi criado e está sendo executado no Eclipse, um importante ambiente integrado de desenvolvimento (ou IDE – *Integrated Development Environment*) utilizado principalmente para criação e para teste de aplicações Java.
- O tipo de teste em questão é o de unidade. Por meio dele, apenas uma unidade do programa (uma função ou uma classe, por exemplo) é testada e não o programa todo.
- A ferramenta utilizada para a efetivação do teste é o JUnit, que já se encontra instalada nativamente no Eclipse.

Na Figura 3.1, você vê uma classe – chamada `CalculoTest` –, que representa um caso de teste. Note que há variáveis com os valores 10 e 5 e uma terceira variável que contém o valor esperado para a execução da unidade que, no caso, realiza uma operação de soma.

Figura 3.1 | Classe de teste escrita em Java

```

1 package processos;
2 import junit.framework.TestCase;
3 public class CalculoTest extends TestCase{
4     public void testeExecutaCalculo() {
5         //Define os valores a serem calculados e testados
6         float PassaValor1 = 10;
7         float PassaValor2 = 5;
8         float RetornoEsperado = 15;
9         //Executa o método "ExecutaCalculo" da classe Calculo e
10        //armazena o resultado em uma variável
11        float RetornoFeito = Calculo.ExecutaCalculo(PassaValor1, PassaValor2);
12        //compara o valor retornado com o valor esperado
13        assertEquals (RetornoEsperado, RetornoFeito, 0);
14    }
15
16
17 }

```

Fonte: Medeiros (2009, [s.p.]).

A classe a ser testada para este caso de teste está descrita na Figura 3.2 e se chama Calculo.

Figura 3.2 | Classe de teste escrita em Java

```

1 package processos;
2 public class Calculo {
3     public void testeExecutaCalculo() {
4         public static float ExecutaCalculo (float Valor1, float Valor2) {
5             float Soma = Valor1 + Valor2;
6             return Soma;
7         }
8     }
9 }
10
11
12 }

```

Fonte: Medeiros (2009, [s.p.]).

A execução da classe CalculoTest através do JUnit retornará sucesso para o caso de teste em questão. Dessa forma, uma função pode ser testada por meio dessa ferramenta.

■ PLANO DE TESTES

Bem, como já compreendemos o conceito de teste, sua efetivação dependerá apenas da iniciativa do seu desenvolvedor em executar o programa com base em alguns casos de teste, não é mesmo?

Responder “sim” a essa pergunta equivale a ignorar uma etapa absolutamente indispensável neste contexto: o planejamento.

Pressman e Maxim (2016) ensinam que teste é um conjunto de atividades que precisam ser planejadas com antecedência e executadas com base em um procedimento padrão, fato que motiva a criação de um modelo para o teste. Tal modelo, segundo os autores, deverá prever o emprego de técnicas específicas no projeto de casos de teste e no método de teste.

Uma definição que deve fazer parte do plano de testes é a de quem deve executá-los. Schach (2008) pondera que a realização dos testes equivale a um processo destrutivo e, como era de se esperar, um programador não desejará “destruir” seu próprio trabalho. Por isso, atribuir a atividade de teste ao mesmo time que desenvolveu o produto certamente não é uma boa ideia, já que é grande a chance de a equipe entender que deve proteger seu programa e não deve criar testes que possam revelar, de fato, problemas no código.

Outra razão para evitar a designação dos criadores do programa como seus testadores é o fato de que um terceiro poderá detectar uma falha no entendimento dos requisitos que passou despercebida ao programador e que foi implementada incorretamente. Um teste feito por outras pessoas pode aumentar a chance de descoberta do problema antes que ele tenha reflexos na operação do cliente.

REFLITA

O programador que desenvolveu o produto deve ser sumariamente excluído do procedimento de teste? Não seria prudente afastar dos testes alguém que conhece como ninguém o produto e que precisou se aprofundar nos requisitos antes de começar a desenvolvê-lo. Além disso, em um procedimento de depuração – que será detalhado adiante – as causas suspeitas de um problema poderão ser mais facilmente levantadas pelo próprio desenvolvedor do sistema do que por outro testador.

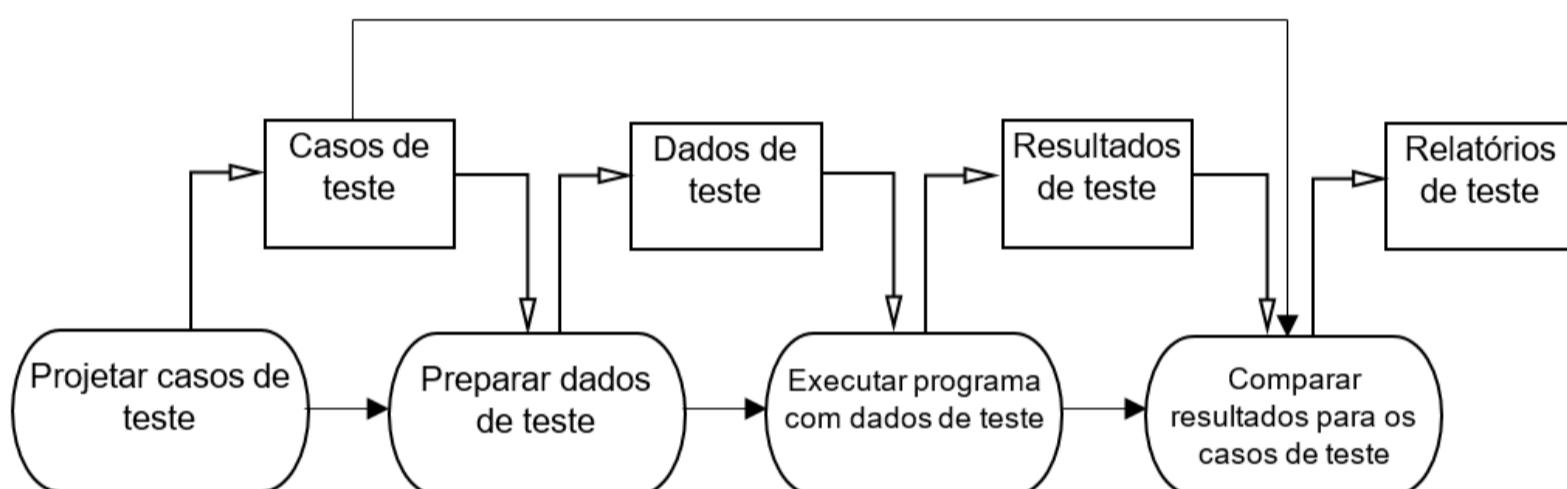
Isto posto, é necessário mencionar que um plano de teste é normalmente separado em quatro grandes etapas:

- **Planejamento:** nesta etapa deve ser definido quem executa os testes, em que período, com quais recursos (ferramentas e computadores, por exemplo) e qual será a técnica utilizada (técnica estrutural ou técnica funcional, por exemplo)

- **Projeto de casos de teste:** aqui são definidos os casos de teste que serão utilizados no processo. No próximo item, esse conceito será detalhado.
- **Execução do programa com os casos de teste:** nesta etapa, o teste é efetivamente realizado.
- **Análise dos resultados:** aqui se verifica se os testes retornaram resultados satisfatórios.

A Figura 3.3 ilustra um modelo de plano de teste.

Figura 3.3 | Modelo de processo de teste



Fonte: Sommerville (2018, p. 207).

Bem, aparentemente os casos de teste são, de fato, o elemento central no processo de teste.

EXEMPLIFICANDO

Porém, antes de iniciarmos essa abordagem, apresentamos uma experiência relacionada a planos de teste. Embora fictícia, ela ilustra bem como alguns detalhes de planejamento podem ser decisivos para o sucesso ou para o fracasso da missão de se testar um produto. Imagine uma empresa de desenvolvimento de software que, depois de passar muitos anos terceirizando a atividade de teste, resolveu realizá-la com sua própria equipe. Em sua primeira experiência de teste, a equipe criou um plano pautado no modelo ilustrado na Figura 3.3 e suas linhas gerais estão abaixo reproduzidas:

1. Todos os desenvolvedores que atuaram na criação do software mais a recém-criada equipe de testadores deveriam atuar no processo de teste. Eles deveriam selecionar casos de teste que exercitassem todas as funções do programa, criar dados de entrada para testar o programa e executá-lo com os dados de teste.

2. Outro membro da equipe, que até então não havia participado de nenhum procedimento, deveria analisar os resultados dos testes e produzir um relatório de como eles haviam sido feitos e quais os resultados obtidos.

Conforme planejado, os desenvolvedores e os testadores puseram-se a criar os casos de teste, planejaram as entradas para o programa e, uma vez concluídas essas atividades, executaram-no. Terminadas as execuções, disponibilizaram ao outro membro da equipe o resultado do teste fornecido pela ferramenta que haviam utilizado e foi a partir daí que a primeira experiência de testes da empresa desenvolvedora começou a se distanciar do sucesso.

Esse novo elemento havia tido apenas um rápido contato com os requisitos do sistema, além de não ter participado do projeto (design) e nem da implementação do produto.

Embora sua primeira atitude ao receber a incumbência de analisar os resultados tenha sido a de buscar informações sobre o sistema, melhor seria se ele tivesse acompanhado todo o processo de criação do produto, incluindo a sua implementação. Outro fator que dificultou bastante a primeira experiência foi o fato de que a comparação dos resultados previstos com os resultados obtidos começou a ser feita manualmente e, portanto, sem a utilização dos relatórios que poderiam ser fornecidos pela ferramenta de teste utilizada. Uma vez sanada a falta de conhecimento

sobre o sistema do novo membro da equipe e utilizados os dados comparativos fornecidos pela ferramenta de teste, o procedimento foi levado finalmente a bom termo.

CASOS DE TESTE

Um caso de teste é o par formado por uma entrada possível a ser dada no programa e a correspondente saída esperada, também dada pelo programa e de acordo com os requisitos previamente especificados.

Nesse caso, devemos entender o conceito de entrada como o conjunto de dados necessários para uma execução do programa e o de saída esperada como o resultado daquela execução ou de função específica. Com um exemplo esses conceitos serão mais bem esclarecidos: imagine que você esteja testando um programa (ou função) que promove a validação de datas inseridas pelo usuário. Um caso de teste possível seria formado pelo par (31/12/2020; válida). Ao receber a entrada 31/12/2020, a função de validação deveria retornar “data válida”.

A boa escolha dos casos de teste é fator crítico para o sucesso da atividade. Um conjunto de casos de teste de baixa qualidade pode não exercitar partes críticas do programa e, em consequência disso, acabar não revelando defeitos no código. O que aconteceria, por exemplo, se o responsável pelos testes usasse apenas datas válidas como entradas?

No mínimo, a parte do programa que trata das datas inválidas não seria executada, o que prejudicaria a confiabilidade do processo de teste e do produto testado.

EXEMPLIFICANDO

Observe os casos de teste a seguir:

`t1= {(15/2/1946; data válida), (30/2/2022; data inválida);
(15/13/2023; data inválida); (29/2/2016; data válida),
(29/2/2015; data inválida), (##/1/1985; data inválida)}.` Vamos analisar cada um dos casos de teste do conjunto t1:

- A entrada do primeiro caso de testes (15/2/1946) é formada por um dia válido (ou seja, contido no intervalo entre 1 e 31), um mês válido (contido no intervalo entre 1 e 12) e um ano válido. Essa entrada, que deverá provocar o retorno da mensagem “data válida” irá, portanto, exercitar trechos do programa criados para tratamento de dia, mês e ano simultaneamente válidos.
- A entrada 30/2/2022 também é formada por dia, mês e anos situados em intervalos válidos. No entanto, para o mês 2, o dia 30 é considerado inválido, pois o mês de fevereiro possui 28 ou 29 dias apenas. Assim, embora uma análise isolada de cada unidade da data possa indicar sua validade, a combinação do dia com o mês a torna inválida. O trecho de programa a ser exercitado, portanto, será diferente daquele exercitado no caso de teste anterior.
- Já a entrada 15/13/2023 também deverá retornar a mensagem “data inválida”, pois o mês está fora do intervalo permitido, independentemente do ano e do dia escolhidos. Mais uma vez, o trecho do programa de validação de data a ser exercitado será distinto dos anteriores.
- Prosseguimos com as entradas 29/2/2016 e 29/2/2015. Elas retornam, respectivamente, “data válida” e “data inválida”. Embora os dias e meses dessas datas sejam idênticos, 2016 foi ano bissexto, o que torna válida a primeira das datas. Já no caso de 2015, o dia 29 de fevereiro não fez parte do calendário daquele ano, o que torna a data inválida. Novamente o fluxo do programa em questão seguirá trechos específicos para realizar as validações.

- Por fim, a última data de entrada deverá retornar a mensagem “data inválida” devido a um caractere não numérico na unidade do dia.

Embora exista um conjunto muito maior de entradas possíveis, esse conjunto de casos de teste será capaz de exercitar grande parte do código, o que aumentará a chance de descoberta de defeitos.

Será, entretanto, que os casos de teste apresentam apenas esse aspecto? Observe o desenvolvimento de outro exemplo: imagine que o cenário de teste agora seja a checagem da funcionalidade de login em um sistema de reserva de passagens feito por agentes de viagens. A verificação a ser feita se apoia na resposta do sistema a diversos padrões de entrada de nome de usuário e de senha. Selecionamos três deles para fins de exemplificação:

1. Checagem da resposta do sistema no caso de o agente entrar com nome de usuário e senha válidos.
2. Checagem de resposta do sistema no caso de o agente entrar com nome de usuário e/ou senha inválidos.
3. Checagem de resposta do sistema no caso de o agente pressionar a tecla Enter com o campo de nome de usuário vazio.

Para que possamos tornar mais específico nosso cenário, trataremos apenas do primeiro item e, com base nele, apresentamos o Quadro 3.1, que descreve um caso de teste relacionado.

Quadro 3.1 | Cenário detalhado do teste e um caso de teste possível

Cenário	Caso de teste	Passos do teste	Dados de entrada	Resultado esperado	Recomendação
Checagem da funcionalidade de login.	Checagem da resposta ao se inserir nome de usuário e senha válidos.	a. Executar a aplicação. b. Informar o nome do agente. c. Informar a senha. d. Acionar o botão “Ok”.	Nome agente: Marcio Senha: 5555	O login deve ser bem-sucedido.	Lo be su

Ver anotações

Fonte: adaptado de How... (2014).

Dessa forma, o caso de teste estará mais bem detalhado e as condições para que seja verificado estarão especificadas por completo. Note que até os passos para a realização do teste – que nos parecem tão óbvios – estão descritos nele. Fica claro que o sucesso no procedimento de testes está diretamente relacionado à boa escolha e ao bom uso dos casos de teste. Idealmente, cada conjunto de casos de teste deverá estar associado a um grande requisito diferente a ser testado. Para que não se corra o risco de defini-los incorretamente, é necessário planejamento e bom conhecimento da aplicação. Uma boa forma de se abordar o problema é a que segue, segundo Pinheiro (2015): definir o ambiente no qual o teste será realizado, definir a entrada desse caso de teste, definir a saída esperada para cada entrada e definir os passos a serem realizados para executar os testes.

Quando um caso de teste é executado, seu resultado deve ser coletado. Podemos assumir diferentes abordagens para definir o resultado da aplicação de um caso de teste específico. A mais comum define as seguintes opções (PINHEIRO, 2015):

- **Passou:** todos os passos do caso de teste foram executados com sucesso para todas as entradas.
- **Falhou:** nem todos os passos foram executados com sucesso para uma ou mais entradas.
- **Bloqueado:** o teste não pôde ser executado, pois o seu ambiente não pôde ser configurado.

REFLITA

O que é um teste bem-sucedido? Imagine que determinada equipe tenha planejado um procedimento de teste, feito a escolha dos casos de teste e, após a execução do procedimento, tenha encontrado 0 (zero) defeito no código. Podemos afirmar que tal teste tenha sido bem-sucedido? Pense melhor e considere se há (ou houve) algum sistema que tenha sido elaborado 100% livre de problemas. Será que o conjunto de casos de teste foi corretamente selecionado?

| DEPURAÇÃO

Enquanto testar significa executar o software para encontrar defeitos desconhecidos, a depuração (ou *debug*) é a atividade que consiste em buscar a localização desses defeitos no código. O fato de saber que há um problema causador de erro no programa não significa, necessariamente, que o testador sabe também em qual ou em quais linhas o problema está. Os ambientes de programação atuais oferecem recursos para depuração do programa e, durante esse processo, o valor assumido pelas variáveis sob inspeção em cada passo do algoritmo

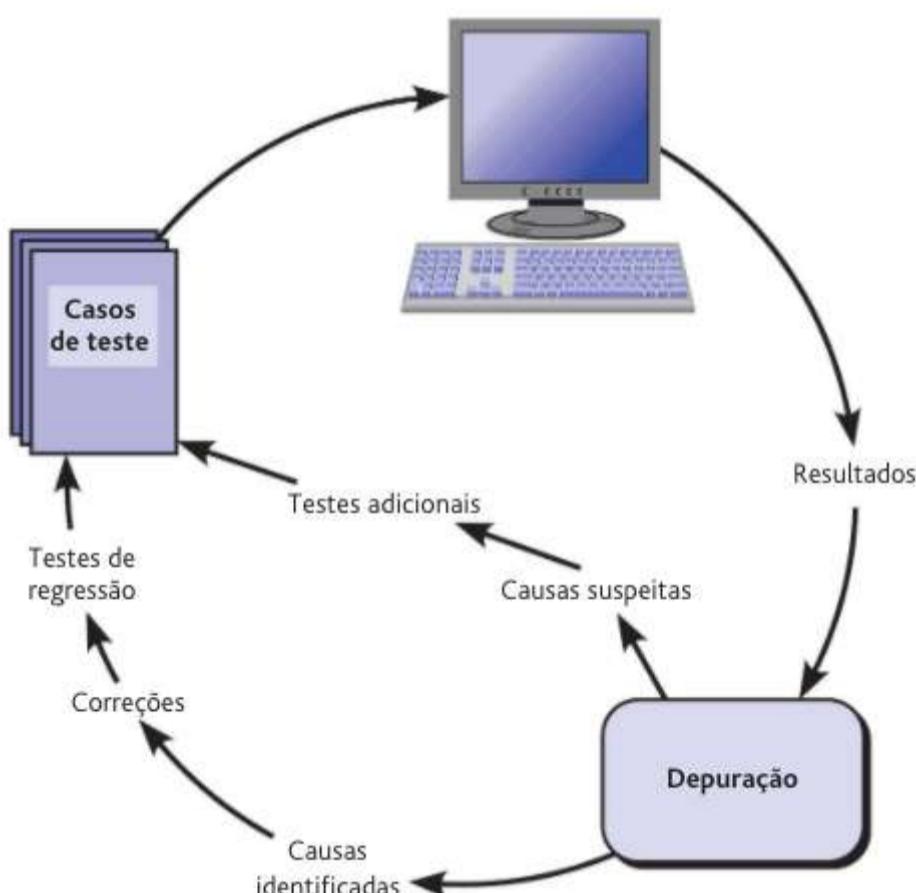
pode ser observado. Além disso, alguns pontos de parada da execução do programa podem ser inseridos no código. Tudo para possibilitar que o testador identifique e isole o defeito no código.

Na visão de Pressman e Maxim (2016), a depuração ocorre como consequência de um teste bem-sucedido, ou seja, quando um caso de teste descobre um defeito. Nesse caso, a depuração é o processo que encontra e remove o erro. Embora não seja um teste, ela ocorre como um desdobramento dele e começa com a execução de um caso de teste. O procedimento de depuração pode apresentar um entre dois resultados possíveis:

- A causa é encontrada e corrigida e a depuração daquele problema é bem-sucedida.
- A causa não é encontrada e, nesse caso, o profissional que está realizando o procedimento tenta outro caso de teste que lhe pareça mais adequado para aquela causa.

A Figura 3.4 ilustra um procedimento de depuração:

Figura 3.4 | O processo de depuração



Fonte: Pressman e Maxim (2016, p. 489).

A figura anterior nos revela o passo a passo do processo de depuração:

1. A depuração começa com a execução de um caso de teste.

2. Os resultados são avaliados e o desempenho apurado é diferente do desempenho esperado para aquele caso de teste.

3. Quando as causas do defeito são plenamente identificadas, o problema é corrigido. Não havendo a perfeita correspondência entre o problema e a causa, procuram-se causas suspeitas.

Observe a natureza cíclica do procedimento e a colocação de “causas suspeitas” como um dos elementos do ciclo. Essa nomenclatura revela que, embora a depuração deva obedecer a um processo ordenado, ela depende bastante ainda da experiência e da sensibilidade do testador.

Um engenheiro de software, ao avaliar os resultados do teste, pode perceber sintomas de um problema e não a sua manifestação inequívoca, o que usualmente indica que a manifestação externa do problema e a sua real causa interna podem não ter nenhuma relação aparente uma com a outra.

Nesse ponto, vale a pena observarmos um caso em que a depuração é necessária e, para esse fim, utilizaremos uma aplicação simples em Java e o ambiente integrado de desenvolvimento Eclipse. Originalmente a aplicação deve permitir a digitação de cinco números inteiros e, ao final, informar o maior número digitado. No entanto, o comportamento do programa não é o esperado e alguns recursos de depuração do ambiente de desenvolvimento devem ser usados para que os defeitos sejam encontrados. O Código 3.1 exibe o código-fonte da aplicação.

Código 3.1 | Aplicação de retorno do maior valor digitado, com dois defeitos inseridos

```
1 import java.util.Scanner;
2 public class Maior {
3     public static void main (String args[]) {
4         Scanner entrada = new Scanner(System.in);
5         int i, x=0, valor;
6         for (i=1;i<7;i++) {
7             System.out.printf("\nDigite o %d valor: ",i);
8             valor = entrada.nextInt();
9             if (valor < x) x = valor;
10        }
11        System.out.printf("\nO maior valor inserido eh:
12            %d ",x);
13    }
```

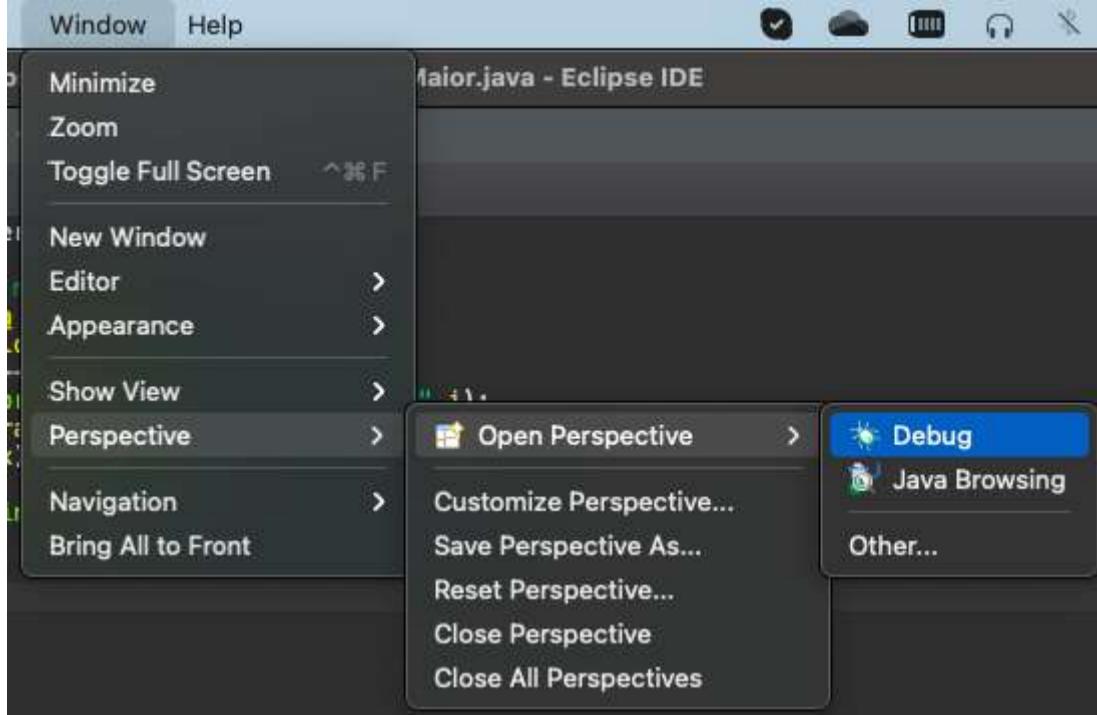
Ver anotações

Fonte: elaborado pelo autor.

Há dois defeitos nesse código: o primeiro está situado na linha 6: ao invés de permitir a digitação de cinco números, a aplicação solicitará que seis números sejam digitados, dada a comparação feita com valor menor que 7 e não menor que 6. O segundo defeito está situado na linha 9, local em que o teste para apuração do maior valor a cada iteração está invertido. Assim, o valor que retornará a aplicação será o menor e não o maior, como se espera.

A utilização dos recursos de depuração do Eclipse começa com a mudança da perspectiva de visualização. Na opção de menu Window, acesse Perspective > Open Perspective > Debug, conforme ilustra a Figura 3.5. Essa providência fornece um padrão de dados adequado para a execução da depuração.

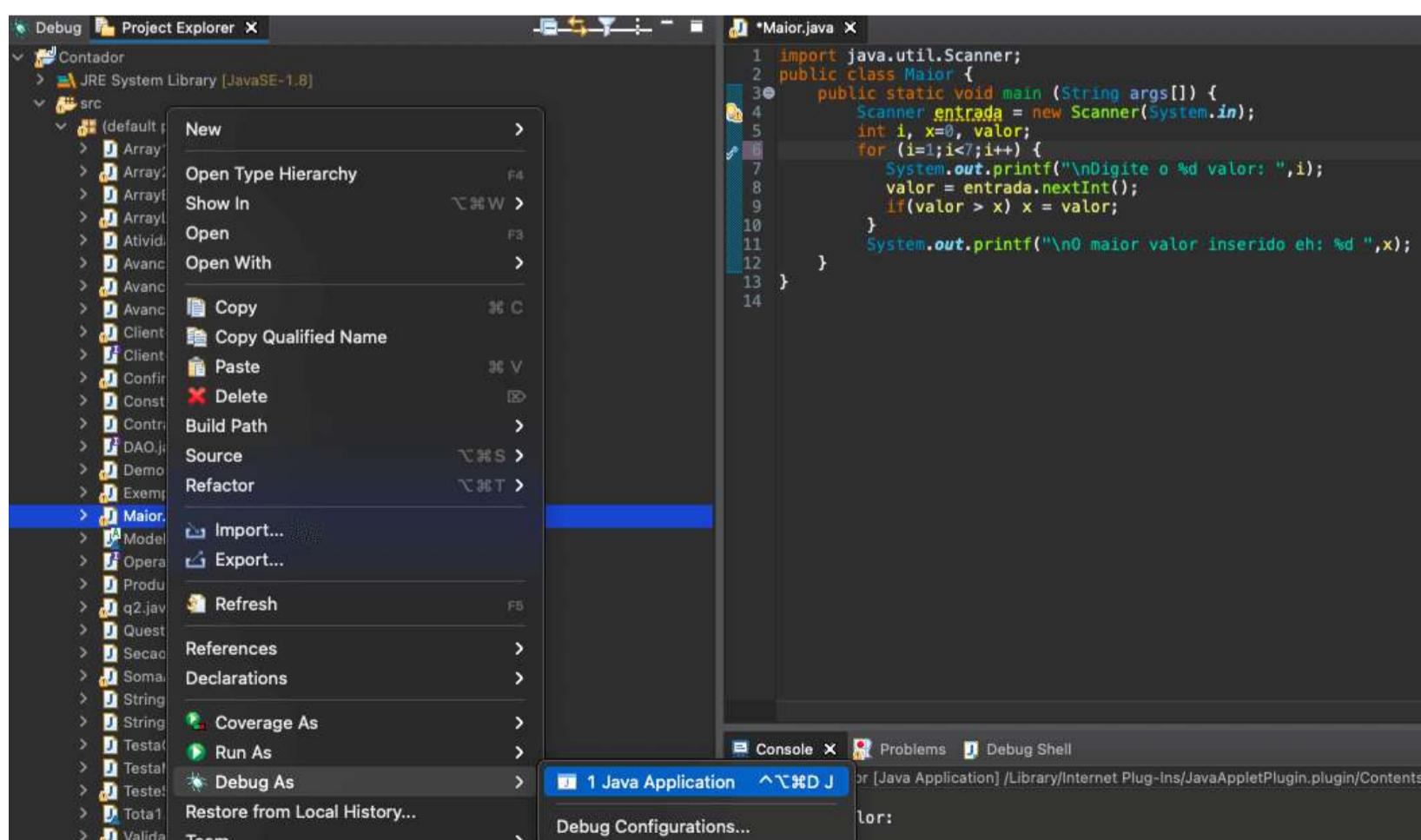
Figura 3.5 | Alteração da perspectiva para o modo depuração



Fonte: captura de tela do IDE Eclipse elaborada pelo autor.

O primeiro passo efetivo do processo se dá com a marcação da linha do programa em que se deseja iniciar, de fato, a depuração. Essa ação é realizada por meio do posicionamento do cursor na linha desejada e o acesso à opção Run > Toggle Breakpoint. Uma marca na parte esquerda da linha será criada e, nesse momento, você estará apto a executar a aplicação no modo Debug, conforme mostra a Figura 3.6. Posicione o mouse sobre a classe (cujo nome deve aparecer no lado esquerdo da tela), acione o botão direito do mouse sobre ela e acesse a opção Debug As.

Figura 3.6 | Execução da aplicação no modo debug



Fonte: captura de tela do IDE Eclipse elaborada pelo autor.

Neste momento a aplicação será executada no modo de depuração e os valores das variáveis poderão ser verificados passo a passo. Dessa forma, será mais fácil identificar as anomalias inseridas no código e corrigi-las.

Essa foi a base introdutória de testes que queríamos compartilhar com você. Há muito ainda a tratar sobre esse procedimento, mas é com a correta compreensão dos conceitos fundamentais que teremos sucesso na continuidade do nosso estudo. Iniciamos a seção tratando da conceituação de verificação e de validação e, na sequência, abordamos o conceito de testes. Depois o situamos como um procedimento e demos a ele uma sequência de etapas. Nesse mesmo contexto, tratamos do plano de testes e demos relevância ao papel do desenvolvedor no processo de teste de seu próprio produto. Por fim, abordamos os casos de teste e a depuração como elementos que darão subsídio à continuidade dos nossos estudos sobre teste.

FAÇA VALER A PENA

Questão 1

Em relação aos casos de teste, analise as afirmações que seguem:

1. Um caso de teste é o par formado por uma entrada no programa e a correspondente saída esperada.
2. Um caso de teste equivale a uma seção em que os testes são realizados.
3. A escolha correta dos casos de teste tem importância relativa no processo, já que é feita pelo cliente.
4. Os casos de teste são específicos para cada programa submetido a teste.

É verdadeiro o que se afirma em:

a. II apenas.

b. IV apenas.

c. I e IV apenas.

d. II e III apenas.

e. II, III e IV apenas.

Questão 2

Leia a sentença a seguir:

A _____ é executada ao final de uma etapa do desenvolvimento de um produto e tem como objetivo determinar se o trabalho que o gerou foi executado _____. Já a _____ corresponde ao processo de avaliação empreendido _____ de o produto ser disponibilizado ao cliente.

Considerando os conceitos de V&V e os tempos em que são executadas, assinale a alternativa que contenha a sequência correta de termos que completam a sentença dada.

a. verificação; corretamente; validação; antes.

b. verificação; rapidamente; validação; depois.

c. validação; rapidamente; verificação; depois.

d. validação; pelas pessoas certas; verificação; depois.

e. verificação; corretamente; verificação; antes.

Questão 3

O procedimento de depuração compõe as ações que visam conferir qualidade a um produto de software. Considerando o conceito, o tempo em que é aplicada e os responsáveis pela aplicação da depuração, analise as afirmações seguintes:

- I. Justamente por ter criado o programa, seu desenvolvedor deve permanecer fora do processo de depuração.
- II. A depuração é uma das etapas do processo de verificação, já que é feita antes de um certo artefato ser entregue.

III. Embora conte com um procedimento de execução objetivo, a depuração pode carecer de elementos subjetivos de quem a aplica.

É verdadeiro o que se afirma em:

a. I e II apenas.

b. II apenas.

c. I e III apenas.

d. II e III apenas.

e. III apenas.

REFERÊNCIAS

HOW to write a TEST CASE? Software Testing Tutorial. [S.I.: s.n.], 2014. 1 vídeo (3 min). Publicado pelo canal Guru99. Disponível em: <https://bit.ly/3poU6ZV>. Acesso em: 14 de dez. 2020.

IEEE Computer Society. **Guide to the Software Engineering Body of Knowledge**. Piscataway: The Institute of Electrical and Electronic Engineers, 2014.

MEDEIROS, M. P. JUnit Tutorial. **DevMedia**, [S.I.], 2009. Disponível em: <https://bit.ly/3sY9ApL>. Acesso em: 17 dez. 2020.

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

PINHEIRO, V. Um comparativo na execução de testes manuais e testes de aceitação automatizados em uma aplicação web. SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE (SBQS), 14., 2015, Manaus. **Anais** [...]. Manaus: Uninorte, 2015.

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

SANTOS, L. D. V., OLIVEIRA, C. V. S.; **Introdução à Garantia de Qualidade de Software**. Timburi: Cia do e-book, 2017.

SCHACH, S. R. **Engenharia de Software**: os paradigmas clássicos e orientados a objetos. 7. ed. São Paulo: McGraw-Hill, 2009.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2018. E-book. Disponível em: <https://bit.ly/3a6T5PN>. Acesso em: 6 dez. 2020.

WAZLAWICK, R. S. **Engenharia de software**: conceitos e práticas. Rio de Janeiro: Elsevier, 2013.

FOCO NO MERCADO DE TRABALHO

CONCEITOS DE TESTES DE SOFTWARE

Roque Maitino Neto

Ver anotações 0

PROCESSO DE DEPURAÇÃO

A depuração (ou *debug*) é a atividade que consiste em buscar a localização dos defeitos no código, e para isso é necessário a implementação de um algoritmo em um ambiente integrado de desenvolvimento.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Esta situação-problema apresenta especificidades que dificultam uma resposta objetiva. Em primeiro lugar, a escolha da linguagem de programação para a implementação do algoritmo fica a cargo do aluno. Além disso, o ambiente integrado de desenvolvimento deverá variar em função da linguagem escolhida, fato que também sugere multiplicidade de respostas. Por fim, para ambientes de desenvolvimento diferentes, há recursos de depuração de código também diferentes.

O fato objetivo nesta atividade é que o algoritmo – que deveria calcular o fatorial de um número fornecido pelo usuário – apresenta problemas. Ao submeter o código ao processo de depuração e ao colocar as variáveis sob inspeção, você perceberá que:

- A variável *f* não assumirá o valor que dela se espera, já que ela foi iniciada em 0, valor neutro da adição. Para que ela contivesse o valor do fatorial ao final do processamento, seu valor inicial deveria ser 1, que representa o elemento neutro da multiplicação.
- A validação do dado de entrada – que é feita no corpo do laço *faça..enquanto* – falhará, já que a condição de parada jamais será satisfeita, pois não há um número que seja menor que 1 E maior que 10 ao mesmo tempo.

Para melhor exemplificar essa resolução, segue o código correto da aplicação na linguagem C.

Código 3.2 | Código na linguagem C

```
1 #include <stdio.h>
2
3 main()
4 {
5     int i, valor, fatorial = 1;
6
7     printf("Programa que calcula o fatorial de um valor
8 informado pelo usuario\n");
9
10    do {
11        printf("\nInforme um valor entre 1 e 10: ");
12        scanf("%d",&valor);
13        } while ((valor<1) || (valor>10));
14
15        for (i=1; i<=valor; i++)
16            fatorial=fatorial*i;
17
18        printf("\nO Fatorial de %d = %d", valor, fatorial);
19
20        printf("\n");
21 }
```

Ver anotações

Fonte: elaborado pelo autor.

Embora sem complexidade, essa situação ilustra a utilidade da aplicação da depuração –especialmente seu recurso de inspeção de variáveis como forma de encontrar os defeitos revelados no teste. A depuração, inclusive, dará ao desenvolvedor condições para que defeitos simples no código sejam encontrados antes mesmo de seu produto seguir para a fase de testes, o que aumentará a percepção de qualidade e reduzirá esforços de teste.

NÃO PODE FALTAR

TIPOS DE TESTE

Roque Maitino Neto

Ver anotações

COMO OS TESTES SÃO CLASSIFICADOS?

Os testes são classificados em funcionais, estruturais, voltados a aplicações móveis, e a aplicações orientadas a objetos.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

PRATICAR PARA APRENDER

Prezada aluna, caro aluno, aqui estamos para mais uma seção dedicada aos testes de software. Depois de nosso primeiro contato com o assunto, é chegado o momento de avançarmos para temas mais específicos e relacionados a técnicas de teste. Em termos simples, uma técnica representa um jeito de se fazer e, durante nosso estudo, você entenderá o motivo de termos mais do que uma maneira de fazer testes. Aplicações imediatas das técnicas ajudarão no entendimento do que se pretende com cada uma delas. Depois dessa primeira abordagem, prosseguiremos com os testes para aplicações móveis, web e orientadas a objetos, como forma de esclarecermos as especificidades de cada um deles.

Durante nosso estudo da técnica funcional, alguns tipos de testes serão descritos e um exemplo prático de teste de componente será desenvolvido. No teste estrutural, trataremos de detalhar a transformação de um código-fonte simples em um grafo, elemento gráfico que o representará e permitirá que uma simulação de cobertura de testes seja feita. Já nas divisões de testes para aplicações móveis, web e orientadas a objetos, serão fornecidas particularidades dessas aplicações, as quais definirão como o procedimento de teste deverá ser conduzido. Com isso, esperamos prepará-lo para tomar decisões quanto a técnicas e tipos de teste a serem aplicados em diferentes ocasiões.

Com a intenção de expandir a frente de atuação em desenvolvimento e testes de sistemas, os gestores de uma determinada empresa de software decidiram que começariam a realizar procedimentos de testes para terceiros. Dessa forma, qualquer desenvolvedor que desejasse terceirizar o teste de seus produtos poderia procurar por essa empresa, e os testes seriam feitos mediante contrato e contrapartida financeira.

Na condição de um dos melhores profissionais dessa empresa, você foi destacado para assumir o primeiro projeto de teste e logo se deparou com um obstáculo: por força de cláusula contratual, você não teria

acesso ao código-fonte daquela aplicação, que fora projetada para ser executada em computadores pessoais conectados em rede local.

O desenvolvedor que confiou os testes à empresa em que você atua mostrou-se absolutamente disponível para fornecer todos os elementos do sistema a você, exceto o código-fonte. Considerando esse cenário, você foi chamado a planejar o procedimento de teste e, com a finalidade de direcionar seus esforços, seu gestor apontou quatro itens que deveriam obrigatoriamente compor o planejamento:

- **O documento que deverá ser solicitado à organização desenvolvedora:**

considere que as funções a serem testadas devem estar descritas em algum documento e é exatamente esse documento que você deverá solicitar ao seu cliente.

- **A técnica de teste que deverá ser utilizada:** dentre as técnicas abordadas, você deve escolher uma delas e explicar o motivo da escolha.

- **Critério que utilizará para a escolha dos casos de teste:** sua descrição neste tópico deve se basear na melhor escolha de casos de teste para o perfil do sistema em teste.

- **Definição de quem deverá participar dos procedimentos de teste:** especifique neste item quem deverá participar dos testes, considerando o conhecimento do sistema que um ou outro elemento pode possuir.

Bom trabalho!

DICA

Quanto maior a sua dedicação ao aprendizado do assunto, maior será a afinidade conquistada com ele. E quanto maior a afinidade, maior será seu interesse. Como já sabemos, testar um produto de software está muito longe de apenas promover algumas execuções de um produto com algumas entradas escoinhas ao acaso. Ao invés disso, testar significa

aplicar técnicas, melhores práticas e critérios para que a qualidade do produto atinja os níveis de excelência que todos desejam.

Siga conosco e bons estudos!

CONCEITO-CHAVE

Como já se sabe, um produto de software necessita de inúmeros elementos para oferecer ao seu usuário um funcionamento pleno. A adequação do hardware e da infraestrutura de rede são apenas dois desses elementos, e o dimensionamento incorreto de um deles terá o potencial de arruinar a experiência do usuário. Há, no entanto, um aspecto que sempre terá papel decisivo na percepção do usuário em relação ao sistema que utiliza: as suas funções. Afinal, será por meio delas que ele interagirá com o programa e será através delas que alcançará seus objetivos traçados lá na fase de requisitos. Funções mal projetadas ou defeituosas podem, inclusive, desestimular o uso do sistema. Com tamanha importância, a área de testes não poderia deixar de direcionar muita atenção às funções do sistema e o faz por meio da aplicação da Técnica de Teste Funcional e do Teste de Funcionalidades, assuntos que desenvolveremos na sequência.

ESTRATÉGIAS DE TESTES

Entretanto, antes de abordarmos especificamente técnicas de teste, vale a pena as situarmos no contexto da **estratégia de teste**, a qual equivale a uma prática cujo objetivo é estabelecer um roteiro que descreve os passos a serem executados no processo de teste ou, em outras palavras, um modelo para os testes. Pressman e Maxim (2016) descrevem genericamente alguns elementos presentes em todas as estratégias adotadas para os testes:

- **Revisões de software:** boas revisões eliminam problemas no código antes da efetiva aplicação dos testes.

- **Progressão do teste:** os testes devem começar em uma unidade do software e progredir em direção à integração do sistema como um todo.
- **Depuração:** esta atividade deve estar associada ao teste embora sejam elementos distintos entre si.
- **Técnicas:** diferentes técnicas de teste são adequadas a diferentes sistemas e são aplicadas conforme os recursos disponíveis à equipe.

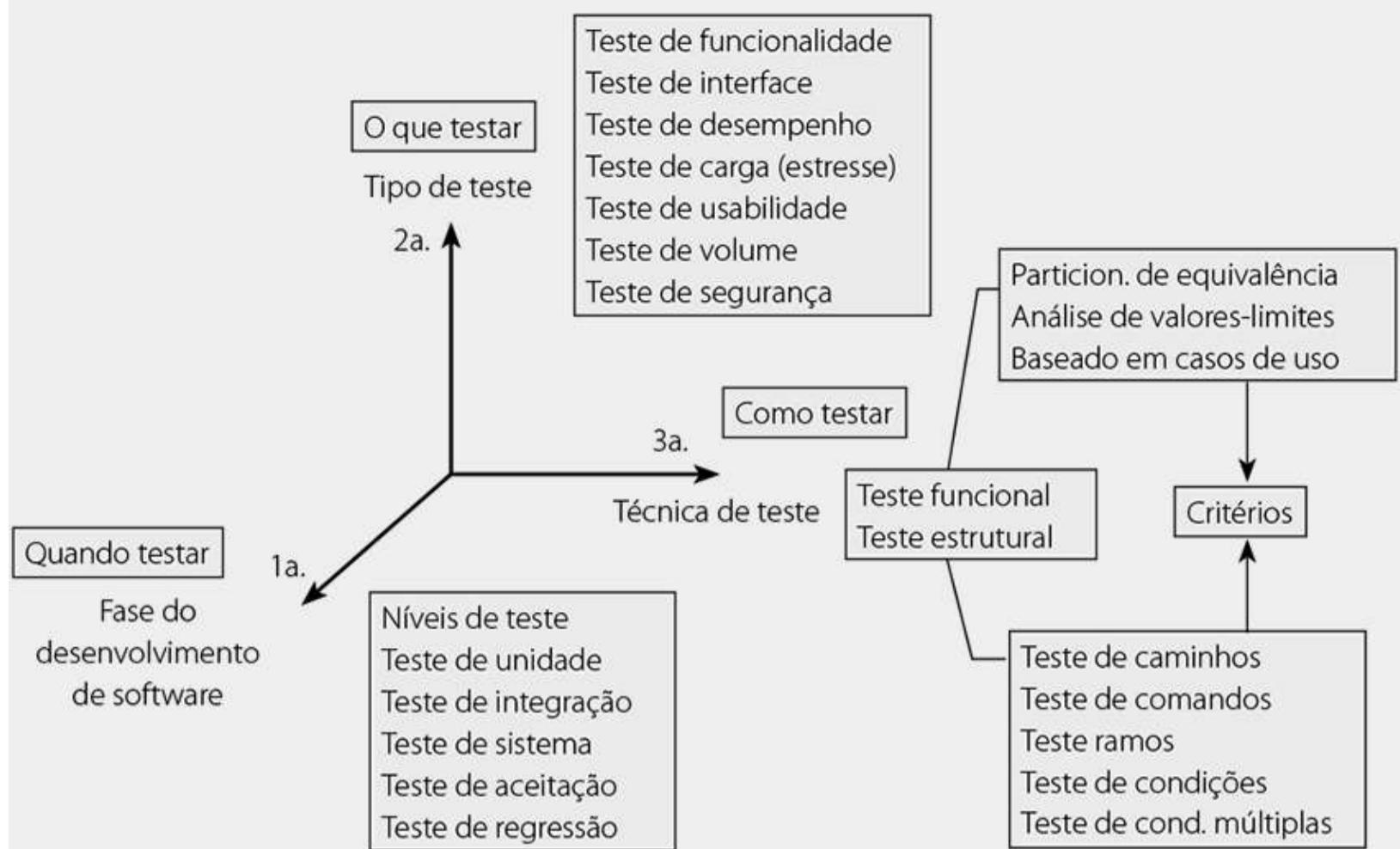
O último item da estratégia de teste embasa nosso próximo assunto e, para entendermos as razões de uma técnica, vale uma breve digressão: os testes são realizados tendo em vista objetivos específicos e são projetados para verificar diferentes propriedades de um sistema. A depender da técnica escolhida, os casos de teste podem ser planejados, por exemplo, para verificar se as especificações funcionais estão implementadas corretamente. Em outros casos, eles são selecionados para averiguação da adequação do desempenho, da confiabilidade e a da usabilidade, por exemplo. Assim, abordagens diferentes de teste motivam a aplicação de técnicas também diferentes de testes (PRESSMAN; MAXIM, 2016).

TIPOS DE TESTES

Embora as técnicas de teste sejam de grande relevância, por remeterem mais diretamente a metodologias de aplicação de testes, esta é não a única dimensão que podemos identificar nesse contexto. Na verdade, podemos estabelecer outros tipos de relações entre um teste e seu objetivo e, para isso, basta fixarmos o momento, o objeto e, como já mencionado, a maneira (ou metodologia) da aplicação.

Parece complicado? Observe a Figura 3.7: ela posiciona os vários tipos de teste nas três dimensões que acabamos de mencionar: o “quando”, o “que” e o “como”.

Figura 3.7 | As três dimensões dos testes



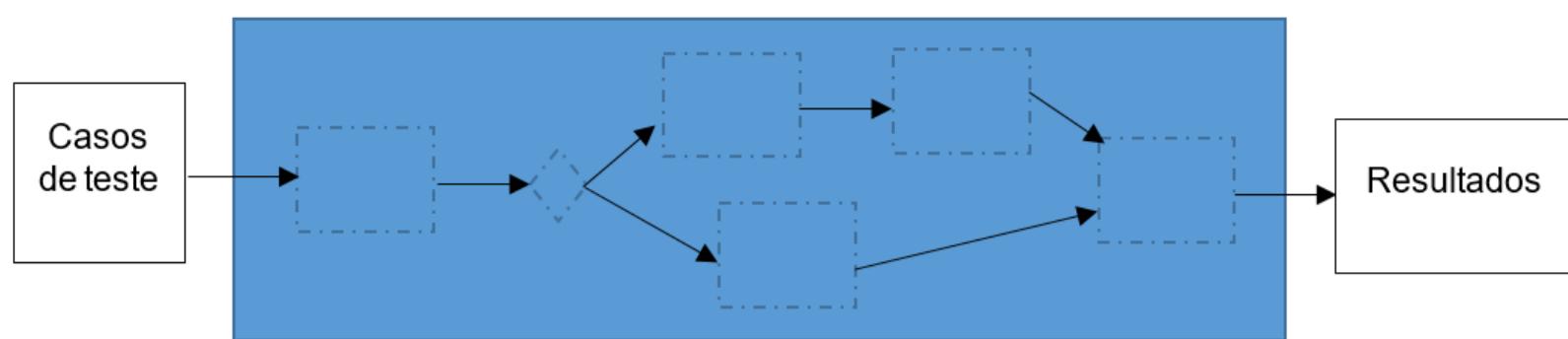
Fonte: Braga (2016, p. 24).

Apesar de não termos tido contato com os testes apontados na figura, é possível destacar que o teste funcional está relacionado a uma maneira de se realizar um teste, daí ter sido posicionado na dimensão “Como testar”. Já o teste de funcionalidade – que também será abordado na sequência – guarda associação com “o que testar”. Feita essa contextualização, passamos à abordagem inicial da técnica de teste funcional e do teste de funcionalidade.

TÉCNICA DE TESTE FUNCIONAL E TESTE DE FUNCIONALIDADE

Essa técnica baseia-se nas especificações do software para derivar os requisitos de teste. O teste é realizado nas funções do programa, daí o nome **funcional**. Não é seu objetivo verificar como ocorrem internamente os processamentos, mas se o algoritmo inserido produz os resultados esperados (BARTIÉ, 2002).

Uma das vantagens dessa estratégia de teste é o fato de ela não requerer conhecimento sobre detalhes da implementação do programa. Sequer o código-fonte é necessário. Observe uma representação dessa técnica na Figura 3.8:



Fonte: Bartié (2002, p. 105).

A ideia ilustrada na figura nos faz entender que o testador não conhece os detalhes internos do sistema e baseia seu julgamento apenas nos resultados obtidos a cada entrada fornecida. A esse respeito, Pressman e Maxim (2016) afirmam que um produto de software pode ser testado caso o responsável conheça a função específica para a qual aquele software foi projetado e, com esse conhecimento, estará em condições de realizar testes que demonstrem que cada uma das funções é operacional, embora o objetivo final do teste seja o de encontrar defeitos no produto. Os autores concluem que essa abordagem de teste se vale de uma visão externa do produto e não se preocupa com a lógica interna do software, a qual é opaca ao testador. Por esse motivo, a técnica funcional também é conhecida como teste de caixa preta.

O planejamento do teste funcional envolve dois passos principais: **identificação das funções** que o software deve realizar (por meio da especificação dos requisitos) e a **criação de casos de teste** capazes de checar se essas funções estão sendo executadas corretamente. Apesar da simplicidade da técnica e apesar de sua aplicação ser possível em todos os programas cujas funções são conhecidas, não podemos deixar de considerar uma dificuldade inerente: não se pode garantir que partes essenciais ou críticas do software serão executadas, mesmo com um bom conjunto de casos de teste.

Um teste funcional não deve ser aplicado simultaneamente em todas as funções do sistema e nem em apenas uma única ocasião. Em vez disso, ele deve examinar elementos específicos em ocasiões previamente conhecidas, característica que levou esse tipo de teste a ser dividido em

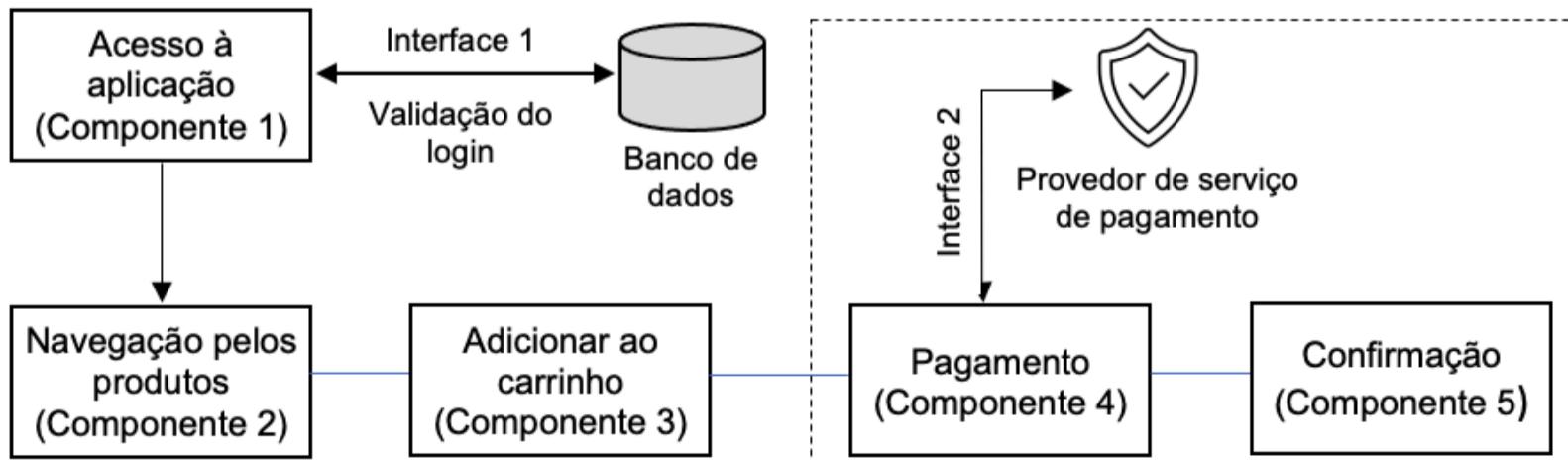
subtipos. Alguns exemplos ajudarão a esclarecer essa circunstância: quando aplicado para verificar funções de um módulo, função ou classe do sistema, ele recebe a denominação de **teste de unidade**. Já o **teste de integração** é executado quando se deseja avaliar como as unidades funcionam em conjunto ou integradas e o **teste de regressão** é um subtipo do teste funcional, que visa garantir que uma alteração feita em uma função não tenha introduzido outros problemas no código (PRESSMAN; MAXIM, 2016).

A menção ao quarto subtipo de teste funcional nos dará a oportunidade para o desenvolvimento de um exemplo prático. Quando testamos um componente do sistema de modo independente para verificar sua saída esperada, chamamos tal procedimento de **teste de componentes**.

Geralmente, ele é executado para verificar a funcionalidade e/ou usabilidade de componentes, mas não se restringe apenas a eles. Um componente pode ser qualquer coisa que receba entradas e forneça alguma saída, incluindo uma página web, telas e até mesmo um sistema dentro de um sistema maior.

Como aplicação prática desse teste, imaginemos um sistema de vendas pela internet. Em uma análise mais minuciosa, seria possível identificar muitos componentes nessa aplicação, mas escolheremos apenas alguns, os quais estão representados na Figura 3.9.

Figura 3.9 | Componentes de um sistema de vendas pela internet



Fonte: elaborada pelo autor.

O componente de login (identificado como componente 1) efetiva a validação de acesso do usuário ao sistema, usando, para isso, uma interface específica, que realiza a conexão com o banco de dados. Esse componente dá acesso à navegação pelos produtos, à inclusão de itens no carrinho, ao pagamento da compra e à sua confirmação. Para efetivação do pagamento, há uma interface que conecta o sistema em teste a um provedor de serviço de pagamentos. Considerando esse cenário, vejamos o que deve ser testado em específico no componente 1 (login):

- A interface de usuário nos itens de usabilidade e acessibilidade.
- O carregamento da página, como forma de garantir bom desempenho da aplicação.
- A suscetibilidade a ataques ao banco de dados por meio de elementos da interface de usuário.
- A resposta da funcionalidade de login por meio do uso de credenciais falsas e válidas.

Se considerarmos o componente 3 (adição do produto ao carrinho), o testador deverá verificar o que ocorre, por exemplo, quando o usuário coloca um item, o qual supostamente deseja comprar, e fecha aplicação em seguida. Um cuidado especial também deverá ser tomado com a comunicação entre o componente 4 e o provedor de serviço de pagamento.

Pelas suas características, o teste de componente nos prepara para o teste que vem a seguir. Se a técnica de teste funcional se preocupa com as funções do sistema, qual seria, então, sua diferença para o teste de funcionalidade? Os **testes de funcionalidades** priorizam interações com o usuário e a navegação no sistema e são executados para verificar se um aplicativo de software funciona corretamente, de acordo com as especificações do projeto, no que se refere à entrada de dados, validação de dados, funções de menu e tudo o que está relacionado à interação do usuário com as interfaces. Além das verificações

mencionadas, um teste de funcionalidades deve se preocupar também em averiguar se funções de “copia e cola” funcionam corretamente e se os ajustes de padrões regionais estão de acordo com a localidade em que o software está sendo executado.

EXEMPLIFICANDO

Um sistema pode ter sido testado em todas as suas funcionalidades em um determinado país e, ao ser instalado e executado em outro, apresentar problemas nas mesmas funcionalidades já testadas. Isso ocorre devido às diferenças de formatos nos sistemas métricos e de data implementados em diferentes regiões do planeta. Um campo que exige a inclusão de uma data, por exemplo, será testado nos Estados Unidos considerando o formato “mm/dd/aaaa”. Na eventualidade de ser executado no Brasil, o programa deverá receber, no campo mencionado, uma data no formato “dd/mm/aaaa”, o que certamente causará inconsistência no processamento da informação recebida.

No escopo da qualidade de software, não há apenas uma acepção relacionada à funcionalidade. Existem as funcionalidades do sistema, que são objetos de teste, e a funcionalidade entendida como um atributo fundamental da qualidade. Neste segundo caso, trata-se do grau com o que o software satisfaz as necessidades declaradas pelo cliente, conforme indicado pelos subatributos de adequabilidade, exatidão, interoperabilidade, conformidade e segurança (PRESSMAN; MAXIM, 2016). Em outras palavras, a funcionalidade oferecida pelo sistema não pode ser confundida com o grau com que um programa atende a requisitos de adequação ao seu propósito, à sua exatidão e à facilidade com que opera com outros sistemas.

Se voltarmos um pouco no texto e observarmos novamente a Figura 3.7, encontraremos o **teste estrutural** posicionado na mesma dimensão do teste funcional. Essa dimensão, que chamamos de “Como testar”, agrupam técnicas (ou maneiras) de se realizar testes. Os testes estruturais (também chamados de caixa branca) são assim conhecidos por serem baseados na arquitetura interna do programa. Contando com o código-fonte e com a estrutura do banco de dados, o testador poderá submeter o programa a uma ferramenta automatizada de teste. Vale aqui a menção de que um teste funcional também pode (e deve) ser realizado de forma automática.

Há várias ferramentas capazes de realizar testes estruturais, mas não usaremos nenhuma em especial para o nosso propósito de detalhar esse teste. Em vez disso, nós nos apoiaremos em uma possível forma de representação do código do programa para construir um método de teste. Vamos considerar que, ao analisar o código do programa, nossa ferramenta construa uma representação dele, conhecida como **grafo**. De acordo com Delamaro (2004), em um grafo, os nós equivalem a blocos indivisíveis de código, o que, em outras palavras, significa que não existe desvio de fluxo do programa para o meio do bloco e, uma vez que o primeiro comando dele é executado, os demais comandos são executados sequencialmente. Outro elemento que integra um grafo são as arestas (ou arcos), e eles representam o fluxo existente entre os nós. Se considerarmos o código da aplicação que calcula o fatorial de um valor, exibido no Código 3.3, teremos que o trecho entre a linha 1 e a linha 8 obedece aos requisitos para se transformar em um nó do grafo.

Código 3.3 | Programa que calcula o fatorial de um número fornecido

```

1 #include <stdio.h>
2
3 main()
4 {
5     int i = 0;
6     valor = 0;
7     fatorial = 1;
8
9     printf("Programa que calcula o fatorial de um valor
10 informado pelo usuario\n");
11
12     do {
13         printf("\nInforme um valor entre 1 e 10: ");
14         scanf("%d",&valor);
15         } while ((valor<1) || (valor>10));
16         for (i=1; i<=valor; i++)
17             fatorial=fatorial*i;
18         printf("\nO Fatorial de %d = %d", valor, fatorial);
19         printf("\n");
20     }

```

0

Ver anotações

Fonte: elaborado pelo autor.

O programa que usaremos para ilustrar um teste estrutural será aquele cujo código pode ser visto no Código 3.4. Sua função é a de verificar a validade de um nome de identificador fornecido com base nas seguintes regras:

- Tamanho t do identificador entre 1 e 6 ($1 \leq t \leq 6$): condição válida.
- Tamanho t do identificador maior que 6 ($t > 6$): condição inválida.
- Primeiro caractere c é uma letra: condição válida.
- Primeiro caractere c não é uma letra: condição inválida.
- O identificador possui apenas caracteres válidos: condição válida.
- O identificador possui um ou mais caracteres inválidos: condição inválida.

```

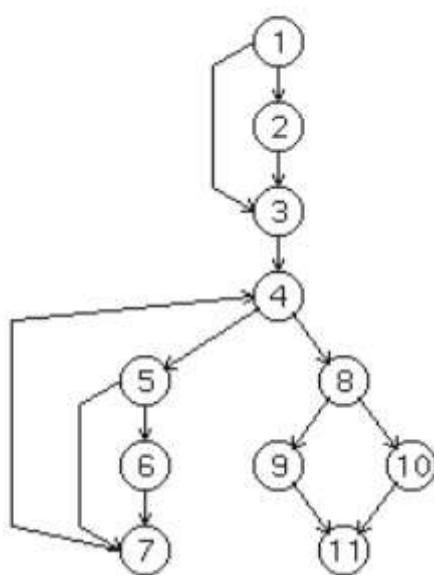
/* 01 */ {
/* 01 */ char achar;
/* 01 */ int length, valid_id;
/* 01 */ lenght = 0;
/* 01 */ printf ("Identificador");
/* 01 */ achar = fgetc(stdin);
/* 01 */ valid_id = valid_s(achar);
/* 01 */ if (valid_id)
/* 02 */     lenght = 1;
/* 03 */     achar = fgetc (stdin);
/* 04 */     while (achar != '\n')
/* 05 */     {
/* 05 */         if (!(valid_f(achar)))
/* 06 */             valid_id = 0;
/* 07 */         lenght++;
/* 07 */         achar = fgetc (stdin);
/* 07 */     }
/* 08 */     if (valid_id && (length >= 1) && (lenght < 6))
/* 09 */         printf ("Valido\n");
/* 10 */     else
/* 10 */         printf ("Invalido\n");
/* 11 */ }

```

Fonte: Delamaro (2004, p. 20).

Quando analisado pela ferramenta de teste, o Código 3.4 será transformado no grafo da Figura 3.10. Note que cada trecho identificado com um número no código é representado em um nó no grafo gerado.

Figura 3.10 | Grafo gerado pela ferramenta de teste estrutural



Fonte: Delamaro (2004, p. 21).

Ao testador cabe selecionar casos de teste capazes de percorrer os nós, os arcos e os caminhos representados no grafo do programa. Apenas para fins de exemplificação, um caminho que pode ser percorrido pelo fluxo do programa é o formado pela sequência de nós 2,3,4,5,6 e 7. Casos de testes diferentes tenderão a exercitar sequências diferentes do grafo e, se considerarmos a existência de um conjunto infinito desses casos, teremos que as escolhas incorretas podem arruinar o teste. Ainda, se considerarmos que os casos de teste são potencialmente infinitos, alguns critérios de cobertura do código devem ser previamente definidos, o que nos leva a uma reflexão.

REFLITA

Se é certo que um teste, por melhor que seja sua execução e por melhor que tenham sido as escolhas para os casos de teste, não conseguirá assegurar que o programa é totalmente livre de defeitos, qual seria a indicação de que é chegado o momento de interrompê-lo? A resposta passa pelos critérios de parada da atividade, os quais foram estabelecidos previamente e que definiam taxa de cobertura de código e quantidade de casos de teste, por exemplo.

Feitas essas abordagens, avançamos rumo aos testes de aplicações móveis e web.

Houve um tempo em que códigos executáveis desempenhavam suas funções apenas em computadores de grande porte ou nos poucos computadores pessoais que existiam. Como os dispositivos móveis eram apenas um sonho, o planejamento e a execução dos testes eram atividades adequadas ao perfil dos programas da época e aos equipamentos que os executavam. Em nossos dias, no entanto, há dispositivos com uma ampla variedade de aplicações capazes de executar programas para as mais variadas finalidades e essa realidade exige algumas adequações nos procedimentos de testes aplicados nas plataformas mais comuns.

A fim de delimitar nosso estudo, trataremos aqui dos testes voltados a aplicações feitas para dispositivos móveis e dos testes executados em aplicações para a internet, nessa mesma ordem. Pressman e Maxim (2016) destacam o que consideram abordagens de teste especializadas para aplicativos móveis:

- **Teste de experiência do usuário:** ninguém melhor do que um futuro usuário da aplicação móvel para expressar suas expectativas de usabilidade e acessibilidade, por exemplo. Se considerarmos uma aplicação móvel de vendas, os vendedores deverão ser incluídos no processo de desenvolvimento desde o seu início, para que o nível de experiência desejado por eles seja atingido.
- **Teste de compatibilidade do dispositivo:** este item sugere que os testadores devem verificar se o aplicativo móvel funciona corretamente com o hardware escolhido para executar o aplicativo.
- **Teste de desempenho:** neste quesito, os testadores devem verificar se indicadores de desempenho exclusivos dos dispositivos móveis correspondem à necessidade do cliente. Esses indicadores incluem tempo de download e autonomia de carga, por exemplo.
- **Teste de conectividade:** aqui são testados a capacidade da aplicação em realizar conexões em redes de diferentes modalidades

(*ad hoc* ou com infraestrutura, por exemplo) e o comportamento da aplicação em caso de queda da conexão.

Essas considerações nos permitem criar um panorama dos testes de aplicações móveis, começando com sua definição: são atividades organizadas de teste, executadas com o objetivo de descobrir erros em aspectos fundamentais de seu funcionamento em um dispositivo móvel. Esses testes são realizados pelos profissionais técnicos, além dos usuários, do cliente e do gerente do projeto. Um teste típico começa por verificar as funções visíveis da aplicação para, então, voltar sua atenção para aspectos de infraestrutura e tecnologia. Normalmente, as etapas desse teste incluem testes de conteúdo, de interface, de navegação, de componente, de configuração, de desempenho e de segurança.

ASSIMILE

O teste de aplicativos móveis é um conjunto de atividades relacionadas com um único objetivo: descobrir erros no conteúdo, na função, na usabilidade, na naveabilidade, no desempenho, na capacidade e na segurança do aplicativo móvel. Para tanto, deve ser executada uma estratégia de teste que abrange as revisões e o teste executável (PRESSMAN; MAXIM, 2016).

Como qualquer teste, temos aqui também a necessidade de apurar se o processo foi corretamente desempenhado. Como não é possível aplicar absolutamente todas as possibilidades existentes de entradas, nem apurar todas as situações que podem ocorrer durante o uso da aplicação, novamente os critérios de parada serão o parâmetro para o fim dos testes.

Outro elemento que demanda providências específicas durante o processo de teste são as aplicações web. Da mesma forma que em qualquer outro teste, aqui também estamos diante da missão de executar um sistema para encontrar e corrigir defeitos. No entanto, as

aplicações web podem operar em conjunto com diferentes sistemas operacionais, navegadores, protocolos de comunicação e plataformas de hardware, o que pode representar dificuldades adicionais na busca por defeitos. Com essas especificidades, os seguintes elementos de qualidade devem ser testados (PRESSMAN; MAXIM, 2016):

- **Conteúdo da aplicação:** deve ser avaliado em dois níveis:
 - Sintático: neste nível devem ser avaliadas a ortografia, a pontuação e a gramática do conteúdo textual da aplicação web.
 - Semântico: aqui devem ser avaliadas a exatidão, a consistência e a ausência de ambiguidade da informação.
- **Funcionalidades da aplicação:** são testadas para fins de descoberta de problemas que colocam a ferramenta em situação de não conformidade com os requisitos do cliente.
- **Estrutura da aplicação:** a facilidade com que a estrutura da aplicação pode crescer e, ainda assim, passar por manutenções é testada neste elemento.
- **Usabilidade:** testes são feitos para que vários perfis de usuários possam se adaptar às características da interface da aplicação.
- **Navegabilidade:** atenção especial é dedicada a links inativos e incorretos neste elemento.

Adicionalmente, desempenho, interoperabilidade e segurança também recebem atenção especial em um teste de aplicação web.

| TESTES DE APLICAÇÕES ORIENTADAS A OBJETOS

Seguindo nosso caminho pelos testes feitos em aplicações ou paradigmas específicos, chegamos até as aplicações Orientadas a Objetos (OO). Por causa das características próprias dos programas OO, os testes aqui aplicados devem considerar a existência de subsistemas em camadas que encapsulam outras classes. De acordo com Pressman e Maxim (2016), é preciso testar um sistema OO em vários níveis

diferentes, de modo que erros eventualmente ocorridos durante as interações entre as classes sejam descobertos. Ainda de acordo com os autores, três providências básicas são necessárias para se testar adequadamente um sistema OO:

- A definição do teste deve ser ampliada para incluir as técnicas da descoberta de erros aplicadas à análise orientada a objetos e aos modelos do projeto.
- A estratégia para teste de unidade e de integração deve ser alterada significativamente.
- O conjunto de casos de teste devem levar em conta as características especiais do paradigma de Orientação a Objetos.

Uma análise mais minuciosa da primeira providência pode nos levar a uma aparente inconsistência: como é possível que aspectos da análise orientada a objetos e modelos do projeto sejam testados? De fato, no sentido convencional, não podem. Ocorre que a criação de uma aplicação OO começa com a criação de modelos de análise e de projeto, os quais começam com representações quase informais dos requisitos e se tornam modelos detalhados de classes e de suas relações conforme o desenvolvimento avança. Por isso, em cada estágio da sua evolução, esses modelos devem ser revisados para que erros sejam descobertos logo em fases iniciais do desenvolvimento (PRESSMAN; MAXIM, 2016).

Entre os vários benefícios trazidos pela adoção da Orientação a Objetos para o desenvolvimento de software, está o fato de que esse paradigma reduz a necessidade de testes, segundo Schach (2009). O autor complementa que a reutilização de código via herança é um dos fatores que torna essa característica ainda mais forte: em tese, uma vez que a classe mãe tenha sido testada, as classes provenientes dela não precisam ser novamente testadas. Quando os desenvolvedores inserem novos métodos na subclasse de uma classe já testada, é natural que eles precisem ser testados. No entanto, métodos herdados não precisam de teste adicional.

Por mais lógicas e confiáveis que nos pareçam, essas afirmações precisam ser analisadas com um certo cuidado. Para começar, a classe é um tipo de dado abstrato e o objeto é a instância de uma classe. Esse fato nos induz ao raciocínio de que uma classe não tem uma realização concreta e que, portanto, não é possível aplicar testes baseados em execução (ou seja, aqueles que temos abordado nesta unidade) diretamente nela. Outra característica da Orientação a Objetos que tem efeitos em um teste é quantidade normalmente elevada de métodos em um objeto que, ao invés de retornarem um valor para o chamador, simplesmente alteram o estado do objeto ao modificarem seus atributos. Segundo Schach (2009), a dificuldade, nesse caso, é a de testar se a alteração de estado foi realizada corretamente.

Tomemos a seguinte situação como exemplo: uma aplicação bancária possui um método chamado *deposito*, cujo efeito é o de aumentar o valor da variável de estado chamada *saldoDaConta*. No entanto, como consequência do ocultamento de informações, a única maneira viável de se testar se o método *deposito* tem uma execução correta é pela invocação de um outro método, chamado *determinarSaldo*, tanto antes como depois da chamada do método *deposito*, a fim de verificar como a mudança do saldo se dá.

Outro aspecto a ser analisado é a necessidade de aplicação de um novo teste em métodos herdados. Observe a hierarquia de classes ilustrada no trecho de código contido no Código 3.5. Na primeira classe (*ClasseArvoreComRaiz*), são definidos dois métodos: o que exibe o conteúdo de um nó e a rotina de impressão, utilizada pelo método *exibeConteudoNo*.

Código 3.5 | Ilustração de uma hierarquia de classes em Java

```

1  class ClasseArvoreComRaiz {
2      void exibeConteudoNo (No a);
3      void rotinaImpressao (No b);
4      // o método exibeConteudoNo usa o método
5      rotinaImpressao
6  }
7
8  class ClasseArvoreBinaria extends ClasseArvoreComRaiz {
9      void exibeConteudoNo (No a);
10     // o método exibeConteudoNo definido aqui usa
11     // o método rotinaImpressao herdado de
12     ClasseArvoreComRaiz
13 }
14
15 class ClasseArvoreBinariaBalanceada extends
16 ClasseArvoreBinaria {
17     void rotinaImpressao (No b);
18     // o método exibeConteudoNo (herdado de
19     ClasseArvoreBinaria)
20     // usa essa versão local de rotinaImpressao dentro da
21     classe
22     // ClasseArvoreBinariaBalanceada
23 }

```

6

Ver anotações

Fonte: adaptado de Schach (2009).

Observe que a subclasse `ClasseArvoreBinaria` herda o método `rotinaImpressao` da classe mãe chamada `ClasseArvoreComRaiz`. Além disso, o método `exibeConteudoNo` é novamente definido nessa subclasse, o que anula o mesmo método definido na classe `ClasseArvoreBinaria`. Já a subclasse `ClasseArvoreBinariaBalanceada` herda o método `exibeConteudoNo` da superclasse `ClasseArvoreBinaria`. No entanto, nela é definido um novo método, o `rotinaImpressao`, que anula aquele definido em `ClasseArvoreComRaiz`. Quando o método

`exibeConteudoNo` usa o método `rotinaImpressao` no contexto de `ClasseArvoreBinariaBalanceada`, as regras do Java especificam que a versão local de `rotinaImpressao` deve ser usada e, em consequência disso, o código real do método `rotinaImpressao`, executado quando `exibeConteudoNo` é chamado no contexto da instanciação da classe `ClasseArvoreBinaria`, é diferente daquele executado quando esse mesmo método é executado na instanciação de `ClasseArvoreBinariaBalanceada`. Esse fenômeno ocorre normalmente apesar de o método `exibeConteudoNo` ser herdado sem modificação, o que acarreta a necessidade do novo teste (SCHACH, 2009).

Também considerando características próprias de sistemas Orientados a Objetos, Masiero *et al.* (2015) dividem o procedimento de testes em três fases:

- **Teste de unidade:** os autores consideram que as menores unidades a serem testadas em um programa OO são os métodos. Por isso, indicam que o teste de unidade consiste no teste de cada método de forma isolada, o que também é chamado de teste intra-método.
- **Teste de módulo:** trata-se do teste de um conjunto de unidades que interagem entre si por meio de chamadas. Essa fase pode ainda ser assim dividida:
 - Inter-método: aplicação de teste nos métodos públicos de uma classe, em conjunto com os outros métodos da mesma classe.
 - Intra-classe: consiste em testar as interações entre os métodos públicos de uma classe quando chamados em diferentes sequências.
 - Inter-classe: consiste em testar as interações entre classes diferentes.
- **Teste de sistema:** consiste em testar a integração entre todos os módulos, o que equivale a um sistema completo. Para esta fase geralmente é usado o teste funcional.

Este foi, portanto, o conteúdo preparado para esta seção. Durante seu desenvolvimento, você teve contato com as técnicas de teste funcional e estrutural e com outros tipos de testes específicos para determinadas aplicações e paradigmas de desenvolvimento. Tivemos a oportunidade de desenvolvermos juntos um exemplo de teste estrutural baseado na criação de um grafo, que representava o código apresentado. Embora simples, o exemplo nos mostrou como a técnica funciona e quanto importante é a correta escolha dos casos de teste. Não deixe de se aprofundar neste assunto e torne-o logo uma boa referência em testes de software. Até a próxima!

FAÇA VALER A PENA

Questão 1

Ao executar o teste da caixa preta, o testador não levará em consideração o comportamento interno do sistema. Essa técnica leva esse nome por considerar o processamento do sistema de forma desconhecida e, por isso, apenas as entradas e as saídas do sistema serão avaliadas.

Um teste de caixa branca não poderá ser executado quando o testador não tiver a posse do:

a. Registro formal do sistema.

b. Grafo do sistema.

c. Código-fonte do sistema.

d. Conjunto de funcionalidades do sistema.

e. Arestas e caminhos do sistema.

Questão 2

Se os usuários encontrarem erros ou dificuldades dentro de um aplicativo móvel, vão procurar o conteúdo e a função personalizados de que precisam em outro lugar. Por isso, você deve procurar e corrigir o

máximo de erros possível antes que o aplicativo móvel seja colocado em uma loja ou repositório de aplicativos.

Considerando os testes de aplicações móveis, analise as afirmações que seguem:

- I. O procedimento geral de teste em uma aplicação móvel é totalmente diferente do procedimento aplicado em sistemas convencionais.
- II. Os objetivos do teste de aplicações móveis devem dar relevância maior às verificações de desempenho do que de interface, dada a natureza dessas aplicações.
- III. Além do corpo técnico de testadores, um procedimento de teste de aplicações móveis deve contar com a presença também dos usuários da aplicação.

É verdadeiro o que se afirma em:

a. I apenas.

b. III apenas.

c. II apenas.

d. I e III apenas.

e. II e III apenas.

Questão 3

Uma questão clássica surge todas as vezes que se discute teste de software: “Quando podemos dizer que terminamos os testes – como podemos saber que já testamos o suficiente?” Há algumas respostas pragmáticas e algumas tentativas empíricas para essa questão.

Considerando o conceito de teste e as características de seu procedimento, assinale a alternativa que contém a sentença indicativa do momento em que o testador deve interromper um procedimento de teste.

- a. Quando ele tiver certeza de que o programa está completamente livre de defeitos.
- b. Quando o grafo do programa for totalmente coberto pelos casos de teste.
- c. Quando todos os casos de teste existentes tiverem sido usados no procedimento.
- d. Quando os critérios estabelecidos para o término tiverem sido atingidos.
- e. Quando os critérios de total ausência de defeitos tiverem sido todos cumpridos.

REFERÊNCIAS

BARTIÉ, A. **Garantia da Qualidade de Software**: As melhores práticas de Engenharia de Software aplicadas à sua empresa. Rio de Janeiro: Elsevier, 2002.

BRAGA, P. H. **Testes de Software**. São Paulo: Pearson Educational do Brasil, 2016.

COMO são feitos os testes de usuário? [S.I.: s.n.], 2018. 1 vídeo (4 min). Canal Alura Cursos Online. Disponível em: <https://bit.ly/3sYiSID>. Acesso em: 24 dez. 2020.

DELAMARO, M. E. **Introdução ao teste de software**. Rio de Janeiro: Elsevier, 2004.

MASIERO, P. C., LEMOS, O. A. L, FERRARI, F. C., MALDONADO, J. C. Teste de software orientado a objetos: teoria e prática. **ResearchGate**, [S.I.], 2015. Disponível em: <https://bit.ly/3iRA1ZR>. Acesso em: 24 dez. 2020.

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

SCHACH, S. R. **Engenharia de Software**: os Paradigmas Clássico e Orientado a Objetos. 7. ed. São Paulo: McGraw-Hill, 2009.

TESTES de Integração e Teste de Sistema. [S.l.: s.n.], 2018. 1 vídeo (12 min). Canal Eduardo Engelharia de Software. Disponível em: <https://bit.ly/39knSsX>. Acesso em: 24 dez. 2020.

FOCO NO MERCADO DE TRABALHO

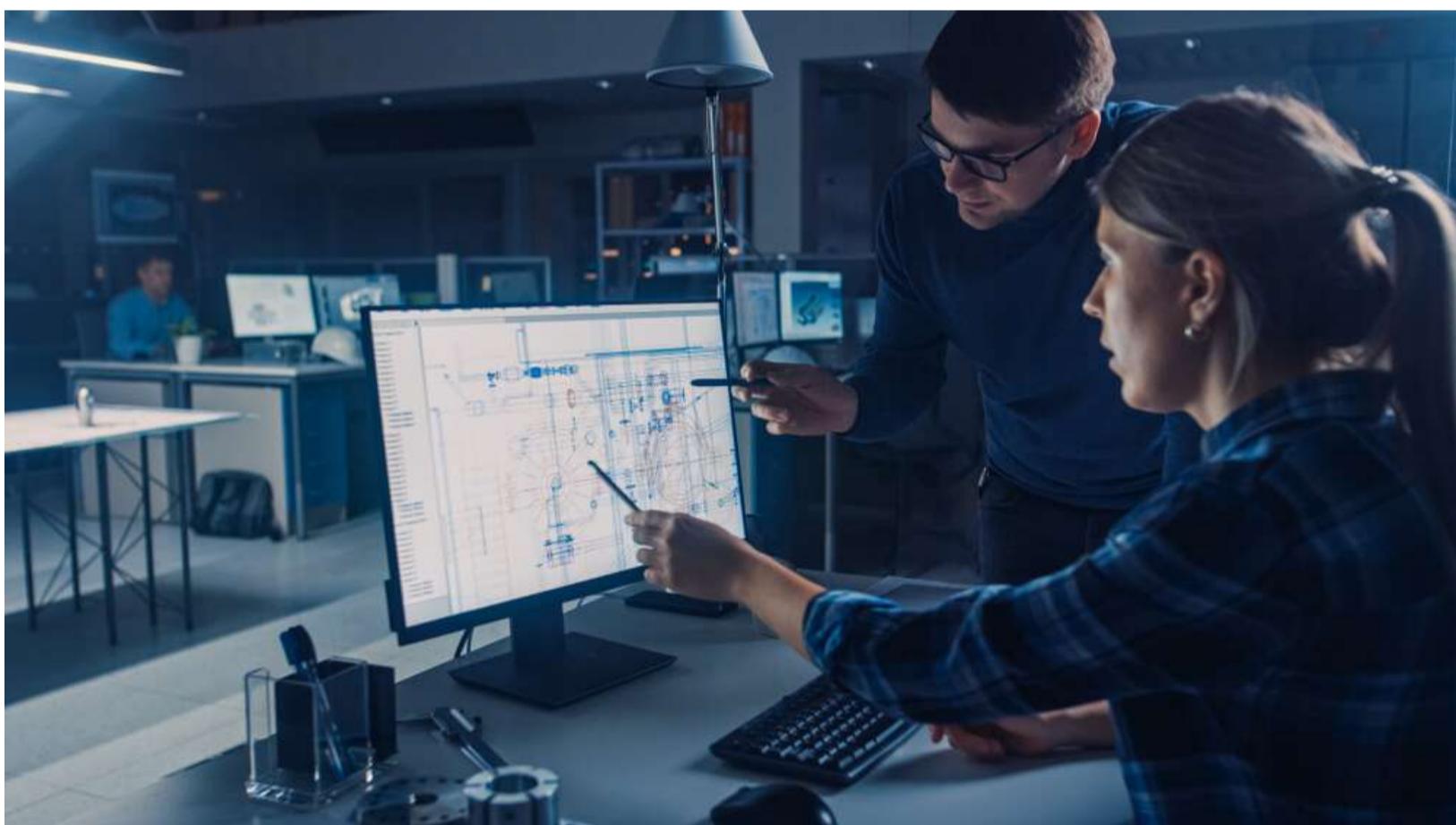
TIPOS DE TESTE

Roque Maitino Neto

Ver anotações

PLANEJAMENTO DO PROCEDIMENTO DE TESTE

O planejamento de teste deve ser composto por um documento que contenha os requisitos do sismtea, as funções a serem testadas descritas, a técnica de teste que será utilizada, o critério utilizado para a escolha dos casos de testes, e a definição de quem participará dos testes.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

SEM MEDO DE ERRAR

Um procedimento de teste que não é precedido por um planejamento criterioso pode perder o rumo ao longo do caminho e não ter uma conclusão satisfatória. Nesse sentido, esta situação-problema procura

conduzi-lo à criação de um planejamento coerente com as possibilidades e restrições impostas pelo cenário. Apenas para resgatarmos o contexto em que a narrativa se dá, você foi designado para testar um sistema convencional (ou seja, não web e não móvel) e, de antemão, sabe que não contará com o código-fonte dele.

Como um documento de planejamento sempre refletirá características, estilo e outras subjetividades de quem o criou, o que forneceremos aqui serão as linhas gerais que nele deverão estar presentes. A primeira providência será a de prever o acesso ao documento em que os requisitos do sistema estão especificados. Será por meio dele que você conhecerá as funções do sistema e poderá criar casos de testes específicos para cada uma. Lembre-se: não há código-fonte disponível e, portanto, a técnica de testes mais imediata para aplicação é a funcional.

O efeito da ausência do código-fonte responde ao segundo item do planejamento. Como já mencionado, a técnica a ser das funções derivadas dos requisitos, o testador poderá aplicar testes em cada uma delas. Esta circunstância utilizada é a funcional e, por meio nos apresenta então outro desdobramento: as características dos casos de teste. Eles deverão ser selecionados de modo que consigam reproduzir o uso corriqueiro das funções, além de serem capazes de verificar itens de usabilidade das interfaces de usuário.

Por fim, o procedimento de teste deve ser acompanhado por ao menos um desenvolvedor do software. Como esse profissional provavelmente atua na organização desenvolvedora, o planejamento deverá prever ocasiões em que ele deverá se deslocar ao local em que os testes serão feitos. A presença do desenvolvedor garantirá que um conhecedor dos detalhes do sistema atuará durante os procedimentos de teste.

O que segue é uma solução viável para este desafio:

Documento de planejamento inicial de teste

1. Objetivo

Este documento tem o objetivo de descrever quatro elementos do

planejamento de um teste, considerando o cenário, que será resgatado em seu corpo.

2. Documento de requisitos de software

A fim de que a equipe conheça as funções do produto a ser testado, deverá ser solicitado ao desenvolvedor (também chamado cliente) o documento de requisitos do software.

Restrição: nenhuma restrição se aplica a este item.

3. Técnica de teste

Considerando a ausência do código-fonte, a técnica a ser utilizada é a funcional. Cada função do produto deve ser conhecida e analisada por meio do documento que agrupa seus requisitos.

4. Casos de teste

Os casos de teste devem ser selecionados de modo que possam exercitar, da forma mais completa possível, todas as funções do produto. Eles devem conseguir reproduzir o uso corriqueiro das funções e verificar as condições de usabilidade das interfaces de usuário.

5. Participantes do teste

Será requisitada a participação de ao menos um desenvolvedor do produto no procedimento de teste, em intervalos regulares ou sempre que for necessária sua intervenção. A presença desse desenvolvedor garantirá que um conhecedor dos detalhes do sistema esteja orientando o procedimento de teste.

NÃO PODE FALTAR

DESENVOLVIMENTO ORIENTADO A TESTES E FERRAMENTAS CASE

Roque Maitino Neto

Ver anotações

O QUE É TDD?

O desenvolvimento orientada a testes (TDD), uma das práticas do XP, é um procedimento que faz com que o desenvolvedor escreva testes automatizados de maneira constante ao longo do processo de desenvolvimento e antes mesmo da implementação do código.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

PRATICAR PARA APRENDER

Prezada aluna, caro aluno, sejam bem-vindos à última etapa desta unidade. Ao longo das seções anteriores a esta, tivemos a oportunidade de estudar os fundamentos da atividade de teste e algumas formas de aplicá-los, tendo como critério a posse do código-fonte e as especificidades da aplicação. Com o desenvolvimento de alguns exemplos, pudemos entender o papel dos casos de teste e o motivo de a atividade de teste, em sentido amplo, ser tão importante para o alcance da desejada qualidade do produto. Há, no entanto, ainda algumas questões a serem respondidas: o perfil metodológico de um teste precisa ser, necessariamente, o de criar o código para então testá-lo? Há ferramentas automatizadas disponíveis para testarmos nossos produtos ou todo o processo deve ser feito manualmente?

Aplicar testes e corrigir defeitos em um produto de software não é uma tarefa simples e a execução desse procedimento não é exatamente algo que cause disputas entre membros da equipe. Por isso, a utilização de meios que facilitem a aplicação do teste e a efetiva inserção do procedimento no centro do desenvolvimento do produto são providências fundamentais para que testar se torne uma etapa natural do procedimento de software e deixe de ser algo que se deseje postergar para sempre. Embora essas providências impliquem mudanças em nosso estilo de programar, certamente trarão benefícios que excederão o custo natural da mudança.

Nesta seção trataremos de quatro temas inovadores e com potencial para despertar mais ainda nosso interesse pelos testes: iniciaremos com a abordagem do Desenvolvimento Orientado a Testes (Test Driven Development ou TDD), procedimento estabelecido por Kent Beck (um dos criadores do Extreme Programming), o qual se baseia em ciclos em que são criados testes antes mesmo da implementação da própria funcionalidade. Na sequência, levantaremos algumas questões a respeito do gerenciamento do teste de unidade, figura central no TDD.

Depois, estudaremos os meios de automatizar testes com a finalidade de torná-los mais confiáveis, mais rápidos e menos propensos ao retrabalho. Por fim, abordaremos as ferramentas que possibilitam a realização de todas essas inovações.

Uma empresa que desenvolve aplicações Java para clientes diversos deparou-se com a necessidade de tornar mais ágeis e menos custosos seus procedimentos de teste. Para atingir tal objetivo, seus gestores resolveram incumbi-lo de implementar o TDD, atualmente a única prática do *Extreme Programming* ainda não seguida na empresa. Em conjunto com a implementação dessa nova tecnologia, você também será responsável por tornar completo o processo de automatização dos testes, executado parcialmente até então.

Entendendo que uma das suas primeiras providências para atingimento desses objetivos deve ser a preparação da equipe de desenvolvedores, você decidiu propor a eles um exercício, como forma de familiarizá-los com o esquema geral de um teste automatizado por completo. Como base para esse exercício, você propôs o código, apresentado no Código 3.6, o qual compõe um sistema de leilão que retorna ao usuário o maior lance dado. Ao receber um lance, o algoritmo percorre a lista buscando o maior valor e o imprime em tela.

Código 3.6 | Código a ser utilizado no exercício proposto à equipe

```

1  public class Avaliador {
2
3      private double maiorDeTodos = Double.NEGATIVE_INFINITY;
4
5      public void avalia(Leilao leilao) {
6
7          for(Lance lance : leilao.getLances()) {
8
9              if(lance.getValor() > maiorDeTodos) {
10                  maiorDeTodos = lance.getValor();
11
12              }
13
14
15      public class TesteDoAvaliador {
16
17          public static void main(String[] args) {
18
19              Usuario joao = new Usuario("Joao");
20              Usuario jose = new Usuario("José");
21              Usuario maria = new Usuario("Maria");
22              Leilao leilao = new Leilao("Playstation 3
23 Novo");
24
25              leilao.propoe(new Lance(joao, 300.0));
26              leilao.propoe(new Lance(jose, 400.0));
27              leilao.propoe(new Lance(maria, 250.0));
28
29          }
30
31
32      }
33
34
35      Avaliador leiloeiro = new Avaliador();
36      leiloeiro.avalia(leilao);
37
38
39      System.out.println(leiloeiro.getMaiorLance());
40
41  }

```

Ver anotações

Fonte: Aniche (2015, [s.p.]).

A atividade proposta aos desenvolvedores consiste em, com base no código dado, descrever uma sequência “cenário – ação – verificação”. Para que você tenha esse exercício resolvido antes de aplicá-lo à equipe,

Bom trabalho!

DICA

Testar não precisa se tornar uma tarefa a ser evitada, tampouco deve ser aquele procedimento executado sem gerenciamento algum e sem o devido suporte provido por ferramentas. Boas escolhas serão capazes de torná-lo parte de uma atividade agradável de ser cumprida e de elevada importância no contexto da qualidade do produto. Falta pouco para que você adquira uma visão bastante abrangente dos testes e possa escolher em quais aspectos quer se especializar. Com base nos exemplos e na situação-problema que temos desenvolvido, você está prestes a atingir o nível técnico adequado para colocar em prática tudo o que sabe, o que certamente vai aumentar suas chances de atuar nesta área.

Bons estudos e boa sorte sempre!

CONCEITO-CHAVE

Tivemos a oportunidade de conhecer técnicas e metodologias de teste que se baseavam em uma sequência bem estabelecida de procedimentos e que eram capazes de acomodar as particularidades de alguns tipos específicos de aplicações. Essa sequência foi assim apresentada:

- Implementar código.
- Selecionar casos de teste.
- Executar o código com os casos de teste selecionados.
- Depurar o código para encontrar defeitos.
- Corrigir defeitos.

- Avaliar resultados.

Considerando o momento da aplicação dos testes, a fase em que ela ocorre é, tradicionalmente, uma das últimas do processo de software, logo após o completo desenvolvimento do produto e antes da entrega ao cliente. Via de regra, qualquer atraso ocorrido durante o projeto impactará no tempo disponível para os testes. Haveria então outro momento mais adequado para aplicá-los? Será que a prática do teste também não teria experimentado uma evolução, impulsionada pelo aprimoramento das metodologias de desenvolvimento? Felizmente a resposta é sim para ambas as questões.

| DESENVOLVIMENTO ORIENTADO A TESTES (TDD)

O advento das metodologias ágeis, especificamente o *Extreme Programming* (XP), ofereceu-nos inovações também no modo como fazemos nossos testes, sendo a mais importante delas o TDD. De acordo com Aniche (2014), a ideia central do TDD é a de fazer com que o desenvolvedor escreva testes automatizados de maneira constante ao longo do processo de desenvolvimento e antes mesmo da implementação do código. Ainda segundo o autor, essa inversão no ciclo compele o desenvolvedor a escrever um código de melhor qualidade, já que a escrita de bons testes de unidade requer o bom uso dos recursos da orientação a objetos.

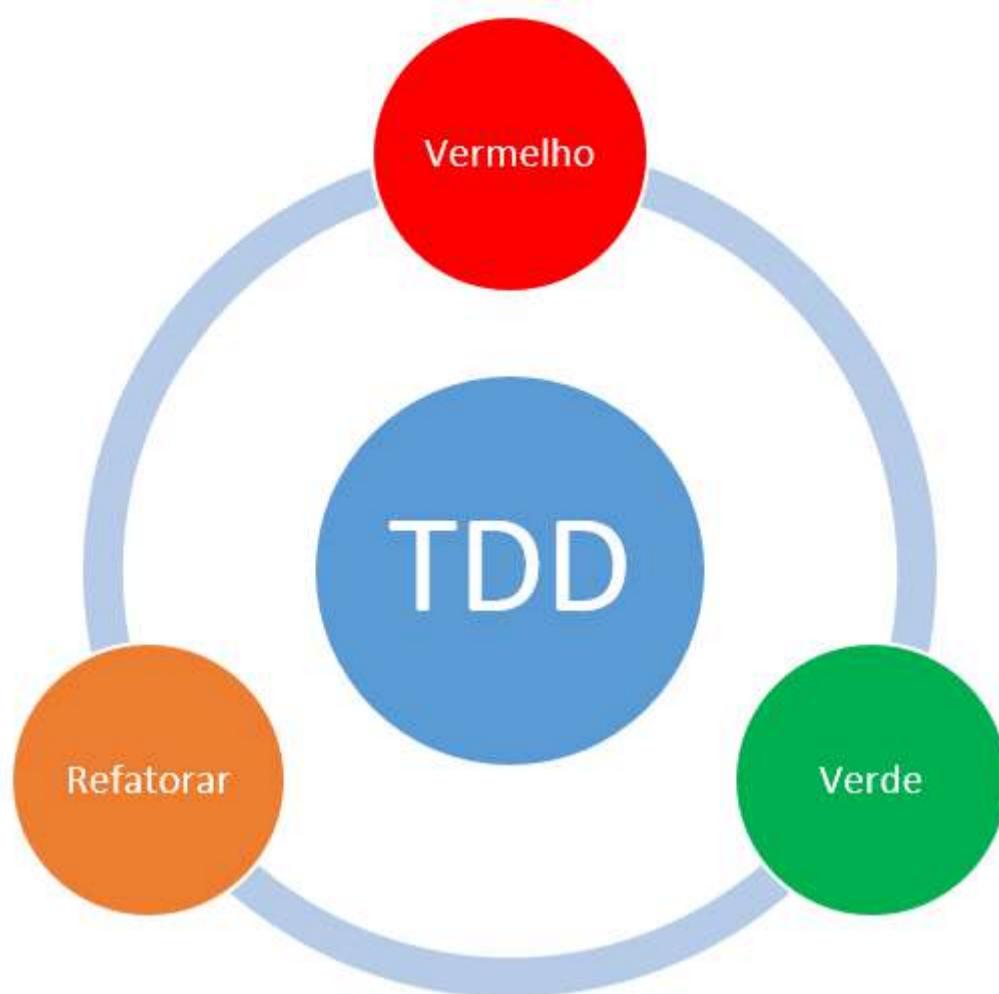
Antes de continuarmos a exploração deste conteúdo, será necessário resgatarmos alguns termos já citados para fazermos uma delimitação do nosso tema. O Desenvolvimento Orientado a Testes será abordado aqui como uma das práticas do XP e, nessa condição, usaremos o teste de unidade como nosso objeto de análise. O fato de o TDD estar situado no contexto do XP explica nossa menção ao paradigma de Orientação a Objetos feita há pouco. Por fim, vale relembrarmos que o teste de unidade é aquele realizado sobre cada classe do sistema e que os testes de aceitação são efetuados sobre cada estória (ou funcionalidade) do sistema (TELES, 2004).

A ilustração de um teste de unidade será oferecida por meio do exemplo de uma loja virtual qualquer na web. Quando o usuário seleciona um produto para comprá-lo, o sistema se incumbe de colocá-lo no carrinho de compras, de fazer contato com a operadora do cartão de crédito, de retirar o produto do estoque, de informar o pessoal da logística para preparar a entrega e de enviar ao usuário um e-mail de confirmação de compra. É natural imaginarmos que o sistema que gerencia todas essas operações (mais aquelas que não mencionamos) seja bastante complexo e que, portanto, esteja implementado em diversas classes, cada qual com uma função específica.

Ao invés de focar o sistema todo, o teste de unidade se preocupará com cada uma dessas classes, que geralmente é a unidade de um sistema Orientado a Objetos. Em nosso sistema de exemplo, muito provavelmente existem classes como CarrinhoDeCompras, Pedido e assim por diante. A ideia é termos baterias de testes de unidade separadas para cada uma dessas classes, com cada bateria preocupada apenas com a sua classe (ANICHE, 2014).

Bem, e como o Desenvolvimento Orientado a Testes funciona? Descobriremos isso ao longo desta primeira etapa da seção e a iniciaremos pela apresentação de uma figura que se tornou referência universal quando se trata desse assunto. A aplicação do Desenvolvimento Orientado a Testes é ilustrada pelo ciclo Vermelho-Verde-Refatorar, mostrado na Figura 3.11.

Figura 3.11 | Ciclo Vermelho-Verde-Refatorar

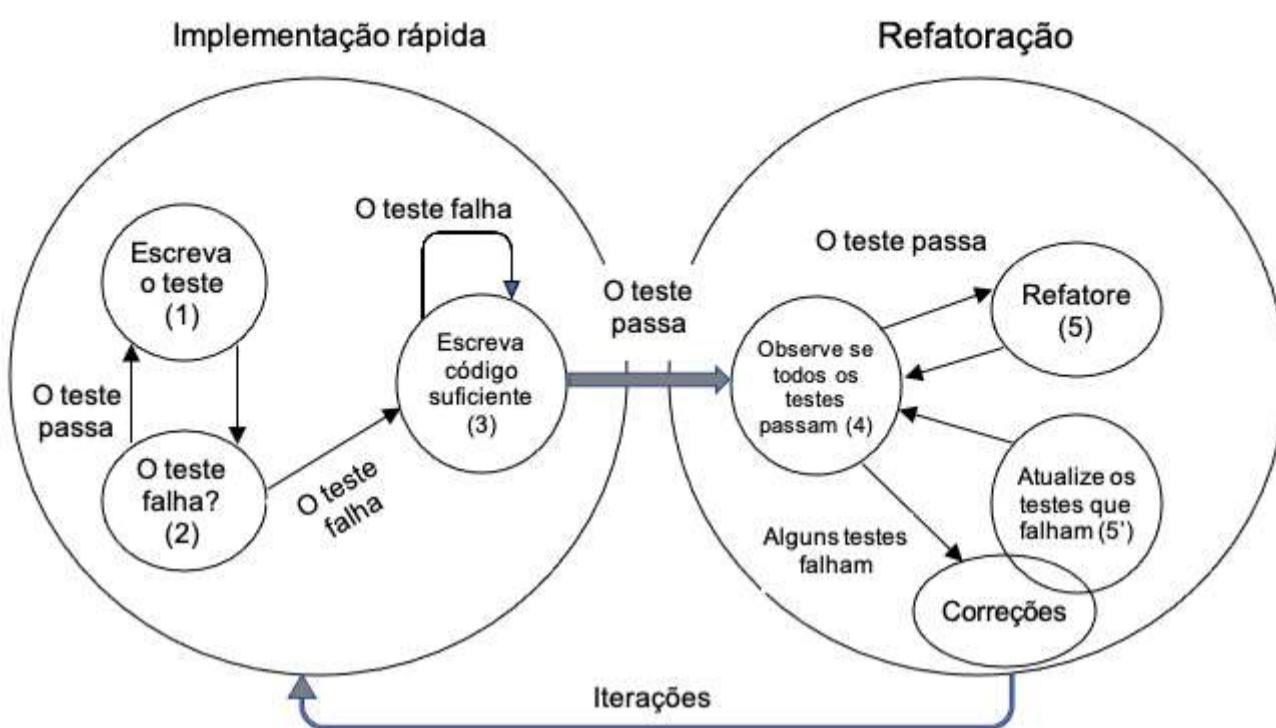


Fonte: adaptada de Qiu (2018).

Essa representação pode ser traduzida de modo textual da seguinte forma:

1. Na primeira fase do ciclo, o desenvolvedor escreve um teste que intencionalmente falhará. O código em que ele está inserido provavelmente sequer compilará.
2. Na sequência, o desenvolvedor escreve um código – referente a uma nova funcionalidade do sistema – que fará o teste escrito na etapa anterior passar.
3. Como última dessas três etapas, o desenvolvedor aplicará a refatoração no código, eliminando eventuais duplicidades, estruturando melhor o código, mudando o nome de variáveis, entre outras providências.

Embora essa ilustração nos dê ideia de um ciclo, ela não deixa claro como ele se efetiva, principalmente para quem se propõe a usar o TDD pela primeira vez. A Figura 3.12 nos oferece visão detalhada do procedimento. Na verdade, o TDD contém duas partes: implementação rápida e refatoração e, na prática, o teste para implementação rápida não se limita ao teste de unidade. Pode ser um teste de aceitação também.



Fonte: adaptada de Qiu (2018).

Observe que, na parte de implementação rápida, o procedimento executado pelo desenvolvedor é visualmente descrito da mesma forma que o fizemos textualmente, ou seja, (1) o desenvolvedor escreve um teste que deverá falhar e (2) verifica se, de fato, o teste falha. Caso não falhe, o teste deverá ser reescrito. Caso falhe, (3) o desenvolvedor deverá escrever o código para que ele passe. Neste ponto um registro deve ser feito: a expressão “código suficiente” indica que o desenvolvedor só deverá codificar o suficiente para que o teste passe, nem mais, nem menos. Quando o teste passa, a parte da refatoração começa e, como primeira providência dessa etapa, (4) o desenvolvedor deve verificar se todos os testes passam e (5) aplicar a refatoração no código. Caso um ou mais testes falhem, (5') eles devem ser atualizados e eventuais correções devem ser feitas. A seta com a legenda de “Iterações” indica que esse procedimento deve ser executado até o fim do procedimento.

REFLITA

No início do desenvolvimento desse tema, foi mencionado que o teste de unidade seria utilizado como o meio de aplicar o TDD. Em relação a essa colocação, vale a reflexão: essa forma de aplicar testes estaria limitada ao teste de unidade apenas? Na verdade, a aplicação do teste de aceitação pode

ser uma ideia melhor, pois garante que o desenvolvedor está fazendo a coisa certa já na primeira vez. Devemos nos lembrar de que os testes de aceitação representam a perspectiva do usuário final e, por isso, um teste de aceitação que passa é a garantia de que o código atende aos requisitos de negócios. Em consequência, esse fato pode ajudar o desenvolvedor a não perder seu tempo de forma desnecessária.

Neste ponto cabe um exemplo da aplicação do TDD e novamente usaremos a loja de comércio eletrônico para desenvolvê-lo, segundo Aniche (2014). A classe exibida no código do Código 3.7 aponta o produto de maior valor e o produto de menor valor no carrinho e o faz percorrendo a lista de compras e comparando valores.

Código 3.7 | Classe que retorna o produto de maior valor e de menor valor

```

1  public class MaiorEMenor {
2
3      private Produto menor;
4
5      private Produto maior;
6
7      public void encontra (CarrinhoDeCompras carrinho) {
8
9          for (Produto produto :
10             carrinho.getProdutos()) {
11
12                 if (menor == null ||
13                     produto.getValor() < menor.getValor()) {
14
15                     menor = produto
16
17                 }
18
19                 else if (maior == null ||
20                     produto.getValor() > maior.getValor()) {
21
22                     maior = produto;
23
24                 }
25
26             }
27
28         }
29
30         public Produto getMenor() {
31
32             return menor;
33
34         }
35
36         public Produto getMaior() {
37
38             return maior;
39
40         }
41     }
42 }
```

[Ver anotações](#)

Fonte: Aniche (2014, [s.p.]).

O método `encontra()` recebe um `CarrinhoDeCompras` e percorre a lista de produtos desse carrinho, comparando sempre o produto atual com o menor e maior já encontrados. Ao final da execução, temos nos atributos `maior` e `menor` os produtos desejados. O Código 3.8 exemplifica o uso da classe `MaiorEMenor`. Observe que o carrinho contém três itens: o que tem preço menor é o jogo de pratos e o que tem preço maior é a geladeira. Ao executar essa aplicação, a saída será:

O menor produto: jogo de pratos

O maior produto: geladeira.

Código 3.8 | Classe que utiliza a classe MaiorEMenor

```

1  public class TestaMaiorEMenor {
2      public static void main(String[] args) {
3
4          CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
5          carrinho.adiciona(new Produto("Geladeira", 450.0));
6          carrinho.adiciona(new Produto("Liquidificador", 250.0));
7          carrinho.adiciona(new Produto("Jogo de pratos", 70.0));
8
9          MaiorEMenor algoritmo = new MaiorEMenor();
10         algoritmo.encontra(carrinho);
11
12         System.out.println("O menor produto: " +
13             algoritmo.getMenor().getNome());
14         System.out.println("O maior produto: " +
15             algoritmo.getMaior().getNome());
16     }
17 }
```

Ver anotações

Fonte: Aniche (2014, [s.p.]).

Sendo assim, podemos concluir que a aplicação funciona, certo? Ainda não. Precisamos verificar se ela funciona também em outro cenário. Se o código do desenvolvedor fizesse a inserção dos mesmos produtos em ordem diferente (geladeira, liquidificador e jogo de pratos), a saída gerada seria a seguinte:

menor produto: jogo de pratos.

*Exception in thread "main" java.lang.NullPointerException at
TestaMaiorEMenor.main(TestaMaiorEMenor.java:5).*

É fato que, se os produtos forem adicionados em ordem decrescente, a classe não retorna a saída esperada e o cliente não conseguirá efetuar sua compra. Essa situação nos leva a duas conclusões:

1. Testar constantemente, considerar vários cenários e repetir o procedimento a cada mínima alteração feita são providências indispensáveis em um procedimento de teste.
2. Proceder manualmente (ou seja, sem uma ferramenta de teste), conforme determina o item anterior, é impraticável.

Ainda nesta seção trataremos de testes automatizados de software como forma de apresentarmos uma solução possível para este caso. Antes, porém, colocaremos como tema algumas questões sobre o teste de unidade e seu gerenciamento, sempre no âmbito do Desenvolvimento Orientado a Testes, e, na sequência, será apresentada uma solução viável para a situação que acabamos de tratar.

GERENCIAMENTO DE TESTES

Na maioria dos casos, criar testes antes para codificar depois não é exatamente o modelo de desenvolvimento com o qual um profissional de TI está acostumado. Como se a falta de familiaridade com esse estilo não fosse obstáculo importante o suficiente, há questões relacionadas ao teste de unidade que demandarão o devido gerenciamento por parte de quem estiver à frente deles. É bom lembrarmos que o teste de unidade é o elemento central do TDD, embora ele possa ter como objeto também o teste de aceitação. Na visão de Teles (2004), as situações a seguir requerem bom gerenciamento para que a condução do TDD seja satisfatória.

Como escrever testes quando o sistema faz acesso a um banco de dados?

Nesta questão, o desempenho é um aspecto a ser seriamente considerado no procedimento de teste. É necessário que os testes de unidade sejam executados muito rapidamente, de modo que o ciclo

“testar-codificar-refatorar” seja completado também rapidamente. O desempenho do teste será tanto mais significativo quanto mais numerosas forem as classes que acessam um banco de dados. O gerenciamento da situação inclui fazer com que uma boa quantidade de testes que usam dados do banco passem a acessar arquivos na memória volátil, em vez de o fazerm no registro em disco. Uma boa alternativa seria escrever uma classe “falsa”, que atuasse como um banco de dados, interceptando os comandos SQL enviados pela aplicação. Embora essa solução seja viável, em algum momento o acesso ao banco real deverá ser testado, mas o bom gerenciamento da situação deverá providenciar que tais acessos sejam feitos com a menor frequência possível.

O que o desenvolvedor deve fazer quando não tiver ideia de como testar uma classe?

Embora a condução dessa situação esteja diretamente relacionada à classe específica, ainda assim é natural imaginar que estamos diante de uma classe problemática. Se a instanciação da classe é feita de forma complexa, é provável que um padrão viável de codificação não foi seguido, o que torna aconselhável a revisão do código. Se a classe sob teste colabora com diversas outras classes de complexidade elevada, a providência deverá passar pela criação de *mock objects*, definidos como objetos falsos, os quais simulam o comportamento de objetos reais e mais complexos.

ASSIMILE

O termo *mock objects* (ou objeto de simulação, em língua portuguesa) é utilizado para descrever um caso especial de objetos que imitam objetos reais para teste. Eles podem ser criados através de *frameworks* que facilitam bastante o processo de criação. Praticamente todas as principais linguagens possuem *frameworks* disponíveis para a criação de *mock objects* (MEDEIROS, 2014).

O gerenciamento dessa situação passa por reuniões com a equipe de desenvolvedores sobre o que tem sido difícil submeter ao TDD.

Como saber se foi testado tudo o que poderia dar errado?

As características de bom senso e de experiência são componentes desta questão. Quando o sistema manifesta um erro no teste de aceitação, há boa chance de estar faltando um teste de unidade. Ao escrever esse teste, o desenvolvedor deve ser orientado a lembrar-se de outros testes que pode não ter escrito e essa providência tenderá a torná-lo mais desenvolto na questão da completude dos testes.

Os desafios que envolvem o TDD não se resumem a esses que abordamos. Será normal nos depararmos com quem presume ser esta uma metodologia que aumenta o trabalho da equipe de desenvolvimento. No entanto, como o TDD prevê que os testes devem ser escritos antes da efetiva codificação, a simplificação da implementação das classes deverá ser um dos bons efeitos desse estilo de desenvolvimento. Como não podemos nos esquecer de que estamos tratando de uma prática do XP, é necessário mencionar que a programação em par constitui um ponto altamente positivo quanto à celeridade da construção dos testes, justamente pela forte interação que se estabelece entre o par.

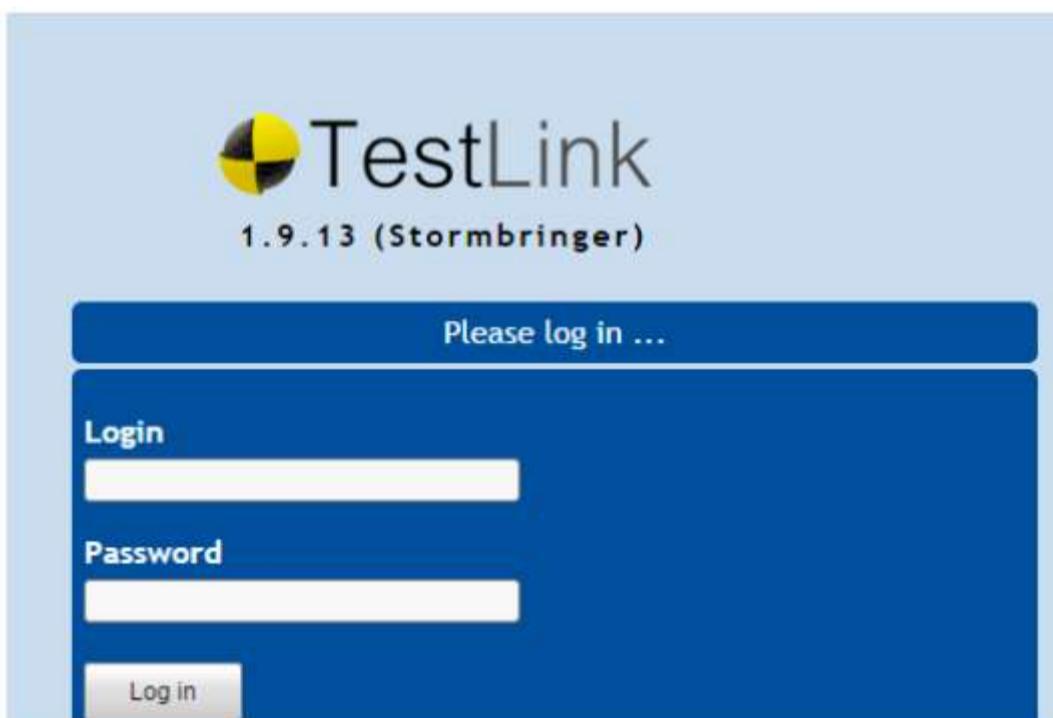
Embora a condução dessas questões deva estar sempre na ordem do dia dos gestores envolvidos em testes, o gerenciamento do processo de teste envolve outros elementos procedimentais e outras ações, já introduzidas em conteúdos anteriores, quando tratamos de planos de testes. Cabe-nos, portanto, fazer alguns resgates de conceitos pontuados naquela seção e apresentar uma ferramenta de gerenciamento de teste bastante utilizada: a TestLink. Antes de tratarmos especificamente dela, vale a observação de que não apenas o procedimento de teste deve ser automatizado, como teremos oportunidade de constatar na sequência. O gerenciamento desse

procedimento, por meio de uma ferramenta, implica facilidades que vão desde o cadastramento de projetos de teste até a designação dos testadores responsáveis por um conjunto de testes.

A apresentação da TestLink tem início pela menção de que se trata de uma ferramenta open source automatizada escrita em PHP e com interface web, cujo principal objetivo é dar suporte às atividades de gestão de testes, permitindo a criação de planos de testes e a geração de relatórios com diversas métricas para o acompanhamento da execução deles. Por meio dessa ferramenta, é possível associar casos de teste a requisitos específicos do produto (CAETANO, [s.d.])

O download da ferramenta deve ser feito no site TestLink (TESTLINK, [s.d.]) e, em vez de apresentarmos aqui os procedimentos de instalação, apresentaremos algumas telas da versão 1.9.13, que reproduzem a criação de um plano de testes, segundo Reis (2018). A Figura 3.13 exibe a tela de login da ferramenta.

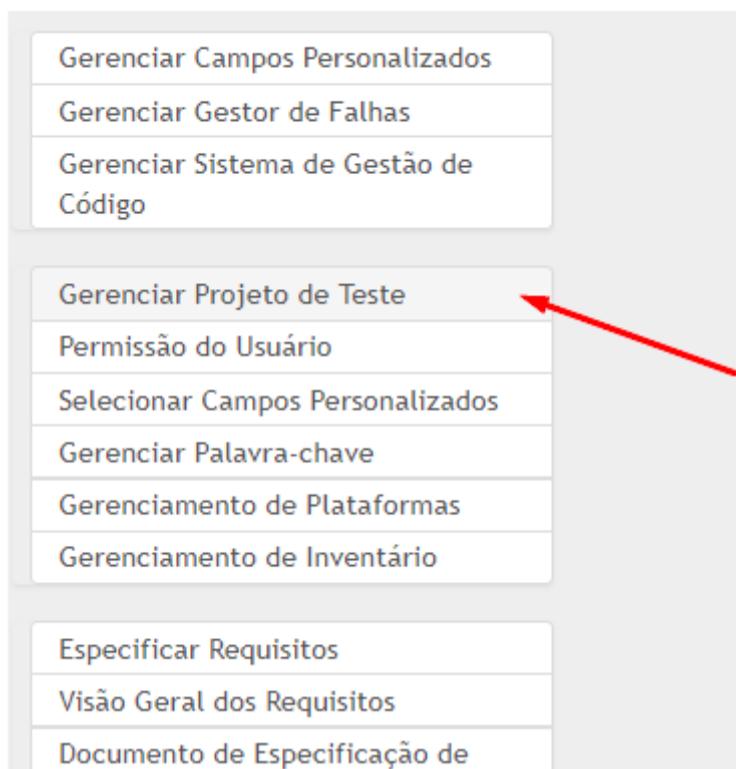
Figura 3.13 | Tela de login da TestLink



Fonte: Reis (2018, [s.p.]).

Após obter acesso à ferramenta, seu usuário poderá criar um Projeto de Teste clicando no link Gerenciar Projeto de Teste, conforme exibido na Figura 3.14.

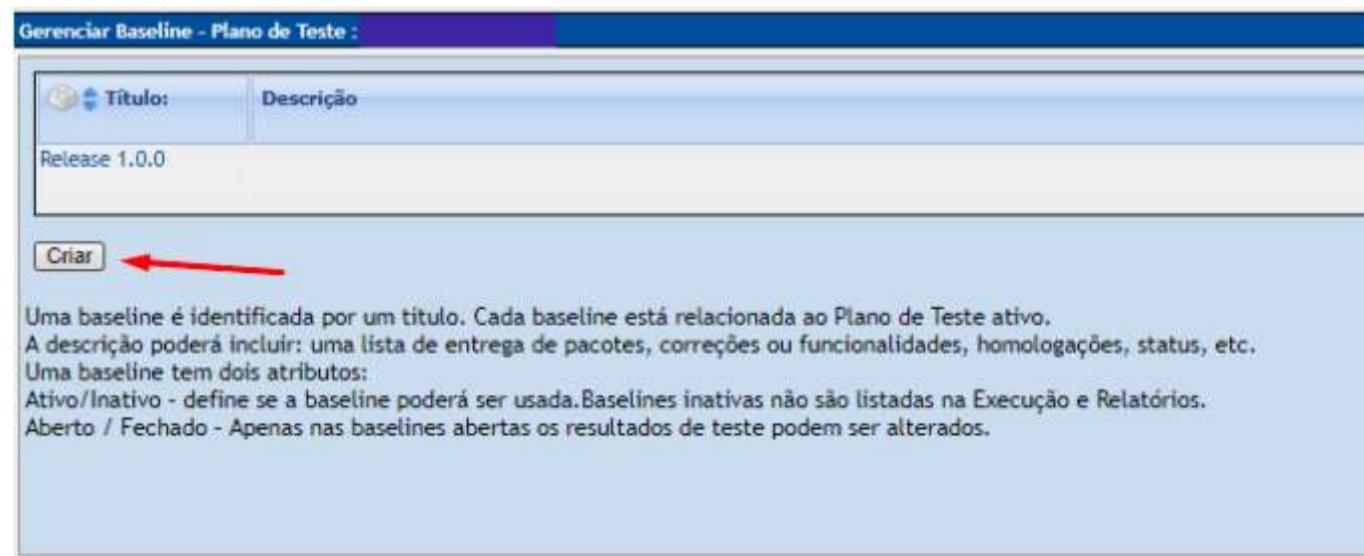
Figura 3.14 | Criação de um Projeto de Teste



Fonte: Reis (2018, [s.p.]).

A partir dessa escolha, o usuário terá acesso ao formulário de criação do Projeto de Teste, por meio do qual poderá informar um nome, a descrição do projeto e habilitar algumas funcionalidades, incluindo a possibilidade de automatizar o teste. A ferramenta deverá, então, habilitar o gerenciamento do plano de teste e, nesse contexto, uma baseline/release do projeto poderá ser criada. A Figura 3.15 exibe o resultado dessa criação e as orientações fornecidas pela ferramenta.

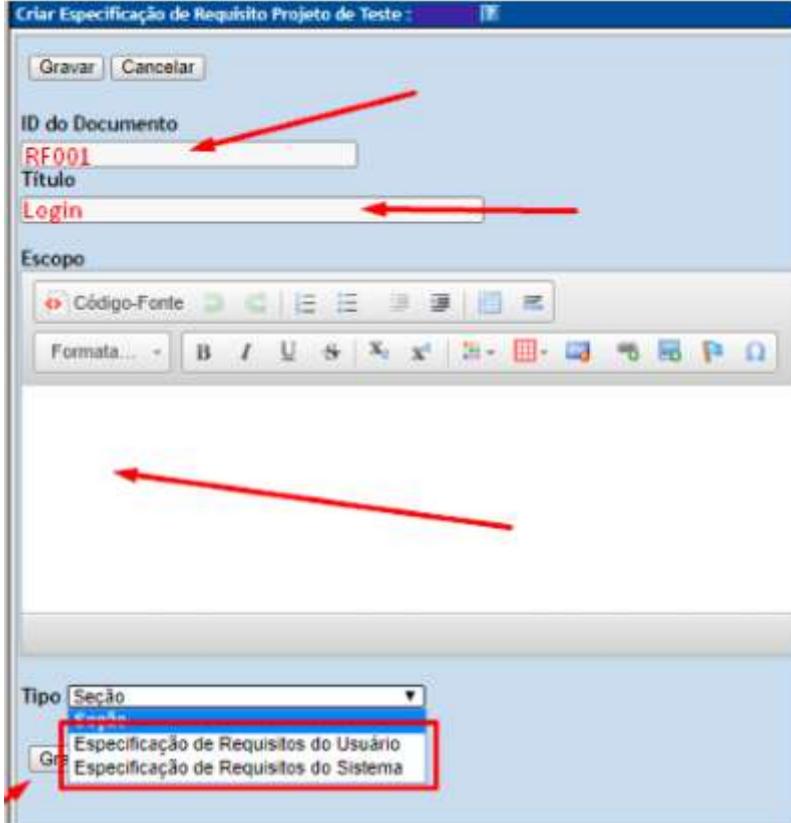
Figura 3.15 | Criação da *baseline/release* do projeto



Fonte: Reis (2018, [s.p.]).

Seguindo esse mesmo padrão procedural e visual, a ferramenta também permite o registro de requisitos e de casos de teste, sempre associados ao projeto atual. Na Figura 3.16, é possível visualizar o contexto de especificação de requisitos do software.

Figura 3.16 | Criação da especificação de requisitos



Fonte: Reis (2018, [s.p.]).

Embora a TestLink ofereça inúmeras outras funcionalidades, fica claro que se trata de um recurso extremamente valioso nos casos em que vários processos de teste estão sendo executados simultaneamente. À propósito, já que mencionamos algo valioso, que tal estudarmos os procedimentos e os benefícios de um teste automático? Siga conosco.

| TESTES AUTOMATIZADOS DE SOFTWARE

Em momentos anteriores desta seção, esclarecemos, por meio de um exemplo, que testar uma unidade considerando vários cenários e por diversas vezes é uma ação fundamental para a obtenção da desejada qualidade do produto. Embora esta seja uma necessidade, não se pode atendê-la apenas contando com o empenho humano. É preciso automatizar o teste. Teles (2004) ensina que os testes de unidade são automatizados na forma de classes do sistema, as quais têm como objetivo testar outras classes. O autor complementa que, de modo geral, para cada classe do sistema deverá existir uma outra com o único objetivo de testá-la e que esse é o passo inicial para que seja possível automatizar os testes de unidade.

De fato, automatizar o processo reduz o custo e o tempo do teste, se é que podemos separar esses dois fatores. Aniche (2014) afirma que escrever um teste automatizado não é tarefa complexa, pois se

assemelha a um teste manual. Para validarmos essa afirmação, voltemos à aplicação de comércio eletrônico. Dessa vez, o desenvolvedor precisa testar o carrinho da loja virtual e, para este exemplo, consideraremos que há dois produtos cadastrados na loja. Ao simular o comportamento de um cliente, ele seleciona os dois produtos, segue para o carrinho de compras e finalmente verifica a quantidade de itens existentes nele. Essa quantidade deve ser dois e o valor da compra deve ser a soma dos valores dos dois produtos.

O que esse desenvolvedor fez foi imaginar um cenário (a compra de dois produtos), executar uma ação (colocá-los no carrinho) e verificar o resultado (checar a quantidade e o valor dos itens comprados). Em um teste automatizado de software, a sequência a ser cumprida deverá ser exatamente essa e a intervenção humana fica restrita, neste caso específico, à conferência do valor obtido com o valor esperado. Com um breve retorno ao Código 3.8 , veremos que aquela classe monta um cenário (um carrinho de compras com três produtos), executa uma ação (chama o método `encontra()`), e valida a saída, que significa imprimir o maior e o menor produto. A simples execução da classe monta o cenário e executa a ação, sem intervenção do desenvolvedor (ANICHE, 2014).

Embora já tenhamos automatizado duas das três etapas da sequência, ainda será necessário tornar a conferência da saída independente da intervenção humana. A plena automatização do teste virá com a utilização do JUnit, o *framework* de testes de unidade do Java, que funciona em conjunto com o IDE Eclipse. A adaptação do nosso código ao JUnit será simples e atingirá a parte da validação, na qual os métodos do framework serão invocados para que comparem o resultado esperado com o obtido. O método `Assert.assertEquals()` fará esse trabalho. O Código, 3.9, mostra a classe `TestaMaiorEMenor` ajustada para funcionamento do JUnit. Como sabemos, esse teste falhará, pois há um defeito na classe `MaiorEMenor`, instanciada pela `TestaMaiorEMenor`.

A presença do else naquele código não permite que o segundo if seja executado e, portanto, o maior elemento nunca é verificado. Com esse defeito corrigido, o teste passará.

o

Código 3.9 | Código adequado para teste no JUnit

```
1 import org.junit.Assert;
2 import org.junit.Test;
3 public class TestaMaiorEMenor {
4     @Test
5     public void ordemDecrescente() {
6         CarrinhoDeCompras carrinho = new CarrinhoDeCompras();
7         carrinho.adiciona(new Produto("Geladeira", 450.0));
8         carrinho.adiciona(new Produto("Liquidificador",
9             250.0));
10        carrinho.adiciona(new Produto("Jogo de pratos",
11            70.0));
12
13        MaiorEMenor algoritmo = new MaiorEMenor();
14        algoritmo.encontra(carrinho);
15        Assert.assertEquals("Jogo de pratos",
16            algoritmo.getMenor().getNome());
17        Assert.assertEquals("Geladeira",
18            algoritmo.getMaior().getNome());
19    }
20}
```

Ver anotações

Fonte: Aniche (2014, [s.p.]).

Dessa forma é que um teste automatizado se efetiva. Naturalmente que a complexidade, a quantidade e o tamanho das classes tornarão os testes mais ou menos complexos, mas a forma geral não se altera significativamente. A agilidade no procedimento, a redução de custos e a capacidade de o teste ser repetido quantas vezes forem necessárias são apenas algumas poucas vantagens da automação.

Antes de terminarmos esta seção, cabe ainda uma questão: com tantas vantagens e recursos interessantes, só temos uma ferramenta de automação de testes disponível?

0

| FERRAMENTA PARA TESTES DE SOFTWARE

Por meio do exemplo de teste automatizado que acabamos de desenvolver, você teve contato com uma das mais conhecidas e utilizadas ferramentas de teste que existem: a JUnit. No entanto, ela está longe de ser a única e, dada a importante função técnica exercida por essas ferramentas, dedicaremos as próximas linhas à explanação de uma delas.

Ver anotações

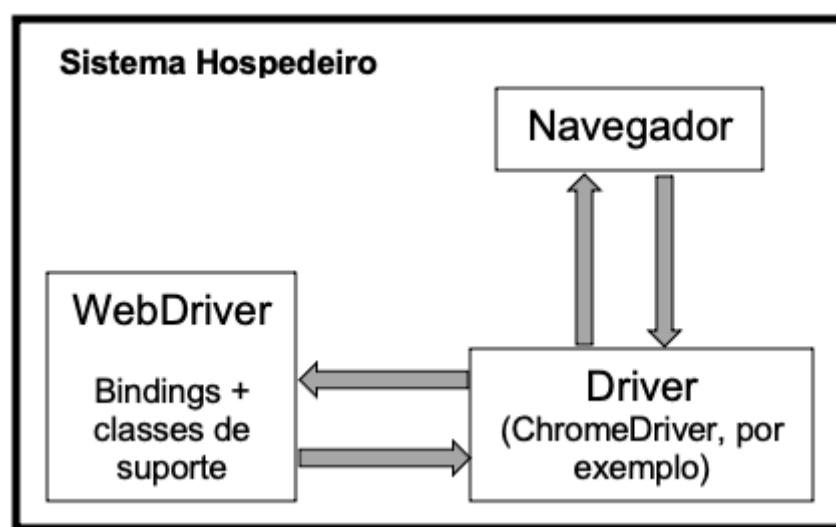
| SELENIUM

Trata-se de uma ferramenta de testes *open source*, multiplataforma e com várias frentes de utilização, especialmente em testes de aplicações web. O site oficial da ferramenta (SELENIUM, [s.d.]) apresenta três projetos e indica o mais apropriado para cada aplicação.

Selenium WebDriver: indicado para quem deseja criar testes automatizados em aplicações baseadas em navegador. O Selenium WebDriver é capaz de automatizar interações com a maioria dos navegadores, tanto local quanto remotamente e realizar a simulação da operação de um usuário real é o seu objetivo.

Observe a Figura 3.17. Por meio dela é possível observar que o WebDriver se comunica com um navegador através de um *driver* de modo bidirecional: o WebDriver passa os comandos para o navegador pelo *driver* e recebe as informações pela mesma rota. O *driver* é específico para o navegador, como o ChromeDriver é para o Chrome, como o GeckoDriver é para o Mozilla Firefox, entre outros. O *driver* é executado no mesmo sistema do navegador.

Figura 3.17 | Comunicação entre componentes do WebDriver



Fonte: adaptada de Selenium ([s.d.]).

Por meio do WebDriver, o Selenium oferece suporte a todos os principais navegadores do mercado, como o Chrome, o Firefox, o Internet Explorer, o Opera e o Safari. Sempre que possível, o WebDriver dirige o navegador usando o suporte integrado dele para a automação, mesmo que nem todos os navegadores tenham suporte oficial para controle remoto. Embora todos os drivers compartilhem uma única interface voltada ao usuário para controlar o navegador, eles têm maneiras ligeiramente diferentes de configurar suas sessões. Como muitas das implementações de driver são fornecidas por terceiros, eles não estão incluídos na distribuição padrão do Selenium (SELENIUM, [s.d.]).

Selenium IDE: trata-se de uma extensão para os navegadores Chrome e Firefox que permite a gravação e a reprodução dos testes neles.

Selenium Grid: esta modalidade do Selenium é capaz de executar testes em várias máquinas ao mesmo tempo, reduzindo, assim, o tempo necessário para realizar testes em vários navegadores ou Sistemas Operacionais.

Este foi, portanto, o conteúdo que queríamos compartilhar com você nesta seção. Longe de ser uma mera opção, automatizar testes é uma providência absolutamente necessária em ambientes profissionais de Engenharia de Software. Como tivemos oportunidade de discutir, as reduções nos custos e no tempo de execução excedem qualquer tempo que se leve para que se possa conseguir familiaridade com as

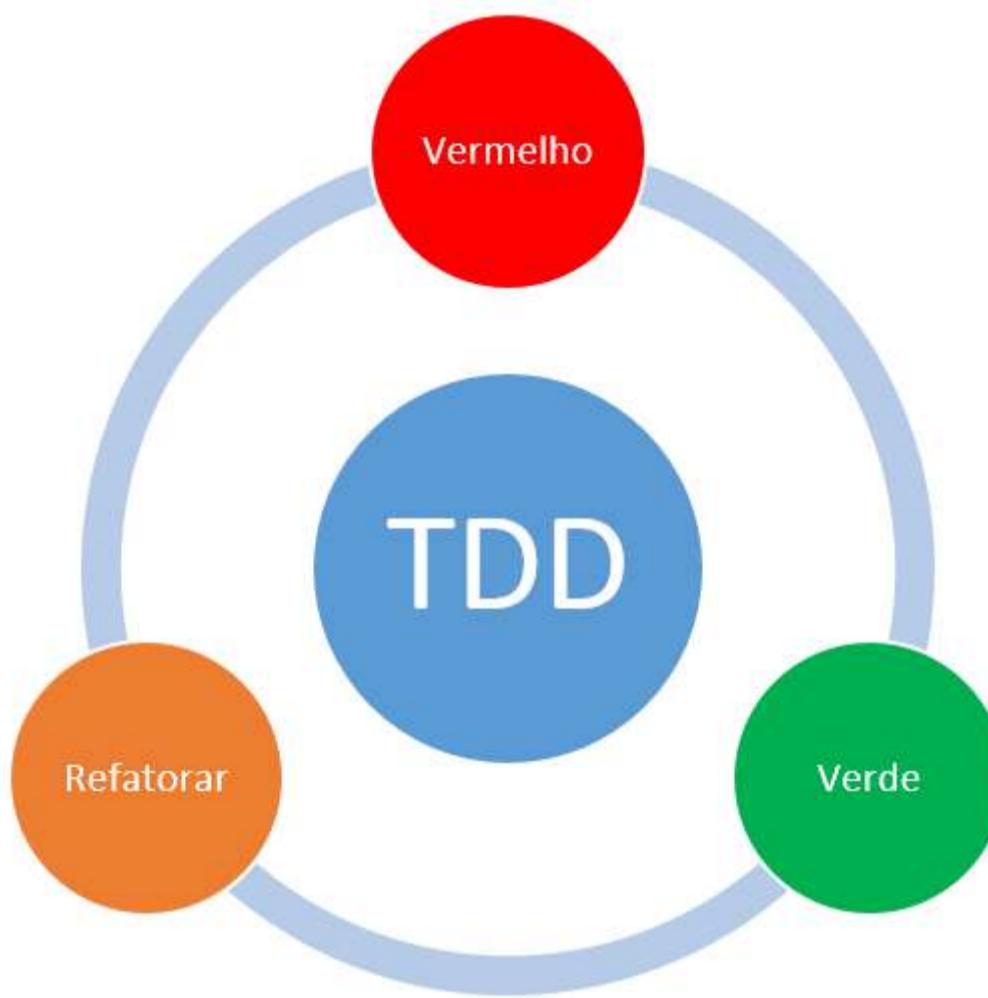
ferramentas de teste. A respeito do custo, inclusive, tivemos a oportunidade de conhecer duas ferramentas com custo zero de aquisição e implantação. Continue focado em seus estudos e boa sorte!

FAÇA VALER A PENA

Questão 1

Observe o esquema representado na figura que segue:

Figura | Representação do processo do TDD



Fonte: adaptada de Qiu (2018).

Ela expressa o formato geral do procedimento de Desenvolvimento Orientados a Testes. O círculo em vermelho representa um teste que falha, o círculo verde representa um teste que passa e o outro círculo representa a ação de refatoração.

Em relação à refatoração, assinale a alternativa que contém, nesta sequência, seu conceito resumido e o momento em que é aplicada.

a. Reimplementação do código – ao finalizar o teste.

b. Aplicação de ajustes no código – após o teste passar.

c. Aplicação de ajustes no código – após o teste falhar.

d. Reimplementação do código – após o teste passar.

Questão 2

Observe o esquema expresso na figura abaixo.

0

Ver anotações

Figura | Esquema Cenário - Ação - Validação



Fonte: elaborada pelo autor.

Em relação a ele, avalie as afirmações que seguem:

- I. Trata-se das etapas de um teste executado em uma unidade e que são passíveis de serem automatizadas por uma ferramenta.
- II. Representa a criação de um cenário de teste, ou seja, uma ação que seria executada pelo usuário e a checagem do resultado.
- III. A sequência pode ser alterada sem que o resultado seja prejudicado, já que ela representa o funcionamento de uma ferramenta automatizada.

É correto o que se afirma em:

a. II e III apenas.

b. II apenas.

c. I apenas.

d. I e II apenas.

e. I, II e III.

Questão 3

Imagine o seguinte cenário: ao iniciar a oficialização do TDD como metodologia de testes da empresa de desenvolvimento que gerencia, você se deparou com algumas dificuldades que, apesar de todo seu

cuidado de planejamento, não haviam sido previstas. Após conseguir delineá-las, você as expressou textualmente para que pudesse discuti-las com a equipe, ação que resultou no seguinte:

- I. As barreiras técnicas para a implantação do TDD justificam o atraso no atingimento desse objetivo.
- II. Os desenvolvedores consideram que o TDD aumentará o trabalho deles e o tempo investido não será compensado a longo prazo.
- III. Os proprietários da empresa entendem que a programação em par nada mais é do que um meio de atingir com dois desenvolvedores o objetivo que poderia ser atingido com apenas um.

Com base nesse cenário, assinale a alternativa cujos itens, de fato, representam um fator que dificulta a implantação do TDD.

a. II apenas.

b. I apenas.

c. II e III apenas.

d. I, II e III.

e. I e III apenas.

REFERÊNCIAS

ANICHE, M. **Test-Driven Development**. [S.I.]: Casa do Código, 2014. E-book.

ANICHE, M. **Testes automatizados de software**: Um guia prático. [S.I.]: Casa do Código, 2015. E-book.

ARAÚJO, C. **Entendendo anotações em Java**. DevMedia, [S.I.], 2012. Disponível em: <https://bit.ly/3iPtMp8>. Acesso em: 31 dez. 2020.

CAETANO, C. Gestão de Testes Ferramentas Open Source e melhores práticas na gestão de testes. **Engenharia de Software Magazine**, [S.I.], ed. 3, p. 58-66, [s.d.]. Disponível em: <https://bit.ly/3t2sA6r>. Acesso em: 6

MEDEIROS, H. Mocks: Introdução a Automatização de Testes com Mock Object. **DevMedia**, [S.I.], 2014. Disponível em: <https://bit.ly/3qX8XLi>. Acesso em: 29 dez. 2020.

QIU, J. **Acceptance Test Driven Development with React**. [S.I.: s.n.], 2018. E-book.

REIS, L. Testlink – uma ferramenta de gerenciamento de testes de software. **Medium**, [S.I.], 24 ago. 2018. Disponível em: <https://bit.ly/2MrAJRp>. Acesso em: 7 jan. 2021.

SELENIUM. **Selenium Projects**. [S.I., s.d.]. Disponível em: <https://www.selenium.dev/projects/>. Acesso em: 30 dez. 2020.

TELES, V. M. **Extreme Programming**: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec, 2004.

TESTLINK. **TestLink Open Source Test Management**. [S.I., s.d.]. Disponível em: <https://testlink.org/>. Acesso em: 13 jan. 2021.

FOCO NO MERCADO DE TRABALHO

DESENVOLVIMENTO ORIENTADO A TESTES E FERRAMENTAS CASE

Roque Maitino Neto

Ver anotações

AUTOMATIZAÇÃO DOS TESTES

Para estruturar um teste automatizado de software é necessário realizar o planejamento de um cenário de teste, fazer a previsão da ação a ser aplicada e a descrição da conferência a ser feita.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

O planejamento de um cenário de teste, a previsão da ação a ser aplicada e a descrição da conferência a ser feita é a forma básica de se estruturar um teste automatizado de software. Se considerarmos as funções do código fornecido, podemos descrever a sequência cenário – ação – verificação da seguinte forma:

1. Cenário: a criação de três usuários aptos a proporem um lance.

No código você perceberá que os usuários foram criados por meio da criação de objetos da classe `Usuario`, não disponibilizada na figura. No entanto, nada impediria que essa criação se desse por meio de cadastramento via formulário.

2. Ação: cada usuário cadastrado deverá propor um lance.

Neste caso, a proposta é feita pela invocação do método `propoe`, da classe `Leilao`. O lance de cada usuário cadastrado é feito via parâmetro passado no momento da instanciação da classe `Lance`, também não oferecida pela figura.

3. Revisão: verificar se os três usuários foram criados.

O desenvolvedor deverá checar se os três usuários foram criados e se os lances foram dados nos valores estipulados pelo código.

A seleção de várias dessas sequências comporá o planejamento e a execução da automatização dos testes.

NÃO PODE FALTAR

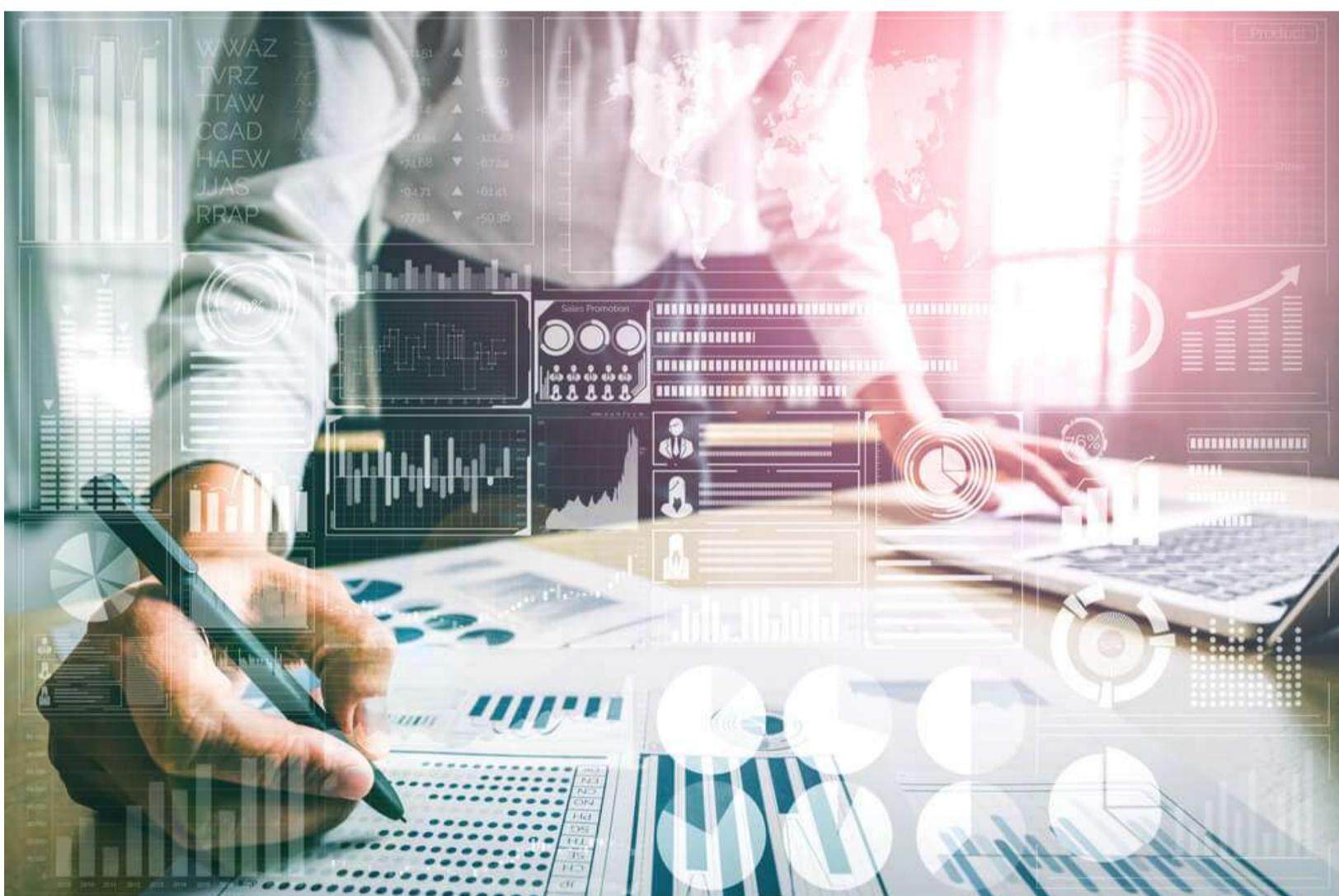
FUNDAMENTOS DE AUDITORIA DE SISTEMAS

Sergio Eduardo Nunes

Ver anotações

O QUE É AUDITORIA DE SISTEMAS DE INFORMAÇÃO?

A auditoria em atividades de desenvolvimento de software têm como objetivo analisar os processos, resultados e sugerir ações corretivas ou melhorias, dessa forma contribuindo com a qualidade dos processos, do produto de software e nos membros da equipe de desenvolvimento.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

C
aro aluno, certamente você já ouviu falar de auditoria. Para alguns profissionais esse nome causa arrepios, pois passar por auditoria, ter o seu trabalho observado, ter apontamentos negativos de algumas atividades desenvolvidas, em alguns casos, exerce certa pressão sobre os profissionais. Mas será que as auditorias são esses “monstros” que os profissionais os enxergam?

Em tecnologia da informação, mais especificamente na área de desenvolvimento de software, as auditorias também fazem parte do ambiente do profissional de sistemas da informação. Dessa forma, é necessário compreender os seus objetivos, particularidades, características e propósitos. Com isso, o profissional passará a enxergar as auditorias como forma de adquirir mais maturidade nos processos e consequentes melhorias.

Na primeira seção, você compreenderá os conceitos relacionados à auditoria em tecnologia da informação, o que lhe permitirá entender como elas são operacionalizadas em ambiente profissional; de que forma ocorre a abordagem técnica de processos de auditorias em desenvolvimento de software com a finalidade de atender as especificidades da área; quais são os tipos de auditorias e, por fim, você poderá analisar como é estruturado o ciclo de vida da auditoria quando aplicada em organizações de desenvolvimento de software.

Já na segunda seção, as discussões visam a uma abordagem para compreensão de como são utilizados os controles gerais de processos de auditoria em sistemas da informação. Em seguida os estudos serão direcionados ao controle de software e de aplicativos a fim de que se mostre como cada plataforma exige uma abordagem específica. Por fim, você poderá ver como é tratada a auditoria durante o ciclo de vida de desenvolvimento do sistema.

Na terceira seção, você compreenderá quais são as necessidades de evolução e de manutenção de software e, depois disso, será possível entender quais são as atividades, processos e ferramentas utilizadas para a manutenção de software. Para finalizar, você verá como a engenharia reversa e a reengenharia de software são utilizadas nas atividades profissionais de desenvolvimento.

Todas essas discussões, em caráter profissional, farão com que você adquira competências a serem utilizadas tanto ao ser auditado, já que conhecerá os processos e rotinas de auditoria de desenvolvimento de software, quanto ao compor equipe para realizar auditoria de software. Assim, esse conhecimento é de extrema importância para os profissionais diretamente ligados a atividades de desenvolvimento de software, pois grande parte das empresas espera que seus colaboradores possuam tais competências.

Pronto para mais esse desafio?

PRATICAR PARA APRENDER

Caro aluno, você já ouviu falar sobre auditoria? Existem diversos segmentos que utilizam essa ferramenta de verificação a fim de gerar informações que permitem compreender as falhas de processos, de produtos e de gestão. As auditorias possibilitam o conhecimento, em detalhes, das atividades desenvolvidas, tornando possível a análise dos processos. Além disso, pode-se dizer que a auditoria não verifica somente falhas, mas também pontos de melhorias e potencialidades.

Você sabia que em Engenharia de Software também são utilizados os processos de auditoria? Evidentemente, aquela que é feita na área de desenvolvimento de sistemas possui algumas particularidades em relação à aplicada em outros segmentos, porém é preciso lembrar que o intuito comum a qualquer auditoria é compreender os processos, diagnosticar falhas ou pontos de melhorias e, posteriormente, buscar os ajustes necessários, ou seja, utilizá-la como instrumento de melhoria.

Interessante, não é?

Incialmente, você será levado a compreender os conceitos de auditoria em tecnologia da informação, com exemplos encontrados nos meios profissionais, o que lhe proporcionará, ainda, entender as atribuições de um auditor.

Em seguida, será possível compreender de que forma são feitas as abordagens de auditoria em sistemas da informação. Em termos práticos, você vai entender como uma auditoria é iniciada, conduzida e finalizada dentro de uma organização, fazendo com que fique clara a importância dos processos de auditoria para a evolução da maturidade dos processos, desenvolvimentos, pessoas e produto de software.

Em seguida, a discussão será voltada aos tipos de auditorias que podem ser feitas, por meio de conceitos teóricos e exemplos profissionais. Esse tema é de grande importância para apreender o último tópico, o qual trata do ciclo de vida da auditoria.

Por fim, para compreender esse último assunto, será demonstrado de que forma que os processos se iniciam, como são planejados e executados e quais são as devolutivas. Em cada um desses subtópicos, você terá exemplos que facilitarão a sua compreensão, para que você possa aplicar os conceitos aprendidos em seu meio profissional.

Em suma, os assuntos discutidos nesta seção permitirão que você conheça os fundamentos de auditoria de sistemas a fim de que sejam utilizados como ferramenta de evolução e de melhoria no desenvolvimento de softwares.

Os sistemas Web são uma solução de grande flexibilidade, pois permitem que os usuários accessem informações em qualquer dispositivo com internet. Mas, assim como qualquer outra forma de desenvolvimento, também ela está suscetível a erros e falhas, as quais por vezes não podem ser detectadas sem que se tenha ferramentas auxiliares para tal finalidade. Pensando nisso, você iniciou um curso voltado a auditorias em sistemas.

Por saber disso, um amigo seu, que trabalha em uma empresa de desenvolvimento de aplicações Web, disse que havia um contrato de trabalho para efetuar processo de auditoria em um desenvolvimento web de uma loja de vendas de artigos variados. Para esse trabalho, o departamento de desenvolvimento enviou o seguinte recado:

1. *Entre no link que o levará a uma pasta do Google Drive (<https://cutt.ly/MjOwsWI>) e faça download dos arquivos.*
2. *Baixe um servidor Web e instale-o em seu computador. Sugere-se o WAMP (WAMPSERVER, [2020]).*
3. *Insira a pasta em C: >> wamp64 >> www >> coloque aqui a pasta.*
4. *No interior da pasta existe um arquivo chamado BD. Abra-o no bloco de notas, notepad ++ ou em qualquer outro editor de texto.*
5. *Acesse o terminal do MySQL no WAMP. Copie todas as instruções no arquivo chamado BD e cole no terminal do MySQL.*
6. *Agora, no navegador de internet, digite: localhost/parque e dê ENTER.*

A página inicial está sendo desenvolvida por outra equipe, então clique no botão ENTRAR, localizado no canto superior direito. É para esse sistema que precisamos de sua auditoria.

Ao acessar o sistema, você terá contato com o desenvolvimento até dado momento. Porém, inicialmente precisamos saber o ciclo de vida de sua auditoria. Poderia nos enviar seu planejamento para isso?

Essa é uma grande oportunidade de colocar em prática de forma profissional os seus estudos. Para isso, o sistema deve ser instalado e em seguida deve ser elaborado o cronograma de auditoria para a empresa poder se organizar.

Você deve ter ficado curioso! Então, vamos descobrir como a auditoria é aplicada na engenharia de software?

CONCEITO-CHAVE

Normalmente, quando as pessoas comentam sobre passar por auditorias, os relatos são sempre carregados de momentos tensos, pressão no trabalho, ajustes de condutas e processos, entre outras situações que causam uma sensação de verificação minuciosa das atividades profissionais desenvolvidas.

Assim como em outras áreas do conhecimento (fiscal, contábil, administrativa, etc.), a Engenharia de Software também conta com processos de auditoria para verificação dos desenvolvimentos de software, pois, como se trata de atividades que necessitam do emprego de habilidades técnicas de programação, criatividade e aplicação correta dos processos, são passíveis de auditoria.

AUDITORIA

Mas, de fato, o que é uma auditoria?

Segundo Cardoso (2015), as atividades de auditoria têm como principal função analisar parcial ou globalmente os processos, resultados e, em alguns casos, sugerir ações corretivas ou melhorias. Elas podem ser aplicadas por diversos métodos (questionário, teste de uso prático, análise documental, entre outras formas), porém os resultados obtidos não devem gerar dúvidas.

Para que você possa compreender em que casos isso será encontrado, vamos tomar como exemplo uma empresa que precise auditar o desenvolvimento de determinada funcionalidade no aplicativo de uma

operadora de convênio odontológico, a qual permite agendamento de consultas. A auditoria foi necessária, pois houve um atraso significativo na entrega do aplicativo. Esse processo serve para compreender em que momento ocorreu o atraso e o porquê ele foi gerado, informações que podem ajudar a corrigir falhas no processo, por exemplo. Nesse caso, a equipe de auditoria poderia analisar os documentos ou fazer entrevistas com os envolvidos, de modo a ter dados o suficiente para gerar informações úteis para a compreensão do problema.

É claro que, embora existam processos de auditoria para verificar as atividades em diversas áreas, a tecnologia da informação possui métodos e normas específicas para si. Segundo Cardoso (2015), as atividades de auditoria em Engenharia de Software são conhecidas por *auditoria em sistemas da informação*. As atividades de auditoria de sistemas não têm apenas a função de verificar códigos, instruções e outras formas de se codificar uma aplicação computacional, mas também a de aplicar os seus esforços conforme demonstrado a seguir:

- **Processos:** diz respeito à forma como são orientados os meios de execução das atividades de desenvolvimento. Em termos operacionais, imagine que a empresa utilize o SCRUM com o objetivo específico de priorizar as atividades. A auditoria de processo poderia analisar especificamente se o SCRUM foi seguido durante o ciclo de vida do desenvolvimento do software.

ASSIMILE

Assim como em diversas áreas, a urgência nas entregas, os picos de alta demanda e a exigência de maior produtividade certas vezes acabam por comprometer alguns atributos importantes da garantia de qualidade do produto. Uma das soluções para esse problema, na área de tecnologia da informação, é a metodologia conhecida como SCRUM.

Ela se trata de uma metodologia *agile* que visa segmentar um projeto de desenvolvimento em pequenos ciclos, reuniões de alinhamentos frequentes para acompanhamento dos desenvolvimentos de softwares.

- **Desenvolvimento:** comprehende a análise dos scripts em si, ou seja, trata-se de uma forma de efetuar a verificação de erros de escrita da linguagem de programação/marcação utilizada. Um exemplo muito atual se refere à auditoria de responsividade de desenvolvimentos Web, na qual é analisada a correta utilização dos frameworks, que têm como objetivo deixar a aplicação responsiva. Nesse caso, os processos de auditoria mostrarão em quais dispositivos existem erros, desconforto de navegação, entre outras avaliações muito úteis a nível de qualidade dos desenvolvimentos.
- **Testes:** a auditoria visa à verificação da eficácia dos testes efetuados nos desenvolvimentos de software. Em termos profissionais, significa que as atividades desenvolvidas pelos testers de software receberão uma auditoria para verificar se de fato as funcionalidades desenvolvidas estão sendo testadas ou se existem falhas, vícios ou qualquer outra inconformidade.
- **Segurança e proteção dos dados:** são aquelas tratativas realizadas quanto à proteção e à segurança dos dados para que não ocorram incidentes indesejados. Um exemplo é quando o processo de auditoria utiliza pentest (profissional de segurança da informação que busca vulnerabilidades) para comprovar, na prática, que um sistema é seguro.
- **Estrutura de desenvolvimento:** está voltada às atividades administrativas e de recursos humanos, ambiente de trabalho e recursos disponíveis para o desenvolvimento profissional. Exemplo: o auditor, dentro de suas análises, poderia voltar a suas atividades profissionais para que seja efetuada a avaliação do ambiente de

trabalho favorece o bom desempenho do profissional de desenvolvimento de software.

VOCABULÁRIO

Os profissionais de tecnologia da informação que efetuam testes de intrusão são conhecidos como *pentest*, uma abreviação da palavra *Penetration Test*. Esses especialistas são altamente requisitados para o processo de auditoria de software voltado à análise de desenvolvimento de software. Isso porque, normalmente, esses profissionais possuem competências e habilidades para efetuar teste de intrusão em: serviços de rede, aplicações web, teste ao lado do cliente (*client side*) e teste de serviços de aplicações web.

O AUDITOR

Certamente você percebeu que existe a possibilidade de efetuar auditorias em muitas áreas que envolvem o desenvolvimento de software. Entretanto, existe um agente dentro dos processos de auditoria de software cujas atribuições e responsabilidades é essencial compreender. Estamos falando aqui do auditor.

Segundo Gallotti (2016), o auditor é o profissional que tem conhecimentos técnicos, gerenciais, operacionais e analíticos para conduzir as atividades relacionadas ao desenvolvimento de auditorias em sistemas da informação.

Para compreender o papel do auditor, observe o Quadro 4.1.

Quadro 4.1 | Atribuições e responsabilidades do auditor de tecnologia da informação

TIPOS	ATRIBUIÇÃO
Inspeção	<ul style="list-style-type: none"> • Efetuar inspeção dos processos e dos desenvolvimentos. • Verificar onde ocorrem inconformidades e apresentar provas convincentes.
Controle	<ul style="list-style-type: none"> • Efetuar a análise de controle dos processos, podendo ser operacionais, de desenvolvimento ou gerenciais.
Risco	<ul style="list-style-type: none"> • Verificar quais riscos podem afetar o projeto, a fim de se indicar a equipe na qual existe um risco. Isso permite que o gerente de projetos possa tomar medidas corretivas/preventivas. • Analisar de forma mais abrangente os riscos, apresentando consequências externas que possam ocorrer.
Contínua	<ul style="list-style-type: none"> • Elaborar relatórios das análises em espaços curtos de tempo. • Utilizar sistemas de verificação constantes para que se faça uma análise de diversos momentos. • Contratar especialistas da área em que é necessária uma análise mais profunda, a fim de se permitir o aumento do detalhamento dos processos.

Fonte: adaptado de Gallotti (2016).

Acerca das responsabilidades e atribuições de um auditor de desenvolvimento de software, vamos a um exemplo: imagine que uma desenvolvedora de games para mobile tenha decidido fazer um jogo exclusivo para um público específico. Um processo de auditoria poderia

responder quais são os impactos positivos e negativos de um jogo cujo tema exclui outros jogadores, porém cabe aos níveis administrativos e gerenciais da empresa, após os resultados das auditorias, tomar uma decisão quanto ao produto elaborado. Uma auditoria aponta falhas, erros, pontos de atenção, no entanto não obriga a empresa a tomar outros rumos. Essa decisão fica a cargo dela.

Vale ressaltar, ainda, que as atividades de auditoria de software são pautadas por testes, análises, avaliações e outras formas que possam provar que algo de inconformidade está ocorrendo ou sendo executado, conforme define Gallotti (2016).

| CICLO DE VIDA DA AUDITORIA

Caro aluno, percebeu a importância da auditoria na Engenharia de Software? As auditorias não devem ser enxergadas como um processo doloroso e negativo, como descrito no início de nossas discussões, mas devem ser encaradas como uma ferramenta de apontamento de falhas, que permite agregar qualidade ao projeto, à equipe, à organização, o que reflete positivamente para o cliente. Mas, em termos práticos, em que momento a auditoria deve ser utilizada durante o projeto de desenvolvimento de software?

Conforme defendem Weill e Ross (2006), os processos de auditoria em sistemas da informação podem ser operacionalizados durante todo o ciclo de vida do projeto de desenvolvimento. O ciclo de vida da auditoria é dividido em quatro processos, que compreendem desde as atividades precedentes à auditoria em si (preparação) até o acompanhamento dela. Os processos de auditoria durante seu ciclo de vida são sequenciais. Para compreender a organização dessas atividades, observe a Figura 4.1.

Figura 4.1 | Ciclos de vida da auditoria



Fonte: adaptada de Gallotti (2016).

De acordo com o observado no ciclo de vida da auditoria, as atividades de auditoria começam antes mesmo de seu início em sistemas da informação. Vale ressaltar, aqui, que neste momento não estamos discutindo o ciclo de vida do projeto, mas o ciclo de vida da auditoria. Em termos de aplicação profissional, estamos agora com foco em todos os processos que compõem uma auditoria em atividades de desenvolvimento de software.

■ PLANEJAMENTO DO CRONOGRAMA

Segundo Vetorazzo (2018), as atividades de elaboração do cronograma mostram o nível de organização da equipe que aplicará a auditoria. Além disso, é possível compreender como o ciclo todo será organizado. Ao divulgar o cronograma da auditoria, a empresa/equipe que será auditada pode ajustar alguns pontos de melhorias. Para que você compreenda como um cronograma de auditoria pode ser organizado, observe o Quadro 4.2.

Quadro 4.2 | Exemplo de cronograma de auditoria

Início	Atividade	Final	Auditor
05/01/2020	Planejamento da auditoria	20/01/2020	Fulano de Tal
21/01/2020	Condução da auditoria	05/02/2020	Fulano de Tal
06/02/2020	Reporte da auditoria	01/03/2020	Fulano de Tal

Fonte: elaborado pelo autor.

Esse cronograma pode ter diversas atividades descritas dependendo do nível de detalhamento de que o projeto necessite. Quanto às informações das colunas, ser minimalista é o suficiente para demonstrar a atividade, as datas inicial e final e o responsável. Mas nada impede que ele apresente outras informações.

Vetorazzo (2018) defende ainda que o planejamento do cronograma de auditoria, em muitos casos, deve ser aprovado pela contratante. Para entender como isso ocorre em termos práticos, imagine que uma empresa desenvolveu um aplicativo para a população votar on-line e com garantia de autenticidade. Porém, antes que ele seja liberado para a população utilizar, é necessário que seja feita uma auditoria. A empresa de auditoria, então, divulga as datas de seu cronograma. Entretanto, há um problema: a data final do reporte está para depois do dia em que deveria ocorrer a votação. Percebeu a importância da aprovação do cronograma da auditoria?

■ PLANEJAMENTO DA AUDITORIA

Para Rainer e Cegielsky (2016), o planejamento da auditoria como primeira atividade está no mapeamento dos processos, o qual identifica individualmente os agentes responsáveis por eles. Ainda é possível que o auditor consulte documentos de auditorias anteriores para que possa voltar suas atividades a apontamentos já feitos. Entre as atividades do planejamento da auditoria, deve-se estar claro qual o objetivo de determinada auditoria, de forma que a empresa de desenvolvimento de software que receberá os auditores possa se adequar às necessidades do cliente.

Por que o objetivo da auditoria deve ser divulgado? Imagine que uma empresa de desenvolvimento tenha quatro equipes: front end, back end, banco de dados e infraestrutura de redes. Mas, devido às muitas reclamações a respeito da performance do tempo de resposta do sistema, será agendada uma auditoria nos setores de desenvolvimento.

Em todas as equipes? Não, somente naquelas que podem impactar diretamente no desempenho da aplicação, ou seja, back end, banco de dados e infraestrutura de redes.

Rainer (2016) defende ainda que o planejamento é um documento que deve ser elaborado pelos auditores de forma que três pontos fiquem claros para equipe de auditores e para a organização/equipe a ser auditada:

- Quem será auditado?
- O que será auditado?
- Quando ocorrerá a auditoria?

REFLITA

A auditoria deve demonstrar claramente qual o seu objetivo.

Além disso, é indicado que a organização/equipe que a receberá saiba previamente quando e como isso vai ocorrer.

Porém, não existe apenas essa forma de execução de processos de auditoria nem somente uma necessidade a ser atendida por ela. Em muitos casos, auditorias são aplicadas de surpresa e o intuito é realmente este: que o auditado não se prepare previamente; assim, poderá ser refletido o que de fato ocorre no dia a dia. Será que esse modo de aplicar auditoria não gera respostas mais fiéis ao que realmente ocorre?

CONDUÇÃO DA AUDITORIA

Conforme defende Vetorazzo (2018), quando o planejamento da auditoria está pronto e aprovado, dá-se seu início, no qual são aplicadas as atividades de coleta de informações, tais como: revisão documental, conversas e entrevistas, análise de dados de processos, observações, entre outras técnicas. Normalmente, a equipe de auditoria possui um checklist com escopo expandido para permitir diversas análises.

Em termos práticos e profissionais, o checklist, para o auditor, é um guia com os pontos a serem auditados. Para que você possa compreender como é estruturado esse documento, observe o Quadro 4.3.

0

Quadro 4.3 | Exemplo de checklist de auditoria

Data	Hora	Atividade	Local	Processo	Auditor	
05/01/2020	08:00	Reunião de abertura	Sala de reunião	----	Todos	
21/01/2020	09:00	Análise documental	Sala de reunião	Questionário de UX	Fulano	
06/02/2020	10:00	Entrevista	Café	Utilização do framework	Fulano	

Fonte: elaborado pelo autor.

Ver anotações

Esse cronograma pode ter diversas atividades descritas dependendo do nível de detalhamento de que o projeto necessite. Quanto às informações das colunas, ser minimalista é o suficiente para demonstrar a atividade, as datas inicial e final, o horário, o local e os responsáveis. Mas nada impede que nele sejam apresentadas outras informações.

■ REPORTE DA AUDITORIA

Segundo Vetorazzo (2018), trata-se da parte de encerramento da auditoria, na qual podem ser feitas reuniões, relatórios, apresentações, entre diversos outros recursos que fornecem a devolutiva do que foi auditado. Dependendo do acordado na contratação desse serviço, podem-se propor melhorias e ajustes. Porém, não cabe à empresa/equipe de auditoria promover as mudanças descritas no reporte.

Em termos práticos, após longos dias de auditoria, é chegada a hora de conhecer os resultados para saber os acertos, os erros e, assim, poder projetar e alcançar novos objetivos e melhorias. Normalmente nesse dia a equipe auditada, com gerentes e demais cargos administrativos, é chamada para uma reunião em que os resultados são apresentados e as sugestões são discutidas.

Ver anotações

EXEMPLIFICANDO

No dia em que são apresentados os resultados de uma auditoria, a empresa marca uma reunião com os auditados e cargos gerenciais/administrativos a fim de expor as conclusões do processo e de discutir as sugestões.

Um objeto interessante para a demonstração dos resultados é o relatório de auditoria, pois podem ser feitas cópias dele, as quais são entregues para cada membro presente na reunião. Abusar dos gráficos e tabelas é uma forma muito eficiente de se demonstrar os resultados, além de ajudar os auditados a compreenderem melhor as análises.

Como você pôde ver ao longo das discussões e dos exemplos apresentados, a auditoria em atividades de desenvolvimento de software pode levar uma equipe à melhoria de seus processos e produtos. A forma como deve ser feita a abordagem da auditoria em sistema da informação visa contribuir com a qualidade dos processos, do produto de software e nos membros da equipe de desenvolvimento. Por fim, as discussões acerca do ciclo de vida da auditoria guiarão o profissional na construção de documentações como cronogramas, checklists e ambientes auditáveis.

PESQUISE MAIS

No capítulo 4 do livro *Introdução a sistemas de informação* (RAINER; CEGIELSKY, 2016), disponível na biblioteca virtual, são tratados os assuntos relacionados à auditoria de

software, sendo este um referencial bibliográfico muito interessante para o assunto. Vale a pena conferir!

RAINER Jr. R. K.; CEGIELSKY, C. G. **Introdução a sistemas de informação.** 5. ed. Rio de Janeiro: Elsevier, 2016.

FAÇA VALER A PENA

Questão 1

Em alguns casos a equipe de auditoria não possui conhecimentos técnicos para analisar, medir e avaliar algumas atividades. Isso ocorre porque é necessário ter habilidades e competências para saber se as atividades estão em conformidade com a qualidade esperada. Dentre alguns profissionais está o *pentest*, que é aquele que utiliza diversas técnicas e ferramentas para tentar penetrar nos sistemas e conseguir apontar, na auditoria, quais são as vulnerabilidades em sistemas.

Assinale a alternativa que apresenta corretamente a área em que o *pentest* realiza a auditoria.

a. Auditoria de processos.

b. Auditoria de desenvolvimento.

c. Auditoria de testes.

d. Auditoria de segurança e proteção de dados.

e. Auditoria de estrutura.

Questão 2

Uma empresa terceirizada que realiza auditorias foi contratada para um serviço. O trabalho em questão deverá ser feito sem que os desenvolvedores tenham ciência de que estão passando por uma auditoria. Com base no exposto, observe as afirmativas a seguir:

- I. Pode se tratar de uma auditoria de inspeção, para fiscalizar se um processo está ocorrendo conforme o determinado pela empresa.

II. Pode ser uma auditoria contínua, pois não é necessária uma análise profunda.

III. Pode ser uma auditoria de controle, para verificar se os processos de desenvolvimento estão alinhados com o determinado.

Assinale a alternativa correta:

- a. Está correta apenas a afirmativa I.
- b. Está correta apenas a afirmativa II.
- c. Está correta apenas a afirmativa III.
- d. Estão corretas apenas as afirmativas I e II.
- e. Estão corretas apenas as afirmativas I e III.

Questão 3

O ciclo de vida da auditoria demonstra como as atividades serão conduzidas ao longo dos processos, sendo essencial, portanto, compreender como tudo é organizado nos processos e como se dá a interação entre eles.

Quanto aos processos no ciclo de vida da auditoria, analise as asserções a seguir e a relação proposta entre elas.

I. O planejamento da auditoria descreve quando, o quê, como, quem e por quem será feita a auditoria.

POIS

II. Tais decisões orientam como deve ser feito o relatório de reporte.

A seguir, assinale a alternativa correta:

- a. As asserções I e II são proposições verdadeiras e a II é uma justificativa correta da I.
- b. A asserção I é uma proposição verdadeira e a asserção II é uma proposição falsa.
- c. As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- d. A asserção I é uma proposição falsa e a II é uma proposição verdadeira.
- e. As asserções I e II são proposições falsas.

REFERÊNCIAS

CARDOSO, A. **Auditoria de sistema de gestão integrada**. São Paulo: Pearson Education do Brasil, 2015.

GALLOTTI, G. M. A. **Qualidade de software**. São Paulo: Pearson Education do Brasil, 2016.

RAINER Jr. R. K.; CEGIELSKY, C. G. **Introdução a sistemas de informação**. 5. ed. Rio de Janeiro: Elsevier, 2016. Disponível em: <https://cutt.ly/wjOtJCU>. Acesso em: 16 nov. 2020.

VETORAZZO, A. de S. **Engenharia de software**. Porto Alegre: SAGAH, 2018.

WEILL, P.; ROSS, J. W. **Governança de TI**: como as empresas com melhor desempenho administram os direitos decisórios de TI na busca por resultados superiores. São Paulo: M. Books do Brasil Editora Ltda., 2006.

FOCO NO MERCADO DE TRABALHO

FUNDAMENTOS DE AUDITORIA DE SISTEMAS

Sergio Eduardo Nunes

Ver anotações 0

PLANEJAMENTO DA AUDITORIA

No planejamento da auditoria a primeira atividade é o mapeamento dos processos, para posteriormente realizar o apontamento de erros e falhas de produção do software.



Fonte: Shutterstock.

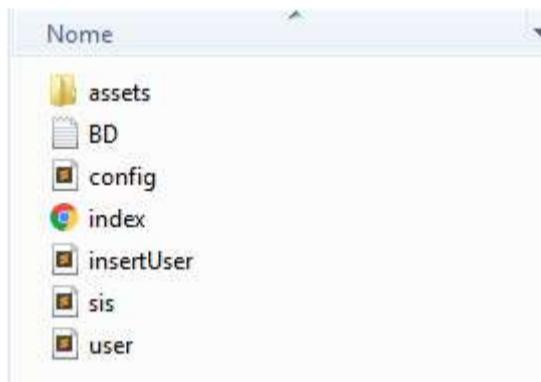
Deseja ouvir este material?

Áudio disponível no material digital.

O contrato de trabalho para efetuar processo de auditoria em um desenvolvimento web de uma loja de vendas de artigos variados, colocou você frente a um desafio muito interessante. Inicialmente, é necessário preparar o ambiente de análise para ter uma noção do cenário que deve ser analisado, permitindo, assim, o desenvolvimento do cronograma de auditoria.

- O primeiro passo é a instalação do WAMP (WAMPSERVER, [2020]). O seu processo de instalação é igual a qualquer outro programa que instalamos normalmente (Atenção: não coloque senha em nenhum dos passos). Dê um duplo clique no ícone criado na área de trabalho para iniciar o WAMP.
- O segundo passo é fazer o download dos arquivos no drive. Os arquivos necessários podem ser observados na Figura 4.2.

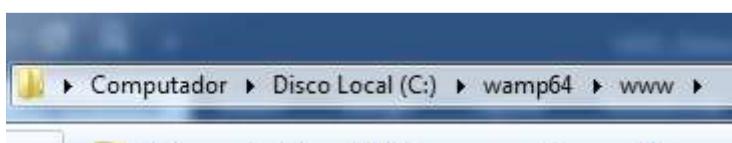
Figura 4.2 | Arquivos para download



Fonte: captura de tela do sistema operacional Windows elaborada pelo autor.

- No terceiro passo, a pasta chamada loja deve ser colocada em C: wmap64 >> www >> coloque a pasta nesse diretório. Conforme pode ser observado na Figura 4.3.

Figura 4.3 | Caminho da pasta www no WAMP

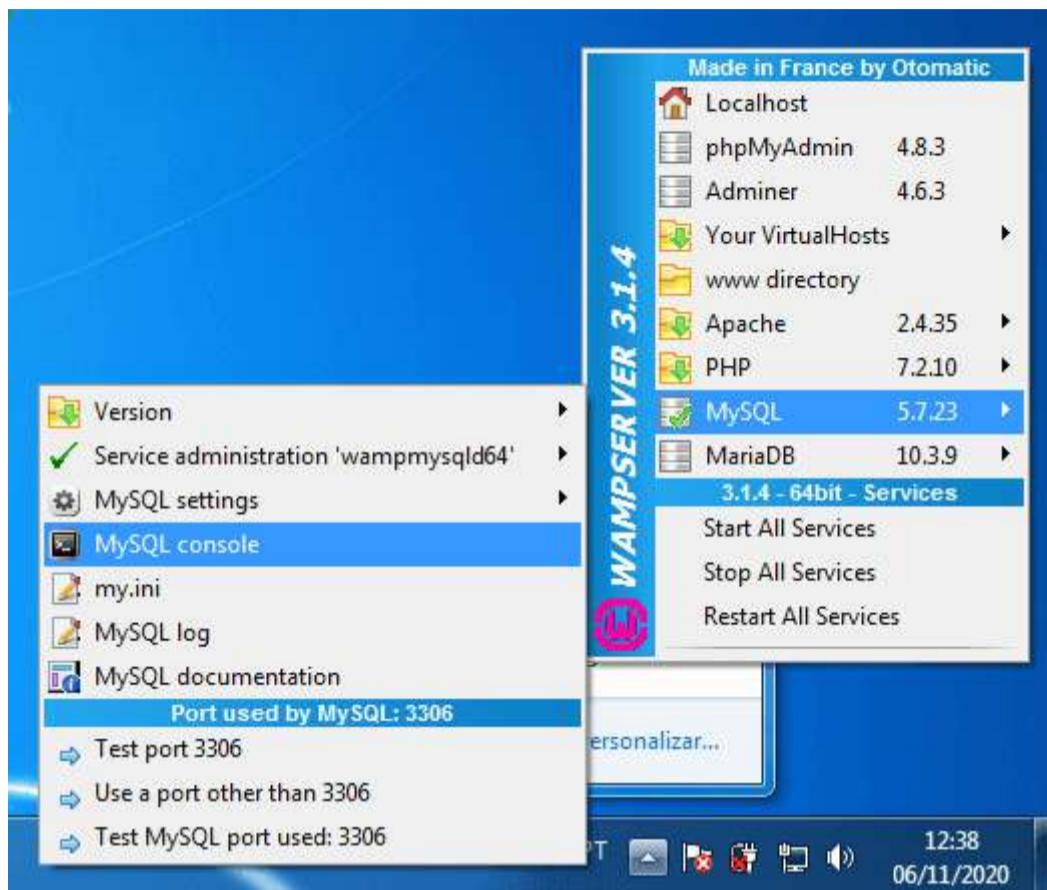


Fonte: captura de tela do gerenciador de arquivos do Windows elaborada pelo autor.

- No quarto passo, está a instalação do banco de dados. Deve ser aberto um arquivo chamado BD no interior da pasta do projeto Loja.

As instruções SQL devem ser copiadas. Em seguida, deve-se abrir o MySQL como mostrado na Figura 4.4.

Figura 4.4 | Acesso ao MySQL



Fonte: captura de tela do WAMP SERVER 3.1.4 elaborada pelo autor.

Com isso será aberto o console (tela igual ao prompt de comando do Windows). Não coloque nenhuma senha, apenas aperte ENTER. Nesse momento é só colar as instruções SQL e dar ENTER (deve ser utilizado o botão direito do mouse para colar).

O sistema está finalmente instalado e deve ser acessado pelo navegador de internet por meio do endereço: localhost/loja. Será aberto o site demonstrado na Figura 4.5.

Figura 4.5 | Página home do site do sistema da loja

O sistema deve ser acessado por meio de um clique no botão ENTRAR, localizado no canto superior direito da página inicial do site. Dessa forma, o sistema será acessado, conforme observa-se na Figura 4.6.

Figura 4.6 | Página do sistema da loja

A captura de tela mostra uma interface web com uma barra superior preta contendo links para 'Usuários', 'Clientes' e 'Produtos'. Abaixo disso, o título 'USUÁRIOS' está centralizado. Um link 'Novo Usuário' aparece no topo da lista de usuários. A tabela contém duas linhas de dados:

Nome	E-mail	Senha	Excluir
Serginho	teste@teste.com	81dc9bdb52d04dc20036dbd8313ed055	
Testando	teste@teste.com	81dc9bdb52d04dc20036dbd8313ed055	

Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

O acesso ao sistema é importante porque ele dará uma base de como organizar o planejamento da auditoria, isto é, com um olhar voltado somente ao sistema, pois o site está sendo desenvolvido por outra equipe. Dessa forma, foi proposto o seguinte planejamento:

Planejamento da auditoria

Dias 1 a 3 - Levantamento dos processos utilizados.

Entrevistas com gerente de projetos e desenvolvedores: o objetivo dessa entrevista é a aproximação com os envolvidos na auditoria (primeiro contato). Nesse bate papo, devem ser feitos questionamentos a fim de se compreender os processos e o relacionamento entre eles. A entrevista pode ser gravada (somente áudio ou áudio e vídeo), mas obrigatoriamente essa etapa deve gerar uma documentação com o detalhamento dos processos.

Dias 4 e 5 - Compreensão da operacionalização dos processos de desenvolvimento.

Observações *in loco*: o levantamento dos processos permite conhecer quais etapas são desenvolvidas nas atividades. Com isso, deve-se posicionar um auditor com a função apenas de observar as atividades

de desenvolvimento, uma vez que não pode interagir com o colaborador para não alterar sua rotina. Tais observações devem ser documentadas.

Dias 6 a 9 - Apontamentos de erros de processos.

Elaboração da documentação: por meio das observações e das investigações, o auditor deve produzir um documento no qual aponte os erros e as falhas dos processos.

Dias 10 a 12 – Apontamento de erros de produção do software.

Elaboração da documentação: por meio das observações e investigações, o auditor deve produzir um documento no qual aponte os erros e falhas do produto de software.

Dia 13 – Reunião.

Gerente de projetos e direção: trata-se de uma reunião de alinhamento, na qual são apresentados os relatórios gerados para o nível gerencial. Não possui um caráter técnico a nível de desenvolvimento, mas um olhar aos processos e produtos.

Dia 14 – Reunião.

Reunião de fechamento da auditoria e entrega do documento final: essa reunião tem um caráter mais técnico e nela são dadas as devolutivas de forma pontual.

Dessa forma, com as observações, pode-se prever um planejamento para que a empresa consiga se organizar para receber a auditoria.

NÃO PODE FALTAR

AUDITORIA DE SISTEMAS DA INFORMAÇÃO

Sergio Eduardo Nunes

Ver anotações

CONTROLES GERAIS DE AUDITORIA DE SISTEMAS

São as estruturas internas das organizações, as políticas administrativas operacionalizadas e os procedimentos utilizados em atividades operacionais nas empresas, ou seja, as políticas internas da empresa, que são um guia para o desenvolvimento do sistema.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Caro aluno, os processos de auditoria estão presentes na Engenharia de Software, a fim de agregar qualidade a processos e produtos. Mas, para que você possa compreender mais a fundo como são efetuadas as análises nos desenvolvimentos, faz-se necessário abrangermos as nossas discussões em torno dos assuntos relacionados à auditoria em sistemas da informação.

Para isso, inicialmente você compreenderá o que são controles gerais e onde são encontrados no dia a dia das organizações. Dessa forma, será possível compreender como os controles gerais são aplicados nos processos de auditoria de sistemas de informação. Os conceitos e aplicações discutidos acerca desse tema permitem ao profissional identificar características e peculiaridades sobre a companhia que podem influenciar nas funcionalidades do sistema, o que levará a uma auditoria mais assertiva.

Em seguida, você verá onde os controles de software de sistema estão localizados em razão do sistema e de que forma eles podem ser passíveis de análises nos processos de auditoria. Isso, em termos profissionais, é muito importante, pois permite ao profissional compreender as dependências funcionais entre software de sistema e a aplicação propriamente dita.

Na sequência, as discussões serão sobre os controles de aplicativos, e, por meio de conceituação e de exemplos, será possível compreender como a auditoria deve ser aplicada na análise de entrada, no processamento e na saída de dados. Trata-se de uma parte de grande relevância para seus estudos, já que nela são tratados os assuntos de auditoria voltados para o próprio desenvolvimento de software.

Finalmente, esses tópicos permitirão que você compreenda como de fato a auditoria é operacionalizada durante o ciclo de desenvolvimento de um sistema. Sendo assim, para aplicações profissionais, é possível

identificar as fases de auditoria e como elas devem receber as devidas tratativas.

No tocante à atuação profissional, os assuntos discutidos nesta seção são de extrema importância, pois, em grande parte das empresas de desenvolvimento de software, tais conhecimentos são utilizados tanto no dia a dia do desenvolvedor, quanto na participação e/ou envolvimento nos processos de auditoria de sistemas.

Caro aluno, você está conduzindo um processo de auditoria em sistema web como terceirizado para uma empresa de desenvolvimento de software. Até o momento, você já definiu o planejamento da auditoria, e os processos estão sendo cumpridos.

Pensando nisso, existe um planejamento que deve ocorrer a partir do dia 10, conforme pode ser observado a seguir.

Dias 10 a 12 – Apontamentos de erros de produção do software.

Elaboração da documentação: por meio das observações e investigações, o auditor deve produzir um documento no qual aponte os erros e falhas de produto de software.

Com isso, entre essas atividades estão os controles de aplicativos, os quais se tratam de relatórios que fazem as tratativas da auditoria, a análise da entrada de dados no sistema, o processamento de dados no sistema e as respectivas saídas geradas pelo sistema. Pode-se dizer que parte mais sensível até o momento está relacionada ao formulário de inserção de novos usuários, por meio do link “Novo Usuário”, na página de Usuários, conforme pode ser observado na Figura 4.6.

Figura 4.6 | Página do sistema da loja

Nome	E-mail	Senha	Excluir
Serginho	teste@teste.com	81dc9bdb52d04dc20036dbd8313ed055	
Testando	teste@teste.com	81dc9bdb52d04dc20036dbd8313ed055	

Dessa forma, a auditoria deve gerar um relatório, com textos e prints, que indique falha, erros ou inconsistências, permitindo, assim, que, ao gerar tais insights, a empresa de desenvolvimento possa fazer as correções necessárias.

Como você deve imaginar, há uma grande quantidade de conhecimentos importantes na área de auditoria de sistemas. Então vamos para mais uma leitura muito interessante acerca de Engenharia de Software.

CONCEITO-CHAVE

Caro aluno, cada organização apresenta algumas características, como costumes, clima organizacional, políticas internas, que fazem com que os colaboradores tenham determinado comportamento e que são parte do DNA de cada uma delas. Na área de desenvolvimento de software, esse ambiente de trabalho, também está presente. Muitas vezes, orientado por métodos, normas ou metodologias de desenvolvimento.

Para que os controles gerais de auditoria possam ser compreendidos, Braz (2017) define-os como as estruturas internas das organizações, as políticas administrativas operacionalizadas e os procedimentos utilizados nas atividades como um todo.

Na prática os controles gerais não são aplicáveis somente à tecnologia da informação. O controle geral é operacionalizado da portaria à direção da empresa, ou seja, todos os colaboradores criam o ambiente, o que ajuda a moldar o controle geral. Mas por que o controle geral interfere nos processos de auditoria?

Para responder ao questionamento, Braz (2017) afirma que, durante um processo de auditoria em que se tenha como objetivo a avaliação de um sistema na empresa (financeiro, contábil, fiscal, etc.), é necessário compreender como o controle geral atua sobre a aplicação. Por

exemplo, se o processo de auditoria ocorrerá sobre o sistema de vendas de uma loja de departamento, precisamos compreender alguns pontos, tais como:

- Um vendedor pode conceder desconto para o cliente?
- O gerente pode conceder desconto para o cliente?
- É possível alterar o prazo de pagamento?
- O número de parcelas no crediário pode ser aumentado pelos colaboradores?

Percebeu como o controle geral das atividades, as regras e os procedimentos estão diretamente ligados aos processos de auditoria? Por exemplo, imagine que somente o gerente tenha uma senha que permita aumentar o percentual de desconto no sistema de vendas do caixa. Mas, ao fazer o processo de auditoria, é feito um apontamento de vulnerabilidade, pois o sistema permite que qualquer colaborador possa inserir o desconto. Isso seria muito negativo para um processo de auditoria, pois seria apontada como vulnerabilidade uma funcionalidade do sistema.

ASSIMILE

Falhas e erros de software ocorrem em algumas situações e podem ter consequências negativas, sobretudo para empresas que utilizam a aplicação. O maior prejuízo é a visibilidade perante os clientes. Imagine que um aplicativo bancário, devido a uma falha, comece a fazer transferências aleatoriamente e deixe o saldo dos usuários zerados. Mesmo que o dinheiro retorne para conta posteriormente, haverá um impacto negativo quanto à visão do cliente.

Segundo Braz (2017), quando os controles gerais apresentam deficiência, ocorre a diminuição da confiabilidade nos controles individuais. Por esse motivo, o primeiro passo de um processo de

auditoria é a avaliação dos controles gerais para então promover as análises por meio de processos de auditoria a fim de avaliar as aplicações computacionais. No processo de auditoria em desenvolvimento de software, no que tange à avaliação do controle geral, ela é organizada em seis categorias, conforme pode ser observado no Quadro 4.4.

Quadro 4.4 | Categorias de controles gerais para processo de auditoria

CATEGORIA	CARACTERÍSTICA
Controle organizacional	São as políticas internas, os procedimentos e a organização estrutural utilizada na empresa. Isso determina quais são as atribuições e as responsabilidades dos colaboradores envolvidos nas atividades de desenvolvimento de software. Um exemplo prático é compreender como são organizados os times de desenvolvimento dentro da empresa. Existe um gerente de projetos responsável por cada time de desenvolvimento? Existe um gerente geral que faz as tratativas com cada gerente de projetos? Existem diversas possibilidades dentro das organizações e é necessário compreender essa estrutura no processo de auditoria.

CATEGORIA	CARACTERÍSTICA
Controle geral de segurança	<p>Este deve verificar se os controles gerais determinam que o sistema possua gerência de riscos e incidentes, quais são as políticas de segurança da empresa, se as funções relacionadas à segurança possuem um setor ou responsável pelo gerenciamento na empresa e se existe supervisão relacionada à segurança da informação.</p> <p>Como exemplo, imagine que uma empresa de móveis planejados possua um sistema para desenvolver um layout para o cliente. Esse sistema permite importar para o layout imagens baixadas pelo cliente. Porém, a política de segurança da empresa não permite que sejam utilizados pendrives de clientes no computador. Ou seja, uma política interna moldará como o sistema deve operar.</p>
Continuidade de serviço	<p>No processo de auditoria deve ser analisado se o controle geral possui tratativas e se, em caso de falhas, erros, bugs ou incidentes de segurança, existe alguma tratativa relacionada à recuperação parcial ou total do sistema.</p> <p>Por exemplo, um sistema de pagamento permite diversos meios, tais como: boleto, cartão de crédito, cartão de débito e PIX. Caso umas das formas falhe, o sistema vai continuar disponibilizando os outros meios de pagamento?</p>

CATEGORIA	CARACTERÍSTICA
Controle de software	O controle geral, neste ponto, tem um olhar de limitação e supervisão de acessos aos arquivos, diretórios e pontos críticos do sistema. Por exemplo, o sistema possui senha de acesso ao driver C:, onde os arquivos e diretórios estão instalados? Caso o sistema permita acesso aos arquivos, o usuário terá permissão de editar algum?
Controle de acesso	No processo de auditoria, deve haver a verificação de existência de recursos ou políticas administrativas que detectam acessos não autorizados para evitar incidentes de segurança e intrusões. Exemplo: o sistema deve emitir um alerta ao administrador se houver um alarme de intrusão no sistema. Dessa forma, no processo de auditoria, é preciso ocorrer uma verificação dessa funcionalidade.
Controle de versionamento	Verifica se existe um controle de modificações e implementações no sistema. Um exemplo é a verificação de utilização de algum software para gerenciamento e controle das versões desenvolvidas.

Fonte: adaptado de Braz (2017, p. 52).

Conforme se pode observar, as categorias de controles gerais para processo de auditoria apresentam diversas tratativas, as quais visam observar todas as características que possam, de alguma forma, interferir na auditoria.

Segundo Lyra (2008), o controle de software de sistema se trata de um conjunto de programas desenvolvidos para gerenciar, controlar e executar atividades de processamento de dados. Observe, no Quadro 4.5, exemplos de software de sistemas.

o

Quadro 4.5 | Softwares de sistemas

Software de sistema	Definição	Exemplo
Sistema operacional	Sistema responsável pelo funcionamento do computador e que faz a conexão do sistema com o hardware.	Windows, Linux, Android, IOS, entre outros.
Utilitários do sistema	Sistema responsável por fazer o gerenciamento dos recursos.	Gerenciador de dispositivos, gerenciador do sistema, sistema de gerenciamento de compartilhamento.
Sistemas de bibliotecas	São partes de programas que realizam determinadas funcionalidades.	Bootstrap (para desenvolver sites responsivos), PyGame (para desenvolver jogos em Python), Google Charts (para gerar gráficos em PHP).
Software de segurança	São sistemas que visam evitar incidentes de segurança	Firewall, Proxy, sistemas de senhas e autenticação de usuário.

Ver anotações

Software de sistema	Definição	Exemplo
Sistema de comunicação de dados	Sistemas que permitem a comunicação da aplicação com o usuário.	WAMP, XAMMP, LAMP, entre outros.
Sistema de gerenciamento de banco de dados	Trata-se de sistemas de banco de dados, podendo ser do tipo relacional e não relacional.	MySQL, Oracle, MongoDB, CouchDB, etc.

Fonte: elaborado pelo autor.

Como se pode observar no quadro, os sistemas estão muito presentes em muitas organizações, e boa parte deles é necessária para que a empresa faça suas operações.

Com isso, neste momento temos um olhar de aplicação profissional para os processos de auditoria no controle de software de sistema.

Assim, vamos tomar como exemplo um sistema de vendas para caixa de supermercado, no qual, inicialmente, temos que listar quais sistemas estão, de alguma forma, relacionados ao sistema do caixa, logo passíveis de análise nos processos de auditoria. A fim de entender o que contexto apresentado comprehende, observe quais sistemas são elegíveis para a auditoria:

- **Sistema operacional:** obrigatoriamente estará instalado nos computadores dos caixas e nos servidores.
- **Utilitários de segurança:** como ocorrem transações que envolvem quantidades, pagamentos e contabilizações, é necessária auditoria sobre esses serviços.

- **Sistema de comunicação:** a auditoria é necessária, pois, para que os caixas possam efetuar as transações, obrigatoriamente são necessários sistemas de gerenciamento de comunicação de dados.
- **Sistema de gerenciamento de banco de dados:** a auditoria deve dar especial atenção a este ponto pois, em um sistema de supermercado, as transações são feitas em cima de banco de dados.

Segundo Lyra (2008), nos processos de auditoria, existe uma preocupação voltada ao controle de software de sistema, que é relacionado:

- Ao acesso ao software de sistema.
- À supervisão do software de sistema.
- Ao controle de alteração do software de sistema.

Ainda de acordo com Lyra (2008), considera-se que esses três pontos discutidos sejam elementos críticos dentro do controle de software de sistema. Em termos práticos, os processos de auditoria devem ser orientadores (onde são necessários), como a verificação de documentação de desenvolvimento, os checklists de funcionalidades que garantam os pontos críticos, as observações e comprovações de funcionamento e eficácia, entre outras ferramentas de auditoria.

SAIBA MAIS

Nos controles gerais para processo de auditoria em desenvolvimento de sistema, existe uma divisão em seis categorias. Entre elas, há uma que trata exclusivamente de versionamento de desenvolvimento de software, a qual é uma parte extremamente importante do processo, principalmente quando as atividades de desenvolvimento são feitas em equipe.

Uma ferramenta muito eficiente para gerenciamento de versão são os conhecidos GITs, entre eles o mais utilizado é o GitHub (c2020).

CONTROLE DE APLICATIVOS

Caro aluno, grande parte dos softwares executam três funções: entrada, processamento e saída. Por exemplo, em um sistema de consulta bancária, no caixa eletrônico, o usuário escolhe a função de saque (entrada), o sistema consulta o saldo, faz as verificações, autoriza o saque do valor desejado (processamento) e, finalmente, o dinheiro é liberado no caixa (saída). Quanto aos processos de auditoria, isso é conhecido por **Controle de Aplicativos**.

Segundo Imoniana (2016), o controle de aplicativos pode ser definido como as funcionalidades que são executadas diretamente nos softwares, os quais têm as três funções básicas de qualquer aplicação, sendo elas: entrada, processamento e saída. A auditoria deve atestar que as três sejam confiáveis e que garantam a integridade dos dados. Para tal, vamos observar os processos de auditoria devem se comportar em cada uma das funções.

CONTROLE DE ENTRADA DE DADOS

Segundo Imoniana (2016), são desenvolvidos para garantir que os dados sejam inseridos na aplicação do tipo e da forma correta. Cabe aos controles de entradas terem mecanismos de entrada de dados correta, bem como evitarem que as transações ocorram de forma incompleta, duplicadas, com falhas e com outras anomalias.

Para você compreender melhor como um processo de auditoria pode atuar sobre o controle de aplicativos, vamos tomar como exemplo uma auditoria feita sobre a validação de campos de determinado formulário. Isso pode ser feito a nível de script ou ainda por meio de teste de uso, sendo mais comum que ocorra a nível de script. Para isso, observe um trecho de um script na Figura 4.7.

```
1 <div class="form-group">
2   <label class="col-sm-2 control-label">E-mail</label>
3     <div class="col-sm-5">
4       <input type="email" class="form-control" name="email"
5         required="">
6
7     <label class="col-sm-1 control-label">RG</label>
8     <div class="col-sm-4">
9       <input type="text" class="form-control" name="rg"
10      maxlength="12"
11      pattern="[0-9]{2}.[0-9]{3}.[0-9]{3}-[0-9]{1}$"
12      OnKeyPress="formatar('##.###.###-#', this)" required>
13     </div>
14   </div>
15
16   <div class="form-group">
17     <label class="col-sm-2 control-label">Celular</label>
18     <div class="col-sm-5">
19       <input type="phone" class="form-control" name="celular"
20         maxlength="13" placeholder="com whatsapp"
21         pattern="\\([0-9](\\d{2}\\))\\[0-9]d{5}-\\[0-9]d{4}$"
22         OnKeyPress="formatar('## #####-####', this)" required>
23     </div>
24
25     <label class="col-sm-1 control-label">Telefone</label>
26     <div class="col-sm-4">
27       <input type="phone" class="form-control" name="telefone"
28         maxlength="12" placeholder="não obrigatório"
29         pattern="\\([0-9](\\d{2}\\))\\[0-9]d{4}-\\[0-9]d{4}$"
30         OnKeyPress="formatar('## #####-####', this)">
31     </div>
32   </div>
```

Figura 4.7 | Exemplo de formulário em HTML (Bootstrap)

```

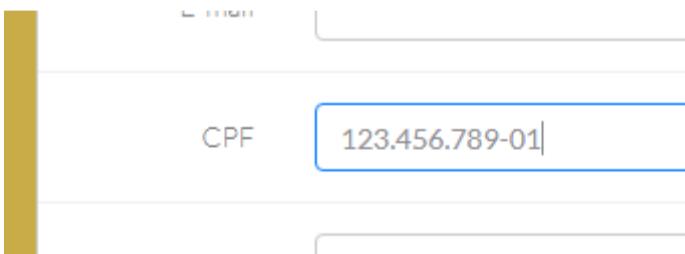
1 ▼ <div class="form-group">
2   <label class="col-sm-2 control-label">E-mail</label>
3   <div class="col-sm-5">
4     <input type="email" class="form-control" name="email" required="">
5   </div>
6
7   <label class="col-sm-1 control-label">RG</label>
8   <div class="col-sm-4">
9     <input type="text" class="form-control" name="rg" maxlength="12"
10    pattern="[0-9]{2}.[0-9]{3}.[0-9]{3}-[0-9]{1}$" OnKeyPress="formatar('##.###.##-#', this)" required>
11   </div>
12 </div>
13
14 ▼ <div class="form-group">
15   <label class="col-sm-2 control-label">Celular</label>
16   <div class="col-sm-5">
17     <input type="phone" class="form-control" name="celular" maxlength="13" placeholder="com whatsapp"
18     pattern="\([0-9](\d{2}\))\([0-9]d{5}-\[0-9]d{4}\$" OnKeyPress="formatar('## #####-####', this)" required>
19   </div>
20
21   <label class="col-sm-1 control-label">Telefone</label>
22   <div class="col-sm-4">
23     <input type="phone" class="form-control" name="telefone" maxlength="12" placeholder="não obrigatório"
24     pattern="\([0-9](\d{2}\))\([0-9]d{4}-\[0-9]d{4}\$" OnKeyPress="formatar('## ####-##', this)">
25   </div>
26 </div>

```

Fonte: captura de tela do Bootstrap elaborada pelo autor.

Nesse exemplo, nas linhas 4, 9/10, 17/18 e 23/24 ocorre o controle de entrada de dados, sendo este um ponto de verificação do controle de aplicativos, para o qual é utilizada uma técnica disponível na linguagem de marcação a fim de se validar a entrada do CPF, do RG, do celular e do telefone. Para isso foram aplicadas máscaras nas entradas. Por exemplo: se o usuário entrar com o CPF 123.456.789-01, o campo apresentará e fará a entrada do dado, conforme demonstrado na Figura 4.8.

Figura 4.8 | Validação de entrada de dados



Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

Reparou que, para efetuar a auditoria em software de sistemas, deve-se conhecer a tecnologia na qual determinada funcionalidade foi desenvolvida? Isso ocorre por se tratar de uma área de desenvolvimento de software, na qual são utilizadas linguagens de programação/marcção.

A validação de dados é algo muito importante na programação, pois garante que as entradas dos dados estejam no tipo e no formato corretos, o que garante uma execução exitosa da transação entre aplicação e banco de dados. Para isso, as linguagens de programação, como JavaScript, ou de marcação, como o HTML 5, possuem ferramentas que auxiliam o desenvolvedor nessa tarefa.

Ver anotações

Já quanto às tratativas de processamento de dados em controles de aplicativos, Imoniana (2016) define que o controle de processamento deve garantir que os dados de entrada sejam executados dentro do sistema, gerando, assim, saídas coerentes. Na prática, para que a análise de processamento ocorra, são necessários recursos auxiliares dentro do sistema.

Um exemplo de recursos auxiliares em sistemas de software são os logs. Segundo Imoniana (2016), os logs são históricos que ficam registrados em um repositório dentro do sistema. Para uma análise a nível de auditoria de sistemas, eles são grandes aliados não somente para verificação de processamento, mas também para diversas outras transações.

A nível de código, a auditoria de sistema de software, quanto ao processamento (segunda fase do controle de aplicativos), normalmente necessita do auxílio de um especialista, o qual deve conhecer bem a sintaxe de linguagem de programação, os seus métodos, funções, objetos e compatibilidade com outras tecnologias. Em termos práticos, as empresas de auditoria fazem a contratação do especialista para auditar os scripts.

Finalmente, o controle de saída de dados é definido por Imoniana (2016) como ferramenta de garantia de integridade de forma consistente a absoluta. No processo de auditoria, é preciso gerar relatórios de saídas de dados para que seja possível uma análise com relação à sua integridade e exatidão.

Em aplicações profissionais, essas análises são mais fáceis de serem executadas nos processos de auditoria. Por exemplo, imagine que um software de controle de vendas de móveis deva aplicar 5% de desconto nos pagamentos à vista. Os testes são de execução fácil, uma vez que basta escolher um produto qualquer e aplicar a opção pagamento à vista. Por meio da verificação do valor gerado, é possível fazer uma análise. Porém, para uma análise consistente, é necessário efetuar muitos testes com produtos e situações possíveis de serem executadas.

REFLITA

Os históricos de registros em sistemas, por meio de logs de uso de sistemas, podem fornecer detalhes importantes para permitir diversas análises de entrada, processamento e saída de um sistema. Porém, ao mesmo tempo, ficam monitorando as atividades que o usuário faz dentro do sistema.

Até que ponto os logs podem ser utilizados para monitoramento do sistema, sem que ocorra a invasão da privacidade dos usuários? Vale a pena essa reflexão.

Caro aluno, percebeu como a auditoria pode estar presente em muitos momentos do ciclo de vida do desenvolvimento de um software? Ao longo das discussões, percebemos que desde o momento do levantamento de requisitos, no início do projeto, até o encerramento/entrega do produto de software, os processos de auditoria podem estar presentes. Isso com o intuito de contribuir positivamente para que o software seja confiável e eficiente.

Em termos profissionais, as discussões desta seção, contribuirão para que você compreenda os períodos de auditoria durante o ciclo de vida de desenvolvimento e de que forma os controles de auditoria, software de sistemas e aplicativos são executados em sistemas da informação. Ainda, por meio dos exemplos práticos, você conseguiu um excelente

referencial para realizar suas próprias aplicações em outras situações dentro das organizações e nas atividades de desenvolvimento de software.

0

PESQUISE MAIS

No capítulo 4 do livro intitulado *Introdução a sistemas de informação* (RAINER Jr.; CEGIELSKY, 2016), disponível na Biblioteca Virtual, são tratados os assuntos relacionados à auditoria de software. Este é um referencial bibliográfico muito interessante para a compreensão dos processos relacionados à auditoria de software, por isso aproveite a leitura!

RAINER Jr. R. K.; CEGIELSKY, C. G. **Introdução a sistemas de informação**. 5. ed. Rio de Janeiro: Elsevier, 2016.

Ver anotações

FAÇA VALER A PENA

Questão 1

As empresas têm características e peculiaridades que determinam os seus controles gerais. Isso impacta diretamente em algumas funcionalidades dos sistemas. Assim, quando são necessários processos de auditoria, essas informações são de extrema importância.

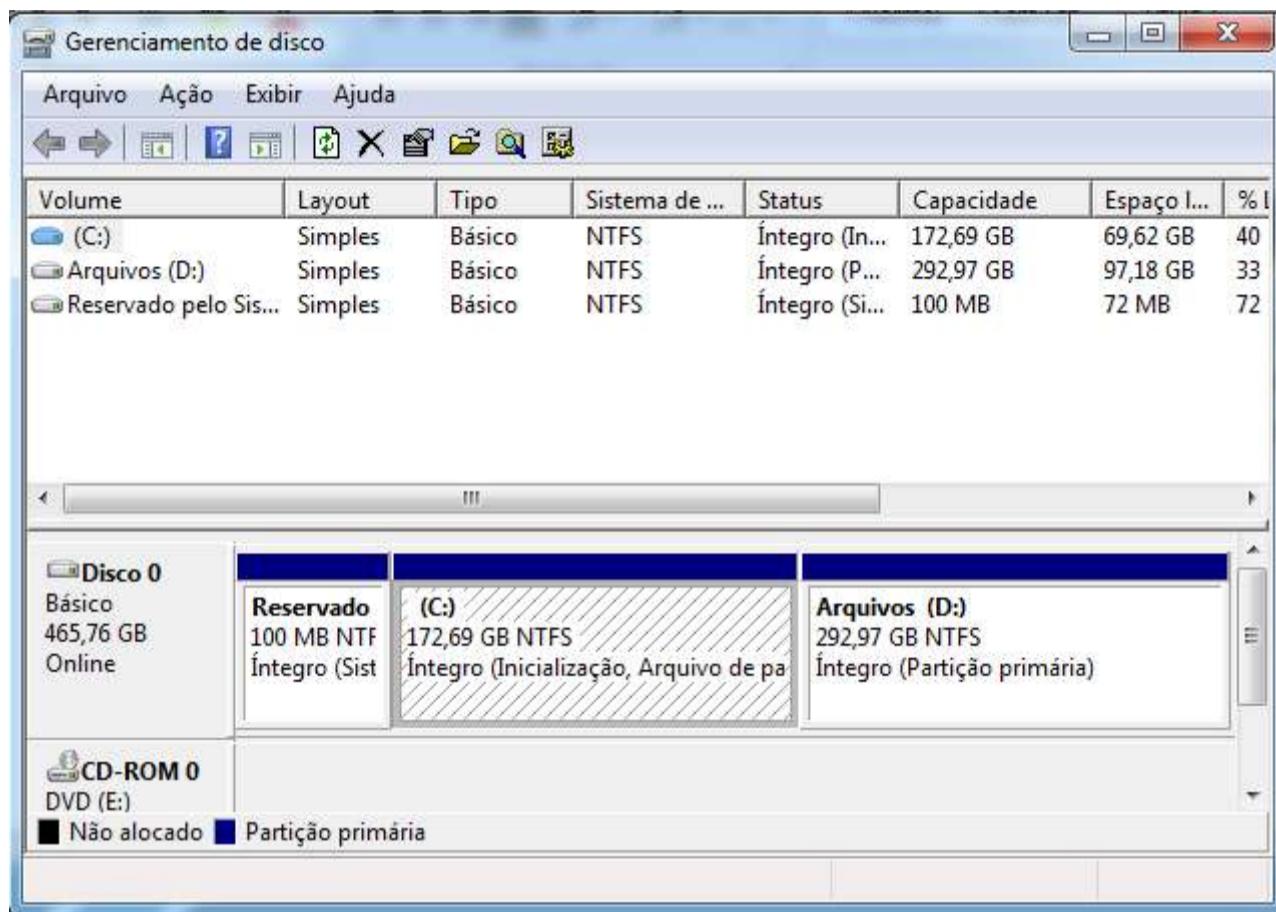
Se os controles gerais na auditoria em sistemas da informação apresentarem deficiências, assinale a alternativa com as consequências.

- a. Ocorre a diminuição da disponibilidade nos controles individuais.
- b. Ocorre a diminuição da disponibilidade nos controles coletivos.
- c. Ocorre a diminuição da confiabilidade nos controles individuais.
- d. Ocorre a diminuição da confiabilidade nos controles coletivos.
- e. Ocorre o aumento da disponibilidade nos controles coletivos.

Questão 2

Os sistemas operacionais apresentam gerenciadores que permitem a interação com o sistema operacional e que, para o desenvolvimento de sistemas e processos de auditoria, são conhecidos como controle de sistemas de software. Com base no exposto, analise as figuras a seguir:

I.



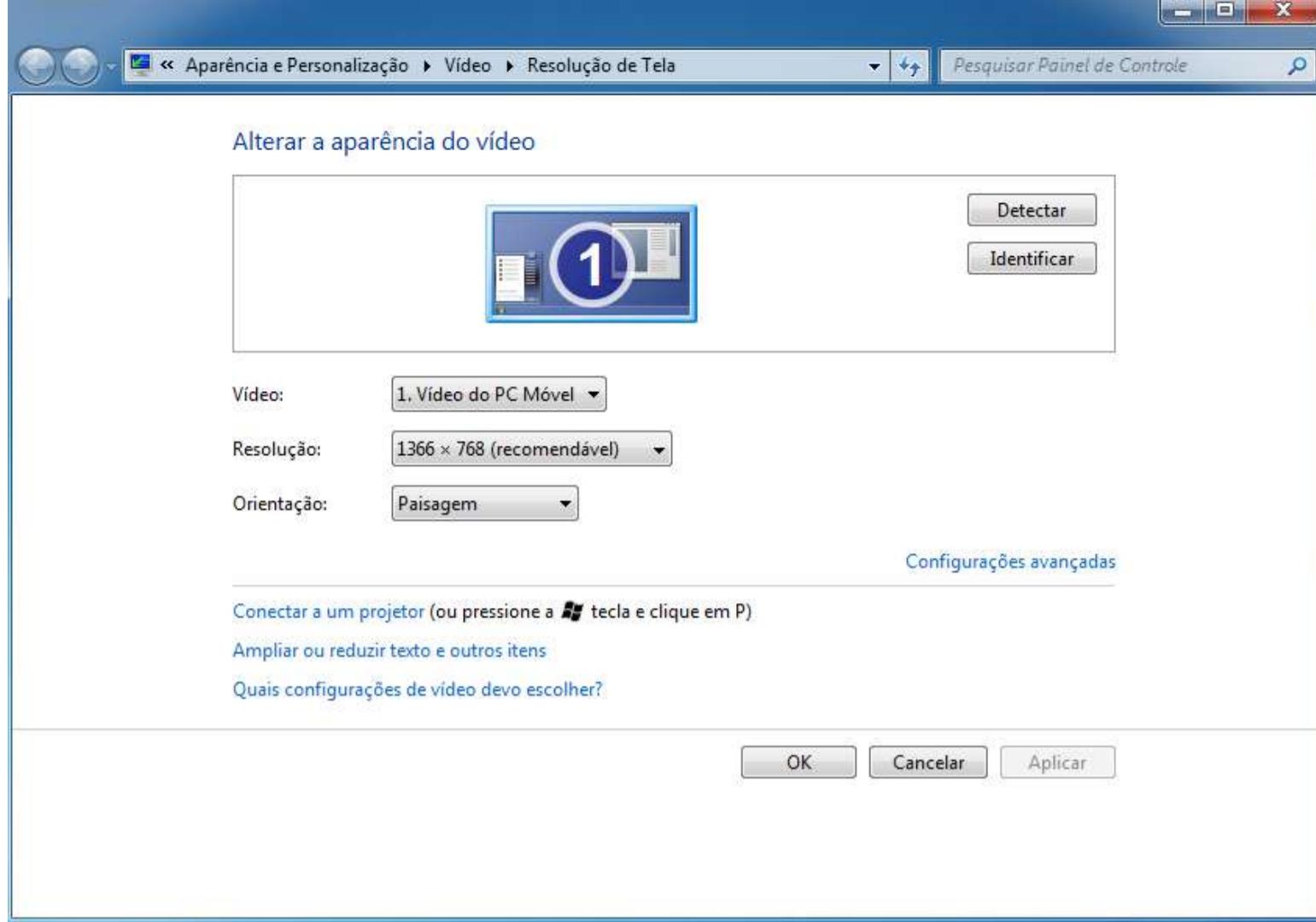
Fonte: captura de tela do Gerenciamento de disco elaborada pelo autor.

II.



Fonte: captura de tela do prompt de comandos elaborada pelo autor.

III.



Fonte: captura de tela do Painel de Controle do Sistema Operacional.

Assinale a alternativa que descreva corretamente quais figuras são correspondentes ao controle de software de sistemas.

- a. Apenas a figura I.
- b. Apenas a figura II.
- c. Apenas a figura III.
- d. Apenas as figuras I e II.
- e. Apenas as figuras I e III.

Questão 3

O controle de aplicativos está diretamente relacionado às atividades de desenvolvimento de software uma vez que a sua análise ocorre em função das atividades que a aplicação se propõe a resolver.

A partir do apresentado, analise as asserções a seguir e a relação proposta entre elas.

I. Telas de login podem ser enquadradas como controle de aplicativos.

POIS

II. As suas entradas geram um processamento e uma saída.

A seguir, assinale a alternativa correta:

- a. As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- b. A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa.
- c. As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- d. A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- e. As asserções I e II são proposições falsas.

REFERÊNCIAS

BRAZ, M. R. **Auditória de TI**: o guia da sobrevivência. Brasília: Asè Editorial, 2017.

GITHUB. **GitHub**, [S./], c2020. Disponível em: <https://github.com/>. Acesso em: 21 nov. 2020.

IMONIANA, J. O. **Auditória de Sistemas de Informação**. 3. ed. São Paulo: Atlas, 2016.

LYRA, M. R. **Segurança e Auditoria em Sistemas de Informação**. Rio de Janeiro: Editora Ciência Moderna, 2008.

RAINER Jr. R. K.; CEGIELSKY, C. G. **Introdução a sistemas de informação**. 5. ed. Rio de Janeiro: Elsevier, 2016. Disponível em: <https://cutt.ly/SjOoalH>. Acesso em: 16 nov. 2020.

FOCO NO MERCADO DE TRABALHO

AUDITORIA DE SISTEMAS DA INFORMAÇÃO

Sergio Eduardo Nunes

Ver anotações

RELATÓRIO DE CONTROLE DE APLICATIVOS

Por meio das observações e investigações, o auditor deve produzir um documento no qual aponte os erros e falhas de produto de software. Nele devem estar as tratativas da auditoria, a análise da entrada de dados no sistema, o processamento de dados no sistema e as respectivas saídas geradas pelo sistema.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

SEM MEDO DE ERRAR

Os desenvolvimentos web surgiram como uma alternativa viável que flexibiliza a forma como podem ser acessados. E, assim como qualquer outro software produzido para servidor local, as aplicações web

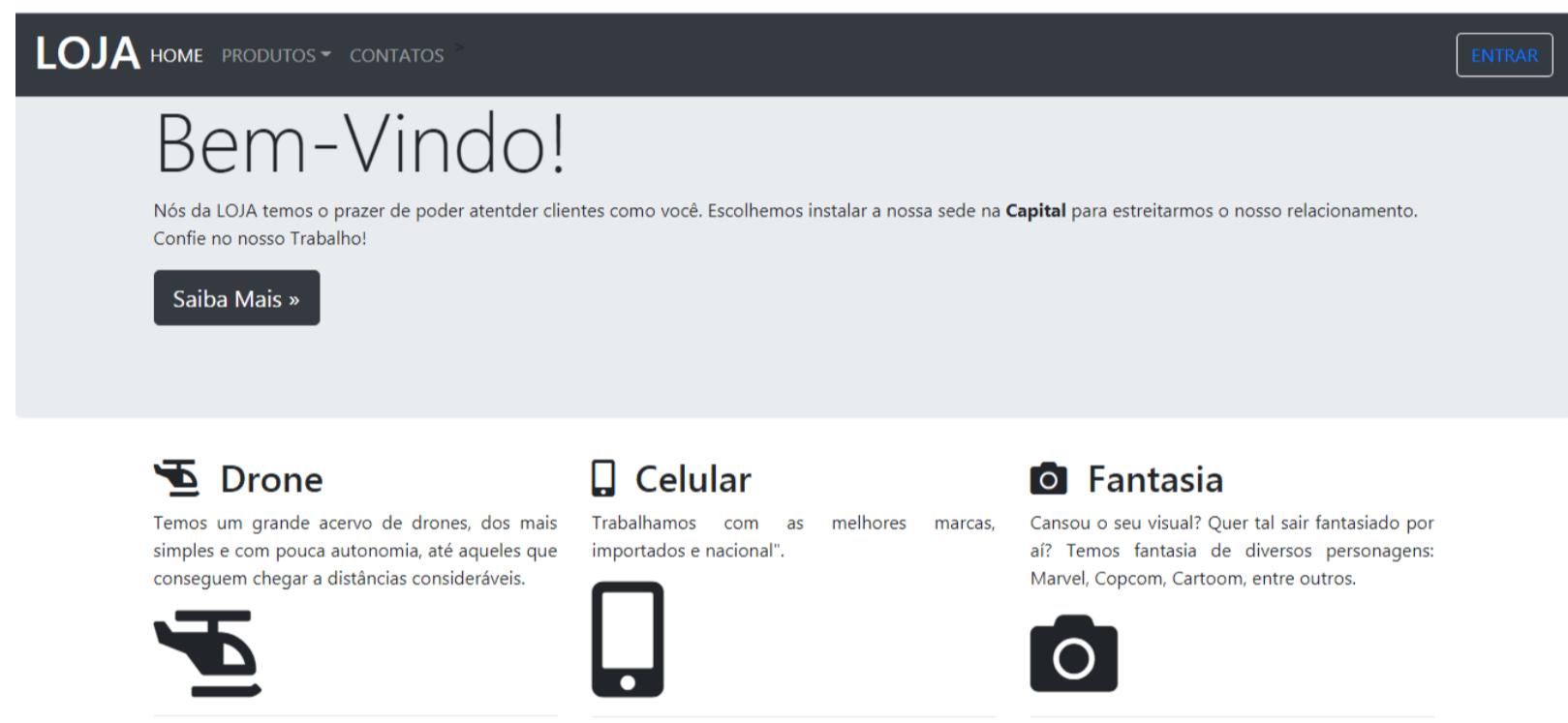
também estão suscetíveis a falhas de desenvolvimento.

Para garantir a qualidade do produto de software, a empresa contratou os seus serviços de auditoria. Nessa fase dos processos de auditoria, está planejada uma análise de produto de software, sendo que deve ser gerado especificamente um relatório quanto aos controles de aplicativos (entrada, processamento e saída de dados) do formulário de inserção de novos usuários.

Para isso, inicialmente o site da loja deve ser acessado no navegador, por meio do endereço: localhost/loja. A página aberta está demonstrada na Figura 4.5.

Ver anotações

Figura 4.5 | Página home do site do sistema da loja



Fonte: captura de tela do site LOJA elaborada pelo autor.

Repare que no canto superior direito existe um botão escrito ENTRAR. É clicando nesse botão que se pode acessar o sistema, conforme pode ser observado na Figura 4.6.

Figura 4.6 | Página do sistema da loja

USUÁRIOS			
Novo Usuário			
Nome	E-mail	Senha	Excluir
Serginho	teste@teste.com	81dc9bdb52d04dc20036dbd8313ed055	
Testando	teste@teste.com	81dc9bdb52d04dc20036dbd8313ed055	

Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

Com isso, basta clicar no link “Novo Usuário” para acessar o formulário passível de auditoria.

Relatório de controle de aplicativos

Inicialmente, a análise se refere ao visual dos campos do formulário conforme pode ser observado na Figura 4.9.

Figura 4.9 | Interface de inserção de usuários

A captura de tela mostra uma interface web com uma barra superior cinza contendo links para 'Usuários', 'Clientes' e 'Produtos'. Abaixo, há três campos de texto rotulados: 'Name:' (Nome), 'E-mail:' (E-mail) e 'Senha:' (Senha). Cada campo tem uma placeholder correspondente ('Nome completo', 'Somente números'). Um botão 'SALVAR' está posicionado no lado direito da interface.

Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

O formulário apresenta apontamentos em seus campos:

1. Não existe um indicativo no campo do e-mail sobre qual o formato de entrada do campo. Exemplo: usuário@provedor.com.
2. O formulário não possui o botão CANCELAR caso não se deseje mais cadastrar um novo usuário (apenas uma contribuição na auditoria, pois não se trata de um controle de aplicativo).

Ainda na análise de entrada de dados:

1. Campo de Nome: permite a entrada de letras e números. Esse campo deve permitir apenas letras.
2. Campo do e-mail: está no formato de entrada incorreto, pois está permitindo entrar com e-mail sem o formato padrão (nome@provedor.com).
3. Campo senha: embora exija somente números na descrição, o campo está permitindo a entrada de letras. Não existe um limitador

mínimo e máximo no campo.

Quanto ao processamento, embora existam alguns erros de entradas, os dados são processados corretamente e geram as saídas condizentes com as entradas, conforme pode ser observado na Figura 4.10.

Figura 4.10 | Interface de lista de usuários

A captura de tela mostra uma interface web com uma barra superior cinza contendo links para 'Usuários', 'Clientes' e 'Produtos'. O link 'Usuários' é o ativo. Abaixo, o título 'USUÁRIOS' está centralizado em uma barra marrom. À esquerda, uma barra azul contém o link 'Novo Usuário'. A tabela contém quatro linhas de dados:

Nome	E-mail	Senha	Excluir
Teste 123	teste@teste.com	0b4e7a0e5fe84ad35fb5f95b9ceeac79	
Segundo Teste	teste@segundo	e10adc3949ba59abbe56e057f20f883e	

Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

Ainda que no campo “senha” seja exibida a senha criptografada, assim mesmo foi permitido o processamento do dado em que:

- No usuário Teste 123, foi utilizada a senha aaaaaaa.
- No usuário Segundo Teste, foi utilizada a senha 123456.

Embora não seja o objetivo da presente auditoria propor soluções, ao ajustar as entradas de dados do controle de aplicativos, o formulário poderá ser ajustado e, consequentemente, melhorado.

NÃO PODE FALTAR

MANUTENÇÃO E EVOLUÇÃO DE SOFTWARE

Sergio Eduardo Nunes

Ver anotações

NECESSIDADE DE EVOLUÇÃO E DE MANUTENÇÃO DE UM SOFTWARE

Os processos evolutivos e de manutenção acompanham o ciclo de vida do sistema, e, há diferentes intenções para evoluir um sistema que não apenas correção de falhas, bugs, erros e inconformidades, como adaptá-lo a novos requisitos e a inserção de novas funcionalidades.



Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

Caro aluno, as atividades de evolução e manutenção de software possuem uma grande aplicação prática no cotidiano do profissional que atua na área de desenvolvimento de software. Com isso, a compreensão de suas características e peculiaridades é mais do que necessária, é um conhecimento obrigatório nessa área da tecnologia da informação.

Com base no exposto, num primeiro momento, serão apresentadas discussões acerca da necessidade de evolução e de manutenção dos softwares, o que lhe permitirá entender o processo de envelhecimento deles e as motivações para promover adequações aos novos requisitos.

Na sequência serão abordados a classificação e os tipos de atividades de manutenção de software, conhecimento de vital importância para que as manutenções que acompanham o ciclo de vida do software sejam feitas de forma mais assertiva.

O terceiro tópico em discussão promoverá o entendimento dos processos e das ferramentas de manutenção de software, os quais devem ser utilizados em prol da evolução do software a fim de garantir que implementações, adequações e ajustes agreguem a qualidade esperada, segundo o requisito do sistema.

Finalmente, para encerrar a seção de aprendizagem, será explanado como a engenharia reversa e a reengenharia são utilizadas nos processos de manutenção e evolução dos sistemas. Elas são ferramentas importantes e muito utilizadas por desenvolvedores cotidianamente.

Em suma, esta seção tem como objetivo proporcionar a base para um bom crescimento profissional, uma vez que os assuntos tratados aqui são altamente aplicáveis no dia a dia. Portanto, a compreensão desses tópicos será um diferencial técnico dentro da Engenharia de Software.

Caro aluno, você conduziu um processo de auditoria como profissional terceirizado para auxiliar determinada empresa em um desenvolvimento web. O seu trabalho foi tão bem executado que a companhia entrou em contato novamente com você para mais uma

consultoria. Como você já conhece as características do sistema de controle de usuários que foi desenvolvido, os gestores acreditam que a sua contratação trará bons resultados ao projeto.

Os gerentes de projetos entendem que é o momento de o software evoluir, ganhar novas funcionalidades, interface e demais benefícios que possam, de alguma forma, promover a evolução do sistema. Essas sugestões devem estar alinhadas às novas necessidades e tendências de mercado para que de fato ocorra essa evolução.

As suas sugestões, alinhadas com os aspectos da manutenção evolutiva, devem, por meio de um relatório, propor a evolução do software. A consultoria está limitada a duas páginas, conforme podem ser observadas nas figuras a seguir:

Figura 4.6 | Página do sistema da loja

USUÁRIOS			
Nome	E-mail	Senha	Excluir
Serginho	teste@teste.com	81dc9bdb52d04dc20036dbd8313ed055	
Testando	teste@teste.com	81dc9bdb52d04dc20036dbd8313ed055	

Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

Figura 4.9 | Interface de inserção de usuários

Nome:

E-mail:

Senha:

SALVAR

Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

A partir do apresentado, desenvolva um relatório para consultoria com base nos conceitos e aplicações da manutenção evolutiva de software, tanto para o front end quanto para as funcionalidades. A empresa está muito empolgada para o evoluir a aplicação, porém o sucesso do projeto dependerá de suas habilidades e conhecimentos.

Com assuntos tão pertinentes, em termos profissionais, para o dia a dia do desenvolvedor de sistemas, acredito que você deva estar curioso para mais discussões no âmbito da Engenharia de Software. Vamos adquirir mais habilidades e conhecimentos?

CONCEITO-CHAVE

Após certo tempo de uso, os sistemas operacionais deixam de receber suporte das empresas e isso significa que não serão mais disponibilizadas correções e atualizações para eles. Já os sistemas operacionais livres e de código aberto não são descontinuados, pois anualmente a comunidade evolui o sistema, efetua correções e insere novas funcionalidades.

Percebeu como existem tratativas diferentes para o envelhecimento do software? Os processos evolutivos e de manutenção acompanham o ciclo de vida do sistema, e, em alguns casos, ao atingir o seu limite, ele é descontinuado. Com base no exposto, nesta seção de aprendizagem, você conhecerá a necessidade de evolução e de manutenção de um software, a classificação e os tipos de atividades de manutenção normalmente executadas. Ainda será possível entender como são utilizadas as ferramentas dentro dos processos de manutenção, e ao final como a engenharia reversa e a reengenharia podem contribuir com a manutenção e os processos evolutivos dos softwares.

Antes de iniciarmos tais discussões, compreendamos qual a motivação para estudar esses tópicos. Imagine que seja desenvolvido um aplicativo para rastreamento de PETs com chip de localização. No seu lançamento, existiam poucos clientes, somente método de pagamento via cartão

(crédito e débito). Com o passar do tempo, o número de clientes aumentou consideravelmente, surgiram novos métodos de pagamento, como o PIX, e novas tecnologias de rastreamentos.

Percebeu como a evolução de outras tecnologias impactou diretamente na aplicação? Para compreender melhor os processos de evolução e de manutenção, entendamos, inicialmente, como ocorre o processo de envelhecimento de um software.

Segundo Vetorazzo (2018), os softwares são sequências lógicas de algoritmos cujo intuito é atender aos objetivos estabelecidos, os quais estão suscetíveis a mudanças de requisitos e ao ambiente que está sendo operacionalizado. Esse processo de envelhecimento é inevitável e exige uma análise de causas, de forma a fazer sua evolução e/ou manutenção, garantindo, assim, sua continuidade.

Vetorazzo (2018) define ainda que existem dois tipos de envelhecimento de software, conforme pode ser observado a seguir:

1. Falha de adequação: ocorre quando a equipe responsável pela evolução do software comete erros e falhas na adequação ou na implementação dos requisitos, ocasionando muitas vezes a perda da integridade e da confiabilidade da aplicação.

Um exemplo prático: um software para conversão de formatos de vídeo, que funcionava em uma versão do sistema operacional, não tem compatibilidade com a versão mais recente desse mesmo sistema operacional.

Caso o software seja utilizado por um usuário comum, certamente ele procurará outra solução. Mas, se uma empresa utiliza esse software em suas operações e em determinado momento os computadores são trocados e vêm com um sistema operacional não compatível com o software de conversão de vídeo, isso será um grande problema.

2. Falha na mudança: é quando existe alguma atualização, manutenção ou implementação que impacta negativamente outras

funcionalidades que já estavam em pleno funcionamento.

Um exemplo prático: um e-commerce possui apenas um gerador de boleto em funcionamento, mas, devido a solicitações dos usuários, precisará apresentar outras formas de pagamento. Para isso, foi implementado (de forma incorreta) uma API de módulo de pagamentos. O impacto desse erro foi o sistema gerador de boletos e o módulo de pagamentos diversos não funcionarem. Isso ocorreu devido ao desconhecimento da estrutura do gerador de boletos.

o

Ver anotações

Caro aluno, percebeu a necessidade de evolução e manutenção dos softwares? São muitas variáveis a serem observadas: evolução dos sistemas operacionais, modos de consumo, novos meios e métodos de pagamento, segurança e diversos outros pontos. Mas, como prever a expectativa de envelhecimento do software? Isso só ocorreria se tivéssemos informações de todas as empresas de tecnologia da informação, as quais poderiam, por meio de uma alteração, impactar no funcionamento do sistema de alguma forma, o que é impossível.

Porém, a compreensão da classificação e dos tipos de atividades de manutenção de softwares trará uma vantagem de recuperabilidade nas atividades de ajustes, manutenções, evoluções e adequações do sistema em casos de inconformidades.

Com isso, Pressman e Maxim (2016) defendem que há diferentes intenções para evoluir um sistema que não apenas correção de falhas, bugs, erros e inconformidades. Para compreender como são classificadas as atividades de manutenção, observe o Quadro 4.6.

Quadro 4.6 | Classificação e tipos de manutenção de software

CLASSIFICAÇÃO	CONCEITO	APLICAÇÃO
Adaptativa	São modificações necessárias para estar de acordo com novos requisitos, os quais podem ser provenientes de leis, regras, ameaças, meios ou métodos novos.	Recentemente (no ano de 2020) o Banco Central autorizou as instituições financeiras a utilizarem o PIX como método de pagamento. Isso exigiu uma adaptação do sistema de internet banking para que essa nova funcionalidade estivesse disponível aos clientes.

Ver anotações

CLASSIFICAÇÃO	CONCEITO	APLICAÇÃO
Corretiva	Sua função é corrigir falhas ou qualquer outro aspecto que seja motivo de degradação dos serviços do software. Além disso, a manutenção corretiva pode ocorrer antes, nas fases de desenvolvimento, ou depois, com o software já em funcionamento.	Nas eleições municipais de 2020, para que as pessoas que não foram votar pudessem justificar o voto, o governo federal disponibilizou um aplicativo para mobile conhecido como e-título. No primeiro turno, ele apresentou problemas desde a sua instalação até o processo de realizar a justificativa. A única funcionalidade em conformidade era a consulta da situação do eleitor quanto à Justiça Eleitoral. No segundo turno, o aplicativo teve de passar por uma manutenção corretiva para que fossem efetuados os ajustes necessários.

CLASSIFICAÇÃO	CONCEITO	APLICAÇÃO
Evolutiva	A manutenção evolutiva tem o objetivo de inserir novas funcionalidades no sistema.	Os chats eram um recurso bastante presente no e-commerce para atendimento aos clientes. Uma evolução desse tipo de atendimento é, em vez de ter um colaborador de plantão para atendimento no chat, utilizar atendimento virtual. Para isso, foram desenvolvidos algoritmos que utilizam inteligência artificial, os quais, com a evolução tecnológica, conseguem responder grande parte das dúvidas dos clientes.

Fonte: adaptado de Pressman e Maxim (2016, p. 798)

O conhecimento da classificação dos tipos de manutenção, em termos profissionais, permite que tanto um desenvolvedor quanto um gestor se posicione quanto às reais necessidades dentro da estrutura do sistema, facilitando o direcionamento de recursos dentro do ciclo de vida de desenvolvimento de software. Para tal, é necessário conhecer os processos e as ferramentas utilizados na manutenção de software, os quais são recursos de extrema importância para que, de fato, os processos sejam otimizados.

Segundo Pressman e Maxim (2016), os processos e ferramentas utilizados na manutenção de softwares têm como objetivo obter métodos que sejam utilizados como boas práticas e garantir conformidade aos requisitos.

Para entender melhor esse tópico, observe algumas das técnicas e ferramentas mais utilizadas a seguir.

CODIFICAÇÃO

É tida como uma parte muito importante na manutenção. A qualidade do código de programação deve possuir legibilidade, ou seja, deve ser fácil e legível. Ainda que as técnicas de indentação e os comentários de código devam estar presentes nas principais linhas da escrita do sistema e as suas funcionalidades. Observe na figura a seguir uma forma de utilizar a codificação com boas práticas.

Figura 4.11 | Exemplo de codificação

```
194 <script type="text/javascript">
195     function limpa_formulario_cep() {
196         //Limpa valores do formulário de cep (rua, bairro, cidade e estado).
197         document.getElementById('rua').value="";
198         document.getElementById('bairro').value="";
199         document.getElementById('cidade').value="";
200         document.getElementById('estado').value="";
201     }
```

Fonte: captura de tela do Sublime elaborada pelo autor.

O trecho de código foi escrito em JavaScript (não sendo necessário saber programar nessa linguagem). Mas repare em algumas boas práticas, que podem ser observadas:

- **Nomes:** os nomes de variáveis devem sempre remeter ao que será usado, exemplo: rua, bairro, cidade e estado (em amarelo nas linhas 197, 198, 199 e 200); outra boa prática é nomear as funções a serem realizadas. Por exemplo: na linha 195 existe uma função (function) para deixar os campos sem nenhum valor dentro, a qual é chamada de limpa_formulario_cep.

- **Comentário:** essa é uma técnica muito importante, mas deixada de lado por alguns desenvolvedores quando as práticas do desenvolvimento não impõem mais dificuldades, o que acaba complicando a execução da manutenção efetuada por outros desenvolvedores. Um exemplo pode ser observado na Figura 4.11, pois, na linha 196, é explicado o que a função irá executar naquele trecho de código.
- **Indentação:** são espaços utilizados para demonstrar o nível hierárquico das funcionalidades dentro do algoritmo. Ainda utilizando como base a Figura 4.11, repare que na linha 195 existe uma estrutura de função em que se abre uma chave, a qual é fechada na linha 201. Para a indentação, as linhas de 169 a 200 sofreram um recuo por meio de um TAB no teclado. Isso organiza o código-fonte e facilita o processo de manutenção.

■ VERSIONAMENTO

São documentações que determinam as modificações e as atualizações dos softwares. Elas podem ser feitas por dois meios:

- **Numeração:** trata-se de um sistema que por meio de uma numeração demonstra a sua versão. O objetivo é especificar as suas características por meio desse sistema numérico. Para um exemplo vamos supor que um determinado software esteja na versão 10.2.4.3 e que cada um desses números tenha um significado, conforme o que se explica adiante:
 - **10:** indica que houve dez mudanças significativas no sistema. Essa numeração muda cada vez que o software faz uma evolução que promova mudanças de grande escala.
 - **2:** esse número indica que foram adicionadas novas funcionalidades no sistema. Por exemplo, um software possui, na nova versão, a funcionalidade de impressão.

- **4:** esse número indica a quantidade de correções de bugs e falhas. Por exemplo, existia uma falha de envio de confirmação de inserção de produtos no carrinho de compras em um e-commerce na versão 10.2.3.3, que foi corrigido na versão 10.2.4.3.
 - **3:** são correções graves relacionadas a incidentes de segurança. Por exemplo, determinado aplicativo expunha os dados dos usuários na versão 10.2.4.2, e a falha foi corrigida na versão 10.4.2.3.
- **Comentário:** uma prática muito comum é adicionar, nas primeiras linhas do arquivo no qual foram feitas as manutenções, informações como: data, objetivo da manutenção, modificações efetuadas e demais comentários úteis para manutenções futuras.

SAIBA MAIS

Os versionamentos em atividades de desenvolvimento de sistemas são essenciais para que se possa gerenciar as mudanças que ocorrem no ciclo de vida do desenvolvimento do software. Para isso, existem softwares que auxilia os profissionais de desenvolvimento. Entre eles, está o GitHub que faz uma tratativa para o versionamento de forma simples e muito eficiente.

Para isso, a página do Git disponibiliza um informativo de como deve ser tratada as versões do GitHub.

GITHUB. 1.1 Começando - Sobre Controle de Versão, [S.I.]
c2021.

| ESTRUTURAR O CÓDIGO PARA EVOLUÇÃO

A atividade de planejamento requer uma reflexão dos limites do software. Dessa forma, permite projetar o código para permitir sua evolução, seja ela por meio de adaptações, mudanças, ou por qualquer outro meio, o importante é que sua estrutura permita a evolução.

O processo de manutenção é muito delicado e requer muita atenção, pois, em alguns casos, existe uma alteração nos algoritmos que impacta toda a lógica pensada para o desenvolvimento. Um código-fonte com severas falhas de indentação pode atrapalhar, e muito, a manutenção do sistema.

Na linguagem de programação Python, a indentação é obrigatória para que os programas funcionem, pois, do contrário, ao serem compilados, será gerada uma mensagem de erro de indentação. Aprender Python faz com que o desenvolvedor leve para outras linguagens a boa prática de indentar os scripts corretamente.

Caro aluno, as discussões e os exemplos acerca dos processos e das ferramentas de manutenção de software em aplicações profissionais estão no cotidiano das práticas de desenvolvimento de sistemas e são essenciais para que se possam fazer manutenções corretamente e para que seja realizado um trabalho a ser continuado por outros profissionais com habilidades técnicas.

Além disso, em termos profissionais, as discussões acerca da classificação e dos tipos de atividades de manutenção podem levá-lo a compreender os momentos nos quais as manutenções adaptativas (ajustar a novos requisitos), corretivas (correções de bugs e falhas) e evolutivas (adicionar novas funcionalidades) são operacionalizadas, em ambiente de desenvolvimento de sistemas, bem como as suas finalidades.

REENGENHARIA DE SOFTWARE

Segundo Pádua (2019), o processo de reengenharia de software é uma forma de reorganizar e/ou modificar o sistema a fim de fazê-lo apresentar um desempenho aceitável. Alguns fatores como falta de

evolução de software ou excesso de contínuas mudanças (pior ainda quando promovidas por equipes diferentes) acabam degradando os serviços do sistema.

Nesse momento, tentar encontrar novas soluções em um sistema comprometido pode não gerar o resultado desejado e ainda comprometer muitos recursos. Dessa forma, fazer uma reconstrução com a correção de erros e falhas, além de adequar as evoluções necessárias é uma ótima saída em busca de soluções.

Ainda de acordo com Pádua (2019), a reengenharia de software tem o objetivo de reimplementar sistemas legados, em que se busca:

- **Melhorar a manutenção:** ao se reestruturar o sistema, as futuras manutenções serão mais fáceis para efetuar a manutenção, visto que os antigos erros devem ser corrigidos na nova versão.
- **Redocumentar o software:** quando o software é reestruturado, a documentação é feita, permitindo, assim, que novas informações sejam colocadas nos scripts.
- **Reestruturar o sistema:** as estruturas que funcionavam à base de reparos e correções podem ser repensadas, o que permitirá a construção de um sistema com otimizações e atendimento aos requisitos.

Quando o software é produzido, existem funcionalidades e módulos que requerem grandes esforços para serem desenvolvidos, e, embora o sistema passe por testes, ao longo do tempo ele pode apresentar falhas. Porém, no processo de reengenharia de software, se o entendermos como uma releitura dos algoritmos, podemos dizer que a chance de erro será menor. Com base nisso, Pádua (2019) defende que os processos de reengenharia possuem riscos reduzidos, pois alguns problemas já haviam sido tratados.

Esses processos são apoiados em metodologias de operacionalização.

Para tal, Pressman e Maxim (2016) defende o modelo apresentado na

Figura 4.12.

Figura 4.12 | Modelo de reengenharia



Fonte: adaptada de Pressman e Maxim (2016, p. 802).

Observe que, ao longo dessa discussão, esses processos foram explorados e exemplificados. Porém, um novo termo muito importante quanto à manutenção e à evolução dos softwares foi citado no modelo representado na Figura 4.12: **engenharia reversa**.

EXEMPLIFICANDO

A área de desenvolvimento de software é muito colaborativa, permitindo, às vezes, encontrar funcionalidades inteiras já em funcionamento. No entanto, para ser utilizada em um sistema em funcionamento, sempre são necessárias algumas adequações.

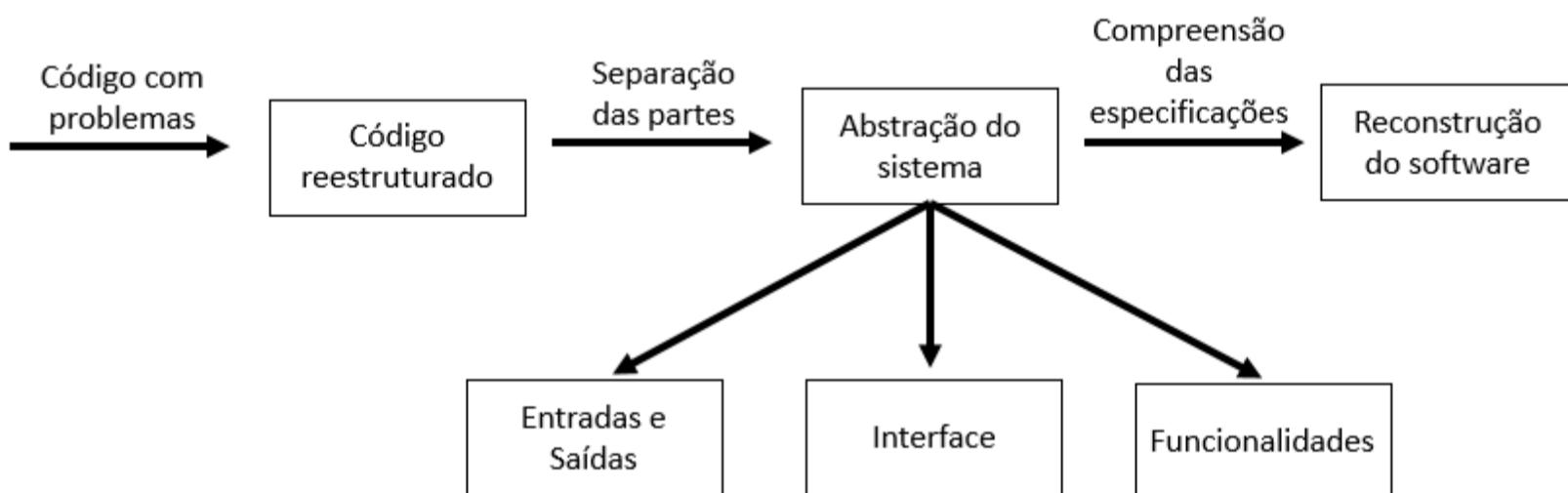
Para isso, é preciso fazer uma engenharia reversa, de modo que, após a compreensão de sua estrutura, seja possível promover as adequações necessárias.

Conforme defende Pádua (2019), a engenharia reversa é o processo de reconstrução do software que parte do princípio de recuperação do código para tornar compreensíveis suas funcionalidades. Assim, elas podem ser reescritas com vistas a serem otimizadas e corrigidas.

Claramente, quando a engenharia reversa é utilizada em hardware, a técnica fica visualmente mais fácil de ser compreendida, pois imaginamos os componentes sendo desmontados, o que nos permite compreender a ordem de montagem e, ainda, analisar cada componente em separado a fim de entender o seu objetivo e funcionamento.

Mas como a engenharia reversa é tratada a nível de software? Antes de responder esse questionamento, observe a Figura 4.13.

Figura 4.13 | Processo de engenharia reversa



Fonte: adaptada de Pressman e Maxim (2016, p. 803).

Observe que gradativamente os métodos de engenharia reversa tratam de reestruturar o código, permitindo, assim, a compreensão das funcionalidades (entradas e saídas, interface e funcionalidades).

Segundo Pádua (2019), ao se utilizar os processos e técnicas de engenharia reversa, é possível visualizar o software de diferentes maneiras, tais como:

- **Nível de implementação:** permite a compreensão das características e especificidades da linguagem de programação

utilizada no processo de implementação.

- **Nível estrutural:** permite a compreensão dos diferentes módulos e funcionalidades e das suas respectivas dependências funcionais. Essa abstração se dá por meio da análise da estrutura da linguagem de programação utilizada.
- **Nível funcional:** permite que as partes que compõem os sistemas sejam compreendidas; com ênfase na lógica utilizada no desenvolvimento.
- **Nível de domínio:** permite compreender onde o software é utilizado.

Com isso, em termos profissionais, tanto a reengenharia quanto a engenharia reversa possuem técnicas relativamente simples, mas que não são tão fáceis de operacionalizar. As técnicas são comumente utilizadas em práticas cotidianas para gerar diminuição no tempo de desenvolvimento. Dessa maneira, os resultados tendem a ser melhores visto que boa parte das funcionalidades já foram desenvolvidas.

REFLITA

A reengenharia permite, além de reescrever o sistema, compreender os erros e falhas e promover os devidos ajustes na nova construção do software. Na engenharia reversa também é possível encontrar erros, falhas, bugs e demais inconformidades?

Caro aluno, ao longo das discussões, exemplos e conceituações desta seção, o objetivo foi mostrar a você, a partir de um olhar profissional, as formas encontradas na engenharia de software para promover os processos de manutenção e evolução do software. Os assuntos discutidos aqui são largamente utilizados nas atividades diárias de desenvolvimento de software, por isso esse conhecimento é tão importante para a sua carreira profissional.

Na unidade 6 do livro intitulado *Engenharia de Software* (PERINI; HISATOMI; BERTO, 2009), disponível na Biblioteca Virtual, são tratados os assuntos relacionados à manutenção de software. Este é um referencial bibliográfico muito interessante para a compreensão dos processos de manutenção e evolução de software, por isso aproveite a leitura!

PERINI L. C.; HISATOMI I. H.; BERTO, W. L. **Engenharia de software**. Pearson Prentice Hall, São Paulo, 2009.

FAÇA VALER A PENA

Questão 1

O versionamento de software é uma técnica que agrupa informações importantíssimas, as quais auxiliam na identificação das alterações promovidas pelos desenvolvedores de software. Porém, para que isso ocorra, é necessário conhecer as partes que compõem essa numeração.

Se um sistema teve a sua versão alterada de 2.3.4.5 para 2.4.4.5, ocorreu uma alteração na:

- a. Estrutura como um todo, fazendo com que o software fosse totalmente modificado.
- b. Implementação de novas funcionalidades dentro do sistema.
- c. Correção de falhas e bugs encontrados na sua utilização.
- d. Segurança, devido à vulnerabilidades do sistema.
- e. Forma de acessar o sistema.

Questão 2

As atividades de manutenção de software estão presentes no ciclo de vida do desenvolvimento de software com o intuito de, no início, evitar que o sistema chegue aos usuários com falhas, e, se o produto de

software já estiver em uso, fazer as devidas manutenções a fim de promover sua evolução. Quanto à classificação e tipos de manutenção, observe as afirmativas a seguir:

- I. Manutenção adaptativa significa que determinada funcionalidade migrará para outro sistema, de forma a se adaptar em um módulo, por exemplo.
- II. Manutenção corretiva diz respeito às atividades que visam resolver falhas, erros e demais motivos que possam estar degradando o software.
- III. Manutenção evolutiva é utilizada para melhorias de funcionalidades já implementadas.

Assinale a alternativa CORRETA:

- a. Está correta apenas a afirmativa I.
- b. Está correta apenas a afirmativa II.
- c. Está correta apenas a afirmativa III.
- d. Estão corretas apenas as afirmativas I e II.
- e. Estão corretas apenas as afirmativas I e III.

Questão 3

Os processos de manutenção e de evolução de software podem ser feitos por diversos métodos ou meios, entre eles a reengenharia e a engenharia reversa.

Quanto às características de ambas na evolução de software, analise as asserções a seguir e a relação proposta entre elas.

- I. A reengenharia de software pode utilizar a engenharia reversa em seus processos.

POIS

- II. É necessário, muitas vezes, fazer o processo reverso de desenvolvimento a fim de reconstruir o sistema.

A seguir, assinale a alternativa correta:

- a. As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- b. A asserção I é uma proposição verdadeira, e a asserção II é uma proposição falsa.
- c. As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- d. A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- e. As asserções I e II são proposições falsas.

0

Ver anotações

REFERÊNCIAS

GITHUB. **1.1 Começando - Sobre Controle de Versão**, [S./.] c2021.

Disponível em: <https://cutt.ly/ujOsgiq>. Acesso em: 07 jan. 2021.

PÁDUA, Wilson. **Engenharia de Software**. 1^a Ed. Rio de Janeiro: LTC, 2019.

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

PERINI L. C.; HISATOMI I. H.; BERTO, W. L. **Engenharia de software**. São Paulo: Pearson Prentice Hall, 2009. Disponível em:
<https://cutt.ly/pjOssev>. Acesso em: 12 de jan. 2021.

VETORAZZO, A. de S. **Engenharia de software**. Porto Alegre: SAGAH, 2018.

FOCO NO MERCADO DE TRABALHO

MANUTENÇÃO E EVOLUÇÃO DE SOFTWARE

Sergio Eduardo Nunes

Ver anotações

RELATÓRIO COM PROPOSTA PARA A EVOLUÇÃO DO SOFTWARE

Por meio de um relatório, o auditor deve propor a evolução do software, com a inserção de novas funcionalidades, interface e demais benefícios que promovam a evolução do software alinhadas às novas necessidades e as tendências de mercado.



Fonte: Shutterstock.

Deseja ouvir este material?

SEM MEDO DE ERRAR

Valeu muito a pena você ter se empenhado nas consultorias de auditoria na empresa pela qual foi contratado. Não demorou muito e você foi chamado novamente para realizar um novo trabalho. Agora, o objetivo é gerar um relatório que proponha a evolução do sistema em duas partes: lista de usuários e inserção de novos usuários.

Para isso, vamos observar a primeira interface na Figura 4.10.

Figura 4.10 | Interface de lista de usuários

USUÁRIOS			
Nome	E-mail	Senha	Excluir
Teste 123	teste@teste.com	0b4e7a0e5fe84ad35fb5f95b9ceecac79	
Segundo Teste	teste@segundo	e10adc3949ba59abbe56e057f20f883e	

Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

Para fins de divisão, a estrutura do relatório será apresentada em tópicos, como se segue:

- **Front end:**

- **Menu e lista:** cor de fundo (black) é moderna, porém as palavras que estão tanto no menu quanto na lista possuem formatos e cores que não favorecem a visualização. Dessa forma, a fonte e a cor devem ser alteradas.
- **Botões:** devem ser inseridos botões para adicionar novos usuários e outro para exclusão do usuário, sendo que este deve ser na cor vermelha, de forma a remeter o usuário a um alerta antes de executar a funcionalidade.

- **Back end:**

- **Menu:** o menu deve ter o sistema de recolher para que lista com os usuários ocupe toda a tela do sistema.

- **Botões:** ao clicar em excluir um usuário, antes de executar a transação, deve ser exibido um pop-up com a mensagem: "Deseja realmente excluir o Usuário?", na qual poderá ser escolhida a opção SIM ou NÃO.
- **Funcionalidade de ALTERAR:** caso o preenchimento do cadastro tenha sido feito com erros, deve haver uma opção para alterar o registro. Portanto, deve ser adicionada mais uma coluna com um botão que permita utilizar a funcionalidade.

Já para as sugestões na tela de inserção de novos usuários, vamos analisar a Figura 4.9.

Figura 4.9 | Interface de inserção de usuários

Usuários Clientes Produtos

Name:
Nome completo

E-mail:
[campo vazio]

Senha:
Somente números

SALVAR

Fonte: captura de tela do sistema em auditoria elaborada pelo autor.

- **Front end:**

- **Menu e lista:** cor de fundo (black) é moderna, porém as palavras que estão tanto no menu quanto no formulário possuem formatos e cores que não favorecem a visualização. Dessa forma, a fonte e a cor devem ser alteradas.
- **Botões:** devem ser adicionadas cores aos botões para destacar onde será clicado.

- **Back end:**

- **Menu:** o menu deve receber mais campos com as informações que a empresa julgar necessárias. Além disso, devem ser

utilizados botões, menus suspensos e outras formas de preenchimento que sejam práticas e modernas.

- **Botões:** deve ser adicionado um botão de voltar caso o usuário não deseje mais preencher o formulário.
- **Validação:** os campos devem receber a funcionalidade de validação de dados com o intuito de garantir a consistência e a integridade dos dados enviados ao banco de dados.

Tais sugestões visam à evolução do sistema de forma que possa trazer uma nova experiência aos usuários.