Unidade 12 - Manutenção e reengenharia de software

Apresentação

O processo de desenvolvimento de *software* vem se modernizando, buscando formas de entregar sistemas de maneira mais rápida e com qualidade. Apesar disso, esse processo ainda apresenta falhas.

Passado o seu tempo de vida médio, o *software* começa a ficar defasado, já não atendendo a todas as necessidades dos usuários. Por isso, torna-se indispensável falar sobre manutenção de *softwares*, ou seja, correções, melhorias, novas implementações ou retirada de funcionalidades que não fazem mais sentido no momento.

Existem vários tipos de manutenção: corretiva, adaptativa, preditiva e evolutiva. Há também as manutenções emergenciais, aquelas que devem ser feitas imediatamente, pois o sistema parou de funcionar.

Nesta Unidade de Aprendizagem, você vai estudar os conceitos de manutenção de *software*, os tipos de manutenção, a reengenharia e as técnicas de manutenção.

Bons estudos.

Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

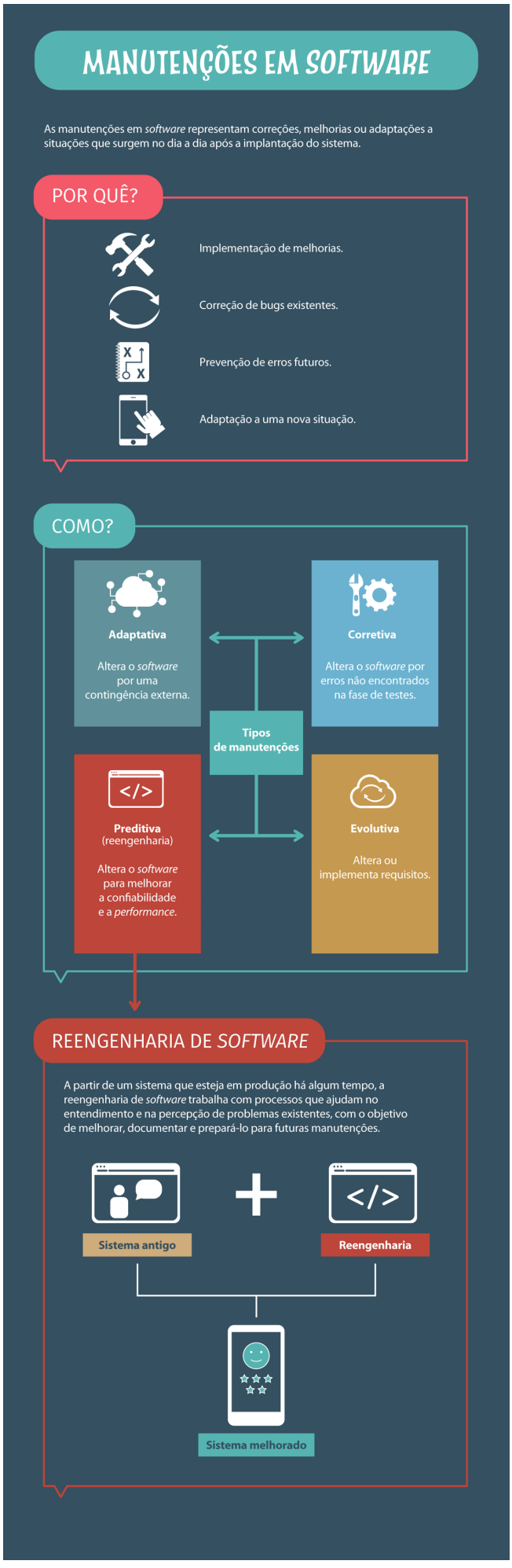
- Reconhecer os fundamentos da manutenção de *software*.
- Explicar os conceitos gerais da reengenharia de *software*.
- Demonstrar a aplicação da manutenção e da reengenharia de *software* na prática.

Infográfico

A implantação de um *software* em produção não significa que nunca mais ele será alterado ou melhorado; muito pelo contrário, ele precisa passar por cuidados para que seu ciclo de vida seja longo e produtivo.

Para alcançar essa meta, as manutenções evolutivas são necessárias. Para isso, novas funcionalidades serão agregadas, como as preventivas, ou seja, aquelas que se antecipam às falhas; e as corretivas, ou seja, aquelas que corrigem os *bugs* que já apareceram. A manutenção faz parte do ciclo do desenvolvimento do *software*.

Acompanhe, neste Infográfico, uma visão geral sobre reengenharia e manutenção de *software*.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Conteúdo do Livro

Todo *software* passa por um processo de envelhecimento natural. Mas o que fazer quando esse momento chega? A empresa deve desativá-lo? E se for uma ferramenta crítica para os negócios? Pensando em cenários como esses, a manutenção e a reengenharia de *software* surgem com técnicas que ampliam a vida útil de sistemas, fazendo com que eles continuem agregando valor para a organização.

No capítulo Manutenção e reengenharia de *software*, da obra *Gerenciamento de projeto de softwares*, base teórica desta Unidade de Aprendizagem, você conhece os conceitos necessários sobre manutenção de *software* e acompanha os insumos para aplicá-la de forma consciente e organizada em seu dia a dia profissional.

Boa leitura.

GERENCIAMENTO DE PROJETO DE SOFTWARES

Whesslley Rimar Bezerra

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Manutenção e reengenharia de *software*

OBJETIVOS DE APRENDIZAGEM

- Reconhecer os fundamentos da manutenção de *software*.
- Explicar os conceitos gerais da reengenharia de *software*.
- Demonstrar a aplicação da manutenção e da reengenharia de *software* na prática.

Introdução

Ao longo do tempo, todo *software* pode apresentar falhas e até mesmo se tornar obsoleto, principalmente quando não há manutenções em sua estrutura. Como você deve imaginar, tecnologias ultrapassadas podem apresentar problemas graves de segurança de dados. Ainda assim, muitas empresas, pensando nos altos custos que as manutenções podem gerar, acabam negligenciando esses importantes cuidados com os seus sistemas.

Neste capítulo, você vai conhecer os tipos de manutenção de *software* existentes e verificar quais são as técnicas utilizadas nesses procedimentos. Além disso, você vai estudar a reengenharia de *software* e ver um exemplo prático de sua aplicação.

Manutenção de *software*

O envelhecimento de um *software* pode ser inevitável e ocorre por diversos fatores, que não estão necessariamente ligados ao uso de tecnologias ob-

soletas. Para Valentim (2016), o envelhecimento de *software* pode ter causas diversas, tais como:

- não uso das boas práticas de programação;
- falhas (*bugs*) no código-fonte;
- degradações naturais, que não são motivadas por interferências humanas.

Também há outro motivo que pode acelerar o envelhecimento de um *software*: a falta de manutenção em sua estrutura. O *software*, assim como o *hardware*, precisa de manutenção para funcionar adequadamente ao longo do tempo.

Antes de você conhecer os tipos de manutenção existentes, precisa ter em mente o que de fato é a manutenção de *software*. Segundo Vargas e Pereira (2019), a manutenção de *software*, que está especificada na norma ISO/IEC 14764 (INTERNATIONAL STANDARD ORGANIZATION, 2006), é caracterizada pelo processo de modificação de um sistema ou aplicação que já está em uso pelo cliente. O propósito da manutenção é corrigir erros e instabilidades que são identificados quando a ferramenta já está em operação e sendo acessada pelos seus usuários. Como sintetiza Sommerville (2013), a manutenção de *software* é um processo geral ao longo do qual um sistema passa por modificações após ser liberado para utilização.

Tipos de manutenção de *software*

Um *software* pode passar por diferentes tipos de manutenções ao longo do seu ciclo de vida. As manutenções são essenciais para que a ferramenta sobreviva às mudanças de paradigmas e atualizações que acontecem na área de tecnologia. De acordo com Vargas e Pereira (2019), há quatro tipos de manutenção: manutenção preventiva, manutenção corretiva, manutenção adaptativa e manutenção evolutiva (perfectiva). Além disso, há a manutenção emergencial. Que tal conhecer cada uma delas?

- **Manutenção corretiva:** como o próprio nome diz, tem por objetivo corrigir falhas e defeitos em funcionalidades do *software*. Essas manutenções podem ser planejadas e executadas em um cronograma previsto. Geralmente, os erros são percebidos pelo usuário, que os comunicam à equipe, formalizando a comunicação por *e-mail* ou por um sistema de chamados. A partir daí, o time de desenvolvimento

dispõe de um prazo para solução definido em contrato. A manutenção corretiva pode envolver problemas de diferentes complexidades, que incluem desde casos mais simples (como erro no carregamento de uma página específica ou falha na ação de um botão), que podem não impactar diretamente as funcionalidades, até questões mais críticas, que podem envolver a paralisação parcial ou total do sistema.

- **Manutenção preventiva:** acontece antes que qualquer problema ou erro seja identificado no *software*. A ideia é se antecipar, por meio de testes em todas as funcionalidades do sistema, a fim de localizar possíveis inconsistências, erros de código e falhas que estejam interferindo no fluxo de execução do *software*. As manutenções preventivas são agendadas e, em geral, acontecem em um horário em que há menos usuários acessando a plataforma. Em algumas situações, quando a manutenção ocorre apenas em um módulo do sistema ou em uma funcionalidade específica, é possível manter as demais ferramentas e funcionalidades disponíveis e plenamente acessíveis para todos os usuários. Quando a manutenção é feita no *software* como um todo, o sistema pode ficar inacessível durante aquele período.
- **Manutenção adaptativa:** ocorre quando há a necessidade de adaptar o sistema a outro ambiente. Geralmente, esse tipo de manutenção ocorre quando há uma mudança de infraestrutura e outras tecnologias são utilizadas, ou quando ocorrem alterações na legislação que impactam diretamente a forma como o sistema se comporta. Outra possibilidade é que o sistema sofra ataques e tentativas de invasões que levem a empresa a adicionar novas camadas de segurança ao *software*.
- **Manutenção evolutiva (ou perfectiva):** acontece com a finalidade de adicionar novas funcionalidades ao *software*, evoluindo suas capacidades. Essas manutenções podem se originar de solicitações do cliente ou da própria empresa. Atualmente, a maioria das empresas implementa manutenções evolutivas em seus sistemas, mantendo-os competitivos no mercado. Os aplicativos móveis são excelentes exemplos disso: você já deve ter percebido que, depois de fazer o *download* de um aplicativo em seu *smartphone*, em poucos dias surgem novas atualizações do programa.
- **Manutenção emergencial:** essa manutenção não é planejada. Esse tipo de manutenção ocorre quando o sistema passa por alguma falha grave ou catastrófica que o deixa indisponível. A equipe, então, concentra-se na solução do problema, a fim de restabelecer o sistema o quanto antes.

Fique atento

Para Pressman e Maxim (2016), a manutenção de um *software* se inicia tão logo o sistema é colocado em produção. Os usuários começam a identificar as falhas nos primeiros dias de trabalho com a ferramenta e já as comunicam à empresa responsável, para que medidas sejam tomadas.

Leis de Lehman

Em meados da década de 1970, Meir M. "Manny" Lehman, professor e pesquisador da área da computação, contribuiu com estudos relacionados à evolução do *software*: as chamadas "leis de evolução de *software* de Lehman" ou simplesmente "leis de Lehman", que foram descritas entre os anos de 1974 e 1996. A seguir, você vai conhecer cada uma dessas leis.

- **Mudança contínua:** essa lei diz que um *software* deve se adaptar continuamente às novas demandas e ambientes. Se isso não acontecer, o *software* perderá engajamento dos usuários e se tornará menos satisfatório, ou seja, perderá sua utilidade. Dessa forma, manutenções devem ocorrer em sua estrutura para que o *software* continue cumprindo o papel para o qual foi projetado.
- **Complexidade crescente:** à medida que um *software* passa por manutenções, sejam elas adaptativas ou evolutivas, é natural que ele se torne mais complexo. Portanto, a equipe de desenvolvimento deve trabalhar para que o nível de complexidade do sistema seja diminuído, mesmo quando há alterações nas funcionalidades já existentes ou adição de novas funcionalidades.
- **Autorregulação:** essa lei diz que a evolução de um *software* é regulada por meio de *feedbacks* (HERRAIZ *et al.*, 2013). É certo que, quando um sistema é desenvolvido, por mais planejada que seja a sua construção, ele não atenderá a todos os objetivos da empresa. A visão de negócio da instituição se estende muito além das capacidades do *software*. Logo, os processos internos da organização regulam o esforço que é aplicado ao desenvolvimento do *software*.
- **Conservação da estabilidade organizacional:** a taxa de trabalho que uma organização investe no desenvolvimento de um sistema se mantém durante a vida útil do *software*. Isso significa que as influências consideradas para tomadas de decisão que culminam na evolução do

software são constantes; essas influências podem ser tanto da equipe gestora como dos usuários.

- **Conservação da familiaridade:** em sua vida útil, o *software* deve evoluir proporcionalmente ao conhecimento que a equipe de programadores tem das tecnologias utilizadas no seu desenvolvimento e nas suas funcionalidades. Assim, quanto maiores forem o conhecimento e a propriedade sobre o *software*, mais saudáveis serão as evoluções que podem acontecer com o sistema.
- **Crescimento contínuo:** um *software* deve permanecer em crescimento durante todo o seu ciclo de vida, de modo a manter o engajamento dos usuários. Logo, é possível perceber que não há como prever todas as funcionalidades que o *software* terá em sua vida útil; esse é um processo gradativo e progressivo. Contudo, esse crescimento é necessário para que o *software* se mantenha satisfatório para os usuários. Ele pode ocorrer por meio de manutenções preventivas, corretivas, adaptativas e evolutivas.
- **Qualidade diminuindo:** um *software* é construído para solucionar algum problema existente. Devido às mudanças que ocorrem na sociedade, na tecnologia e no ambiente operacional em que o *software* está inserido, suas funcionalidades se depreciam com o passar dos anos, tornando-se obsoletas. Logo, apesar de a qualidade ser algo intangível, ela pode diminuir quando o *software* não recebe atualizações que o mantenham adaptado às realidades da empresa, visto que esse ambiente pode passar por mudanças contínuas. Essa lei não tem ligação direta com defeitos de *software*, pois o sistema pode perder sua qualidade mesmo que esteja funcionando corretamente. Assim, os fatores que podem fazer a qualidade cair estão mais ligados ao comportamento dos usuários do que ao sistema em si, pois com o passar do tempo os usuários tornam-se mais exigentes com o sistema e mais insatisfeitos com ele. Por fim, a equipe deve se esforçar para continuar atendendo às expectativas dos usuários, prolongando a vida útil do *software*.
- **Sistema de *feedback*:** como o *software* pode ser utilizado por múltiplos usuários, sua evolução se dá com os retornos (*feedbacks*) positivos e negativos provenientes deles. Além disso, a verba destinada à evolução do *software* e a quantidade de solicitações para adição de novas funcionalidades também são indicativos de que o *software* está (ou não está) evoluindo.

Técnicas de manutenção de *software*

Agora que você conhece os tipos de manutenção de *software* e sabe como as leis de Lehman influenciam a evolução e a manutenção desses sistemas, vai verificar quais são as principais técnicas utilizadas nesse processo. Veja a seguir.

- **Documentação:** qualquer manutenção que ocorra em um *software* deve ser documentada. Quanto maior é o *software*, mais complexo se torna o entendimento de sua estrutura. Logo, para que nenhuma informação relevante se perca, deve haver documentações que armazenem todo o conhecimento e todas as regras de negócio que o *software* agrega. Um dos motivos de se documentar é que, se houver mudanças na equipe, o novo colaborador poderá se ambientar e entender como o *software* funciona ao consultar as documentações.
- **Controle de versões:** todo *software* que segue as boas práticas de programação e está em constante evolução deve ter seu código organizado em versões. Isso significa que, à medida que o *software* é atualizado e passa a ter novas funcionalidades, versões são guardadas de maneira a documentar as alterações que ocorreram na sua estrutura, facilitando manutenções posteriores. A ideia é que, a cada versão lançada do sistema, possam ser corrigidos erros e falhas da versão anterior, além de serem lançadas novas funcionalidades para o produto. Manter um histórico atualizado de versões do *software* facilita a identificação, o entendimento e a correção de possíveis erros. Ferramentas como Git e Subversion (SVN) são conhecidos programas de controle de versões. Na prática, eles registram detalhadamente as modificações que ocorrem nos arquivos do projeto de *software*, guardando informações como o nome do usuário que fez a alteração, o horário e o ponto do código em que houve mudança. Além disso, os sistemas de controle de versão, como são chamados, podem mediar conflitos de código. Esses conflitos ocorrem quando mais de um usuário altera a mesma versão de um arquivo no mesmo ponto. Nesses casos, o sistema oferece ao programador o poder de escolha: ele pode manter uma ou outra das versões conflitantes. Se as atualizações incluem erros que não foram previstos, o sistema de controle de versão permite ainda que o usuário recupere versões anteriores do produto, a fim de restaurar o sistema a partir de um ponto específico que talvez seja mais estável do que a versão atual.

- ***Status reporting*:** quando há muitas pessoas trabalhando em um mesmo projeto, é comum a confecção de documentos que reportem o que cada membro da equipe está fazendo. Isso garante que a equipe permaneça alinhada quanto ao progresso do projeto. Esses documentos podem ser lançados semanal ou diariamente e relatam os avanços da equipe, bem como os problemas e inconsistências que estão impedindo o fluxo do projeto.
- **Código legível:** há uma premissa das boas práticas de programação que enfatiza que um código deve ser lido e compreendido por qualquer pessoa que faça a manutenção daquela estrutura. Um código complexo demais para ser compreendido pode dificultar e atrasar manutenções emergenciais, por exemplo. A legibilidade engloba desde a adição de comentários no código-fonte (que expliquem o que determinado trecho de código faz) até a declaração de variáveis e funções com nomes intuitivos (que de alguma forma indiquem qual é o seu propósito no código). Por exemplo: uma variável utilizada para armazenar o nome de um usuário do sistema será compreendida com muito mais facilidade se for chamada de `nomeUsuario` ou simplesmente `nome`.

Reengenharia de *software*

No desenvolvimento de sistemas, a reengenharia representa a reconstrução de um *software* visando a implementar melhorias que permitam o surgimento de um novo produto, com qualidade igual ou superior à do produto anterior. Para isso, são utilizadas tecnologias mais atuais. Durante a reengenharia e quando não se tem registro documental de como o *software* foi construído, pode-se utilizar também o processo de engenharia reversa, que você vai estudar adiante.

Um dos objetivos da reengenharia é fazer alterações significativas em um *software* para que a sua manutenção seja facilitada. Assim, a reengenharia trabalha com modificações na estrutura do *software*, permitindo que as manutenções aconteçam em tempo oportuno, sem que os desenvolvedores tenham dificuldades para lidar com as tecnologias utilizadas.

Há, porém, diferenças significativas entre a construção de um sistema novo e a construção de um sistema desenvolvido por meio do processo de reengenharia de *software*. Um novo *software* geralmente é construído a partir de um planejamento escrito e visual (protótipos) do que se deseja desenvolver. Já na reengenharia, é utilizado como modelo e ponto de par-

tida o *software* já existente. Logo, o sistema é estudado a fundo para que sejam compreendidas as suas especificidades. A partir desse ponto, pode-se modificar as funcionalidades existentes, dando-lhes novas perspectivas. Nesse momento, são aproveitadas as possibilidades de utilização de novas tecnologias, dando sobrevida às funcionalidades do *software* antigo no novo *software* por mais alguns anos.

O processo de reengenharia de *software* é aplicado principalmente em *software* legado, ou seja, em programas que foram construídos quando era comum a utilização de outras técnicas e linguagens de programação e que, portanto, passaram pelo processo de obsolescência. Como esses programas ainda atendem às necessidades das empresas e como a construção de novos produtos pode levar muito tempo, é comum aplicar a reengenharia nesses contextos.

Além disso, muitos desses programas legados suprem necessidades consideradas críticas pelas organizações, e não se pode simplesmente descartá-los. Dependendo do *software* e do seu tempo de existência, pode ser que não haja nem mesmo uma documentação adequada sobre as suas regras de negócio, inviabilizando a sua desativação. Esse é mais um motivo para que a reengenharia seja aplicada; assim, o *software* pode ser utilizado por mais tempo.

Fique atento

A reengenharia é um processo que exige bastante estudo e cuidado, uma vez que o *software* pode ter sido desenvolvido quando não havia técnicas organizadas de desenvolvimento. Assim, o código-fonte pode ser muito complexo. Em resumo, um *software* só deve passar pelo processo de reengenharia quando a organização o considera muito útil e, ao mesmo tempo, tem grandes dificuldades de mantê-lo.

Benefícios da reengenharia de *software*

Três grandes benefícios se destacam na reengenharia de *software*: redução de riscos, redução de custos e redução de complexidade de sistemas legados. Sommerville (2013) afirma que há um alto risco na reconstrução total de um *software* que seja crítico para os negócios. Logo, a redução de riscos se refere à ideia de que, quando um *software* é reconstruído com as técnicas da reengenharia, a empresa e o time de programadores não têm de passar pelas etapas de desenvolvimento a partir do zero, pois podem aproveitar as especificações que já existem no sistema antigo, evitando erros críticos na construção.

A redução de custos ocorre porque os custos da reengenharia de um *software* são mais baixos do que os custos de uma reconstrução total, já que não há a necessidade de construir tudo do zero. Muitos elementos podem ser aproveitados do sistema legado. Por exemplo: se o projeto for articulado para a reconstrução apenas do *front-end* (toda área visual) de um sistema *web*, pode-se aproveitar ainda o *back-end* (parte funcional com as regras de negócio e o banco de dados) do sistema anterior e conectá-lo ao novo *front-end* desenvolvido. Isso reduz os custos com o desenvolvimento de um novo *back-end*.

A redução de complexidade, por sua vez, caracteriza-se pela ideia de que, na reconstrução do sistema, novas tecnologias, com *performances* melhoradas, são utilizadas. Isso pode facilitar a realização de manutenções futuras.

Manutenção e reengenharia de *software* na prática

Um questionamento que você pode fazer é: como se dá a aplicação prática da manutenção de *software*? Vargas e Pereira (2019) destacam que a manutenção envolve algumas tarefas. Veja a seguir.

- A pessoa, equipe ou empresa deve estabelecer uma organização para que a manutenção seja possível.
- Sequências padronizadas de eventos devem ser definidas para pedidos de manutenção.
- Procedimentos para a documentação das atividades de manutenção devem ser definidos com base em dados relativos ao *software*. Por exemplo:
  - nome do *software*;
  - data de instalação do *software*;
  - quantidade de comandos-fonte;
  - quantidade de pessoas dedicadas à manutenção;
  - linguagem de programação utilizada;
  - quantidade de comandos-fonte adicionados a cada alteração do *software*;
  - tipo de manutenção realizada;
  - datas de início e fim da manutenção.
- Critérios de revisão e avaliação da manutenção devem ser definidos.

Alguns programas podem ser utilizados como apoio na manutenção de *software*. Eles auxiliam no processo de testes, identificando erros no código-fonte da aplicação ou latências na infraestrutura, que impactam o desempenho do *software*. Como exemplos, você pode considerar o Bugzilla e o Micro Focus Silk Performer.

- **Bugzilla:** é uma ferramenta baseada na *web* que faz parte do projeto Mozilla (mesma criadora do navegador Mozilla Firefox) e que tem o objetivo de auxiliar desenvolvedores a rastrear erros em *software*. Teve sua primeira versão lançada em 1998 e passou por diversas atualizações até chegar à sua versão mais estável, datada de 2018. Além do rastreamento de erros, destacam-se: o recurso para comunicação entre desenvolvedores; a possibilidade de submeter e revisar *patches*, que são alterações no *software*; a emissão de relatórios de *bugs*; e o controle que visa a estimar o tempo que um *bug* levará para ser corrigido (BUGZILLA, 2020).
- **Micro Focus Silk Performer:** é uma ferramenta utilizada na realização de testes de desempenho (performance) de aplicações *web*, *mobile* e corporativas. Com ela, é possível avaliar quão estável está uma aplicação. Os testes podem ser utilizados com o propósito de melhorar a experiência do usuário, pois os responsáveis pela manutenção do *software* podem criar *scripts* que descobrem erros sob a perspectiva do usuário comum (MICRO FOCUS, 2020).

Fique atento

É importante você ter em mente que os programas citados aqui são apenas exemplos. Assim como qualquer outra ferramenta, também podem se depreciar com o tempo e ser substituídos por outros mais modernos, com mais recursos e melhores *performances*.

Atividades de reengenharia de *software*

O processo de reengenharia de *software* pode ser dividido em seis atividades principais: análise do inventário, reestruturação do código, engenharia reversa, reestruturação dos documentos, reestruturação dos dados e engenharia direta. No Quadro 1, a seguir, veja a definição de cada uma das etapas da reengenharia de *software*.

Quadro 1. Atividades de reengenharia de *software*

Atividade	Descrição
Análise do inventário	A empresa deve ter um inventário com os dados de todas as aplicações que estão ativas. Nesse inventário, devem ser armazenadas informações como tamanho, idade e nível de importância para os negócios (para cada aplicação). Esse documento deve ser revisitado e analisado com periodicidade, uma vez que os *status* dessas aplicações podem mudar ao longo do tempo, modificando também as prioridades da reengenharia de *software*.
Reestruturação do código	A reestruturação do código é aplicada quando, em um sistema legado, as partes principais estão codificadas de maneira bem estruturada, mas módulos específicos podem ter sido codificados de maneira complexa, o que dificulta a compreensão necessária para testá-los. Dessa forma, esses módulos específicos podem ser reestruturados.
Engenharia reversa	A engenharia reversa é o processo pelo qual um produto é desmontado até que se chegue à sua estrutura, a fim de entender o seu funcionamento. No desenvolvimento de *software*, a engenharia reversa é realizada por meio de ferramentas que conseguem extrair o código-fonte a partir do *software* executável, exibindo detalhes da implementação ao programador. Ele, por sua vez, pode alterar e melhorar o código com técnicas e práticas de desenvolvimento atuais.

Atividade	Descrição
Reestruturação dos documentos	A documentação é um ponto muito importante no desenvolvimento de um *software*, principalmente porque ela representa as particularidades do sistema, de modo a transmitir esse conhecimento para as partes interessadas. A documentação, entretanto, deve atingir um mínimo aceitável, ou seja, não precisa ser feita de forma exagerada.
Reestruturação dos dados	Os dados só devem ser reestruturados quando estão organizados em uma estrutura considerada fraca, ou seja, quando estão em um modo não relacional. Os dados, diferentemente do código-fonte, ao passar por um processo de reengenharia, devem passar por esse processo de maneira completa. Em suma, não é possível reestruturar apenas uma parte deles.

Fonte: Pressman e Maxim (2016).

Ao trabalhar com desenvolvimento, você deve ter em mente que a manutenção precisa ocorrer em diversos momentos do ciclo de vida de um *software*. Isso garante um aumento significativo do tempo de uso do sistema. Ignorar ou negligenciar as manutenções em um *software* pode acelerar o processo de envelhecimento do programa, tornando o seu uso inviável para a maioria dos usuários.

Referências

BUGZILLA. *What is Bugzilla*? 2020. Disponível em: https://bugzilla.org/about. Acesso em: 15 set. 2020.

HERRAIZ, I. *et al.* (2013). The evolution of the laws of software evolution: a discussion based on a systematic literature review. *ACM Computing Surveys*, New York, v. 46, n. 2, p. 1-32, Dec. 2013. Disponível em: https://www.researchgate.net/profile/Gregorio_Robles/publication/262297736_The_Evolution_of_the_Laws_of_Software_Evolution_A_Discussion_Based_on_a_Systematic_Literature_Review/links/54ef91470cf25f74d7227eb0/The-Evolution-of-the-Laws-of-Software-Evolution-A-Discussion-Based-on-a-Systematic-Literature-Review.pdf. Acesso em: 15 set. 2020.

INTERNATIONAL STANDARD ORGANIZATION. *ISO/IEC 14764*: Software Engineering, Software Life Cycle Processes, Maintenance. Geneva: ISO, 2006.

MICRO FOCUS. *Silk Performer*. 2020. Disponível em: https://www.microfocus.com/pt-br/products/silk-performer/overview. 2020. Acesso em: 15 set. 2020.

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

SOMMERVILLE, I. *Engenharia de software*. 9 ed. São Paulo: Pearson Education, 2013.

VALENTIM, N. A. *Envelhecimento e rejuvenescimento de software*: 20 anos (1995-2014): panorama e desafios. 2016. Dissertação (Mestrado em Ciência da Computação) – Faculdade de Computação, Universidade Federal de Uberlândia, Uberlândia, 2016. Disponível em: https://repositorio.ufu.br/bitstream/123456789/18212/1/EnvelhecimentoRejuvenescimentoSoftware.pdf. Acesso em: 15 set. 2020.

VARGAS, A. A. F.; PEREIRA, J. V. dos S. Gerenciamento da manutenção de software. *Pesquisa & Educação a Distância*, [*S.l.*], n. 16, p. 1-14, 2019. Disponível em: http://revista.universo.edu.br/index.php?journal=2013EAD1&page=article&op=view&path%5B%5D=8294&path%5B%5D=3992. Acesso em: 15 set. 2020.

Fique atento

Os *links* para *sites* da *web* fornecidos neste capítulo foram todos testados, e seu funcionamento foi comprovado no momento da publicação do material. No entanto, a rede é extremamente dinâmica; suas páginas estão constantemente mudando de local e conteúdo. Assim, os editores declaram não ter qualquer responsabilidade sobre qualidade, precisão ou integralidade das informações referidas em tais *links*.

Conteúdo:

SAGAH
SOLUÇÕES EDUCACIONAIS INTEGRADAS

Dica do Professor

Manter um *software* não é uma tarefa simples, mas é extremamente necessária para que ele continue sendo assertivo em seus propósitos. Algumas técnicas podem auxiliar e facilitar esse processo.

Nesta Dica do Professor, você conhece alguns conceitos importantes sobre manutenção e reengenharia de *software*, que são essenciais para o dia a dia de quem trabalha com desenvolvimento de sistemas.

Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

Exercícios

1) **Após o desenvolvimento de um projeto de *software*, ou seja, quando ele já está em produção, manutenções podem ocorrer, a fim de ampliar a sobrevida do *software*. Um dos tipos de manutenção é a adaptativa.**

Escolha a alternativa que defina o objetivo dessa manutenção.

A) Corrigir erros e inconsistências ocorridos durante o processo de codificação dos programas.

B) Corrigir erros e inconsistências ocorridos no processo de levantamento de requisitos.

C) Alterar o programa para implementar novas funcionalidades não previstas no levantamento de requisitos.

D) Modificar um programa devido à fórmula de cálculo de um imposto ter sido alterada pelo governo federal.

E) Corrigir uma falha percebida por alguns usuários após o sistema já estar em operação.

2) **Com relação à manutenção de *software*, ou seja, qualquer alteração no sistema após a sua implantação, vale dizer que:**

I. É o processo que ocorre logo após a fase de levantamento de requisitos e é classificada como corretiva ou evolutiva.

II. É o processo que ocorre após a fase de implantação do *software* e pode ser adaptativa, corretiva, preventiva ou evolutiva.

III. Quanto maior o esforço empregado para tornar o *software* manutenível, menor o custo da manutenção.

IV. É um processo mais rápido e mais barato quando o profissional for o mesmo que participou do desenvolvimento.

Assinale a alternativa que apresenta as afirmações corretas.

A) I, II, III e IV.

B) I e IV.

C) II e IV.

D) II, III e IV.

E) III e IV.

3) **A reengenharia de *software* é uma das formas de se fazer manutenção nos *softwares*.**

Assinale verdadeiro (V) ou falso (V) para cada uma das afirmações a seguir:

( ) Reconstrução de algo do mundo real, com melhorias e aperfeiçoamentos em relação ao modelo inicial.

( ) A reengenharia é composta por processos de engenharia reversa, seguida de processos de engenharia progressiva.

( ) Reorganização e modificação de sistemas legados, produzindo um sistema novo com maior facilidade de manutenção.

( ) Em algumas situações, as empresas acabam optando por ficarem com os sistemas legados por medo de que a reengenharia não perceba as regras de negócio implícitas nos sistemas.

( ) A importância da reengenharia para os *softwares* já existentes é iniciar um novo produto e fazer com que todo o conteúdo dele seja desconsiderado.

Assinale a alternativa que apresenta a sequência correta.

A) V, V, V, V, F.

B) F, V, F, V, F.

C) V, F, V, F, V.

D) F, V, V, V, V.

E) V, F, F, F, V.

4) **Com base no escopo "venda de seguros para veículos leves", foi desenvolvido um projeto para venda de seguros para veículos. Os testes foram feitos com diversas marcas de veículos e o sistema entrou em produção. No primeiro dia, o lojista foi vender o seguro para o proprietário de um caminhão e o sistema não permitiu; isso foi reclamado junto ao SAC da empresa. Para liberar a venda do seguro para o caminhão, será necessária uma manutenção no sistema.**

Qual é o tipo dessa manutenção?

A) Evolutiva.

B) Corretiva.

C) Adaptativa.

D) Preventiva.

E) Constante.

5) **Para fazer todo o processo de manutenção, é necessário o conhecimento das técnicas de manutenção de *software*.**

Relacione as técnicas com as descrições.

Técnicas:
I) Documentação.
II) Versionamento.
III) *Status Reporting*.
IV) Codificação.

Descrições:
( ) Documentação de todas as alterações efetuadas no sistema, com a possibilidade de recuperação das versões anteriores.
( ) Indentação, comentários e práticas que produzem legibilidade.
( ) Atualização das alterações efetuadas com o objetivo de transmitir conhecimento sobre o sistema.
( ) Alinhamento que exibe o estado atual do projeto e o que foi alterado em um período pela equipe de desenvolvimento.

Assinale a alternativa que apresenta a sequência correta.

A) I - II - III - IV.

B) I - IV - III - II.

C) II – IV – I – III

D) III - I - IV - II.

E) IV - III - II - I.

Na prática

Provavelmente, você já tenha ouvido a frase: "Quebrou? Joga fora e compra outro!". Muitas vezes, isso pode se aplicar ao nosso dia a dia, porém nas empresas, principalmente no que tange ao desenvolvimento de *software*, essa não deve ser uma prática comum. Por quê?

Softwares custam caro e levam tempo para serem desenvolvidos. Eles necessitam de profissionais especializados, e tudo isso faz com que cada vez mais as empresas mantenham seus *softwares* legados. Isso porque além de estarem funcionando, eles continuam resolvendo os problemas para os quais foram projetados e, para as novas funcionalidades, fazem-se as manutenções evolutivas.

Acompanhe, neste Na Prática, a história de uma empresa de engenharia que dispunha de um *software* de controle de obras utilizado por muitos anos, até que começou a sentir que algumas necessidades não estavam previstas para a versão atual. Por isso, teve que fazer uma manutenção.

Conteúdo interativo disponível na plataforma de ensino!