

# TEXTO DE APOIO



## AULA 2

### Teste de Software

**Professor** Calebe de Paula Bianchini



Universidade Presbiteriana  
**Mackenzie**





# Sumário



**REVISÃO DE SOFTWARE ..... 3**

**REFERÊNCIAS ..... 13**

# TÍTULO DO TEXTO DE APOIO

## REVISÃO DE SOFTWARE

Um dos princípios básicos de qualidade de software é que não existe software perfeito. Essa máxima não leva as empresas e suas equipes ao desânimo ou à desistência no que diz respeito a conseguir qualidade, mas impulsiona a todos em procurar desenvolver os artefatos ao longo do processo de desenvolvimento de software com o máximo de cuidado e atenção.

Além disso, a entrega de um software com qualidade e que atende às expectativas do cliente é uma questão estratégica e de posicionamento de mercado na economia moderna. Por isso, atentar-se às boas práticas é necessário, e também realizar atividades que aumentam ainda mais a qualidade do software é vital para o produto, para o processo e para a empresa.

As revisões são atividades tão importantes para o desenvolvimento de software que podem ser comparadas a uma borracha no processo de se escrever um texto: uma vez encontrado algum problema no texto, ele é removido e um novo trecho é produzido. Ao percebermos que a natureza humana produz artefatos de software com defeito (já que errar é humano), é importante lembrar a origem desses defeitos (como eles surgem) e o processo de propagação desse defeito, conforme mostra a Figura 1. A origem do problema é no ser humano que, por meio de um *erro*, produz um artefato com *defeito*, que, uma vez inserido no código-fonte, pode produzir uma *falha*, que é o resultado visível do problema deste software.

**Figura 1 – O processo de inserção de *defeito* em software**



Fonte: Elaborada pelo autor.

Uma observação importante a ser feita neste ponto é que um defeito produzido pode estar em qualquer tipo de artefato de software: no processo, nos modelos, na documentação, nos requisitos, na arquitetura, nos dados de teste, no planejamento, no código-fonte etc. E, por isso, a revisão é uma atividade de filtragem que pode ser aplicada em qualquer artefato ao longo do processo de desenvolvimento de

software, como garantia de qualidade.

Na perspectiva da gerência de controle de qualidade de software, a aplicação da revisão permite utilizar a diversidade de papéis e perfis de pessoas para:

- apontar pontos de aperfeiçoamento e melhoria dos artefatos e, consequentemente, das pessoas e equipes envolvidas;
- confirmar que mudanças, melhorias ou correções são desnecessárias ou indesejáveis;
- uniformizar a construção de artefatos de software com qualidade, uma vez que o conhecimento pode ser difundido, inclusive nos processos de revisão.

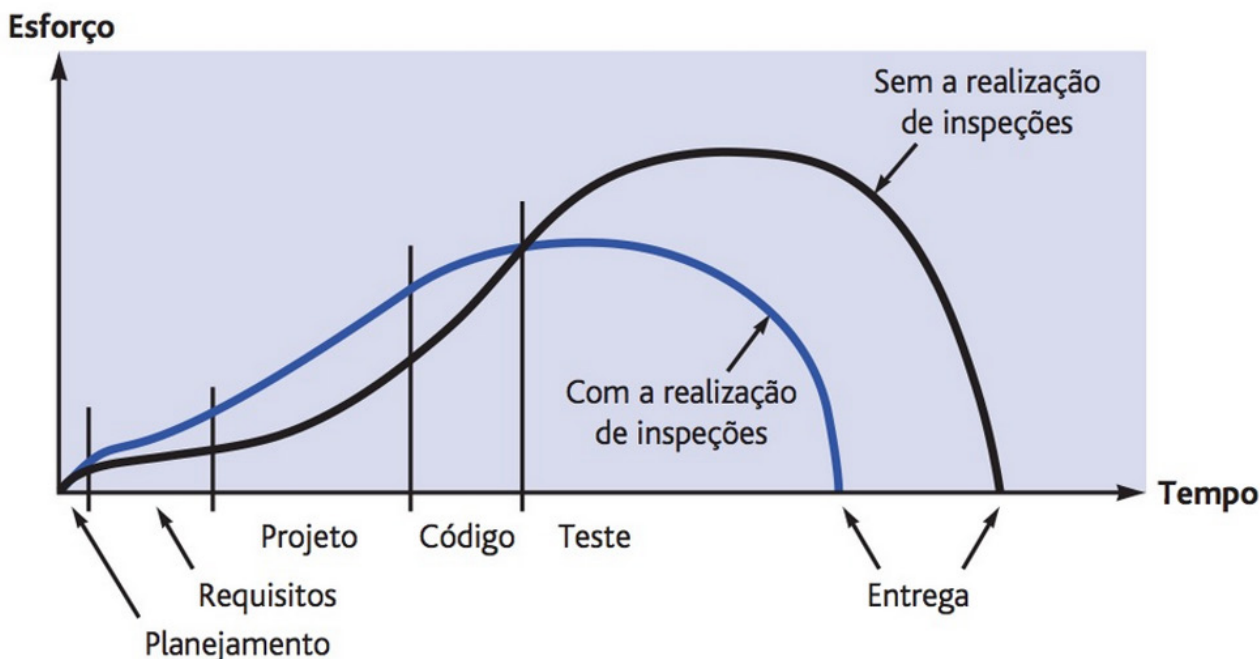
A redução da quantidade de defeitos em um software nas fases iniciais do processo de desenvolvimento é um dos fatores que deveria levar as empresas a considerarem atividades de revisão. A essa altura, você já deve saber que a correção de um defeito no software em fases avançadas do processo de construção aumenta os custos dessa correção. Portanto, esse conhecimento justifica a aplicação de técnicas de revisão em artefatos o mais breve possível, não só para corrigi-los, mas também para diminuir o efeito propagador destes defeitos para outros artefatos, aumentando ainda mais o custo de correção.

Infelizmente, o processo de revisão não é tão comum nas empresas, que argumentam não ter tempo suficiente ao longo do processo para realizar esse tipo de atividade. Podemos, portanto, considerar que os prejuízos para a correção atrasada de defeitos não só envolvem questões financeiras, mas também operacionais e emocionais. A somatória destes e de outros fatores na efetiva construção do software sem defeitos pode ser traduzida, de forma geral, como *esforço*. Algumas empresas observaram os esforços empregados na construção de software com e sem a aplicação de técnicas de revisão e, conforme mostra a Figura 2, constataram que:

- de fato, é necessário um maior esforço no início do projeto de desenvolvimento de software, caso alguma prática de revisão seja adotada (como a inspeção);
- o tempo para a entrega de um software que aplicou técnicas de revisão (como inspeção) foi menor;
- o retorno do investimento em técnicas e práticas de revisão (como inspeção) no software foi de dez vezes;

- a quantidade medida de defeitos e problemas encontrados diminuiu em uma ordem de grandeza (ou seja, reduziu dez vezes).

**Figura 2 – Esforço para a construção de um software, considerando a revisão de software**



Fonte: Elaborada pelo autor.

Fica evidente que adotar práticas de revisão de software trará benefícios diretos e indiretos para o software. Porém, a adoção de diversas técnicas se faz necessária, pois, a cada momento do processo de construção de software, artefatos diferentes são produzidos. Existem, na literatura, três tipos básicos de revisão técnica de software, e você certamente já ouviu falar de algumas delas – talvez, não com o nome que apresentamos aqui.

A primeira técnica é conhecida como Walkthrough. Essa técnica consiste na avaliação cuidadosa (passo-a-passo) do artefato de software por uma equipe pequena (entre uma e cinco pessoas). Se esse artefato for um código-fonte, a avaliação é feita com base em um conjunto de dados que, na maioria das vezes, também poderá ser utilizado para as atividades de teste de software. Esse tipo de avaliação é muito parecido com o (famoso) teste de mesa (tão utilizado por professores e instrutores durante uma avaliação). No final, o que se pretende é avaliar o resultado da simulação daquele trecho de programa.



“Você sabe mesmo como fazer um teste de mesa? Veja o vídeo da SoftBlue e lembre os detalhes dessa técnica. Disponível em: <<https://www.youtube.com/watch?v=Atcfaafvs4M>>.”

Caso o artefato seja algum outro documento estático (que não seja possível “executar”), essa técnica se preocupa em avaliar o documento gerado, normalmente a partir de um outro artefato anterior. Por exemplo, se o artefato for um modelo de classe de domínio, a avaliação será feita a partir dos documentos anteriores a ele, como, por exemplo, um modelo de casos de uso, as histórias de usuário ou, até mesmo, um modelo de processo de negócio – perceba que todos esses podem ter dado origem às classes e seus relacionamentos.

Uma segunda técnica de revisão é chamada de Peer Review. Nesta técnica, os artefatos são produzidos por pares de engenheiros de software, sendo que um deles realiza o papel de revisor enquanto o outro desenvolve o artefato. Essa técnica pode ser adotada por alguns motivos: revisão dos artefatos ao longo de sua construção, espaço para discussão de ideias e soluções ao longo da construção, disseminação de conhecimento entre os membros da equipe, contínua atenção à excelência técnica e de desenho do projeto, diminuição de burocracia e o consequente aumento na comunicação livre, entre outros motivos que podem ser encontrados na literatura. Além disso, essa prática envolve a troca dos papéis ao longo da construção do artefato, fortalecendo os motivos apresentados anteriormente.

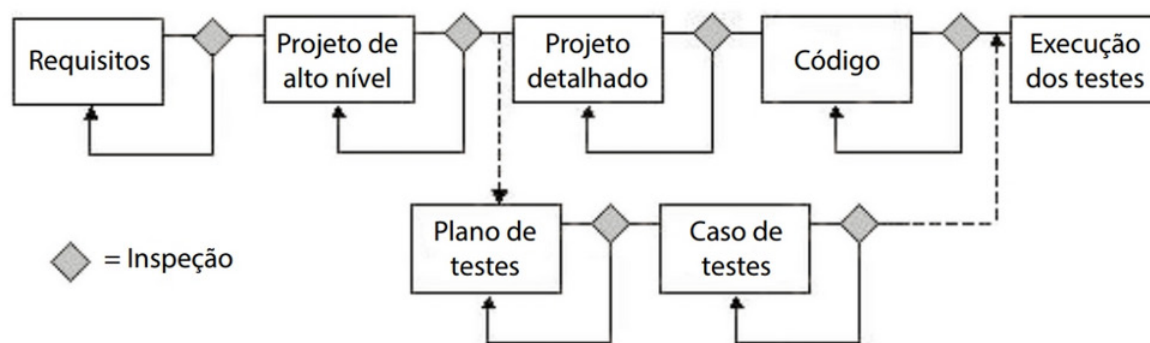
Novamente, essa técnica pode ser empregada em qualquer tipo de artefato, seja ele um documento, modelo, projeto ou código-fonte. Vale ressaltar, porém, que no processo de *Peer Review* é fundamental escolher (e definir) as normas e procedimentos que serão adotados ao longo da construção do artefato. Isso já pode ter sido feito pela área de qualidade responsável pelo projeto ou, caso necessário, pode ser feito pelos pares. Por exemplo, caso o projeto adote a descrição de histórias de usuário como parte dos artefatos de requisitos de software, o par envolvido na realização dessa tarefa deve entender sobre o assunto e estabelecer (ou seguir) padrões.



“Você conhece as histórias de usuário e como descrevê-las? Caso tenha interesse, veja mais sobre esse assunto em <<https://viniciuspessoni.com/2018/06/21/como-escrever-uma-boa-historia-de-usuario-user-story-para-automaizar-com-bdd/>>.”

Uma terceira técnica pode também ser adotada como revisão de software. Neste caso, ela é ainda mais formal, pois demandará uma estrutura de processo, bem como diversos papéis para sua completa realização. Ela é chamada de Inspeção de Software. De forma geral, essa técnica revisa os artefatos de software gerados ao final de cada fase ou iteração do processo de desenvolvimento de software (veja um exemplo na Figura 3). Nesta técnica, têm-se os seguintes passos:

- Planejamento: basicamente, nesta fase, algumas decisões de estruturação da inspeção são feitas, como escolher a equipe de revisores, definir os artefatos, escolher os autores, estabelecer datas, agendas e cronogramas, entre outras atividades que envolvem o planejamento.
- Apresentação: nesta etapa, os itens planejados são publicados para toda a equipe e os membros das equipes são convidados a assumir os papéis definidos. É possível haver uma fase de negociação.
- Preparação: antes de realizar a inspeção, todos precisam se preparar, conhecendo o processo, as listas de avaliações, os artefatos escolhidos e papéis envolvidos. A falta desse conhecimento antecipado fará com que o processo de revisão seja mais demorado.
- Reunião de revisão: os artefatos são apresentados pelos autores que, uma vez preparados, tem espaço para defender suas ideias e soluções adotadas. Neste momento, é importante saber que muitas cabeças não ajudam na solução do problema – além de que não é um tribunal de inquisição. Os defeitos que realmente forem encontrados devem ser registrados em um relatório (ou similar).
- Retrabalho: neste momento, o autor deve corrigir os problemas identificados e que foram documentados no relatório.
- Análise final do moderador: ao final, um relator é responsável por avaliar as mudanças feitas pelo autor, com base no relatório de defeitos apontado na reunião de revisão.



Fonte: Elaborada pelo autor.

Independentemente da técnica, aplicar a revisão em todo o artefato de software que for produzido ao longo do processo de desenvolvimento de software pode se tornar um grande problema (devido aos custos e prazos envolvidos). Por isso, a gerência de garantia de qualidade pode utilizar técnicas auxiliares para decidir onde e quando as revisões podem (ou devem) ser feitas. A natureza do software



já é um importante indicativo sobre essa decisão. Por exemplo, se o software que está sendo construído for de natureza crítica, a revisão dos artefatos se torna algo indispensável.

De forma geral, a escolha dos artefatos para revisão pode ser feita utilizando princípios conhecidos, por exemplo:

- análise de risco: quanto maior o impacto sobre a possível ocorrência de problemas em um software durante seu uso, deve-se construí-lo com muito mais zelo e, por isso, justifica-se a aplicação de revisão;
- princípios de indução: se a maior parte dos defeitos registrados foram encontrados em um determinado artefato de software, por indução, todos os demais artefatos derivados e relacionados a este artefato defeituoso sofreram interferência negativa, ou seja, também apresentarão uma alta densidade de defeitos;
- senioridade das equipes: a capacidade de abstração e habilidade técnica das equipes também devem ser consideradas não só para a construção dos artefatos, mas também para o processo de revisão. Alguns indicativos sobre a qualidade dos artefatos gerados pelos profissionais e equipes podem demandar revisões mais profundas e/ou criteriosas.

A prática de revisão não é realizada somente por métodos tradicionais e clássicos de desenvolvimento de software. As principais metodologias ágeis possuem os princípios de revisão como parte natural do processo. Vejam, por exemplo, o *eXtreme Programming* (XP): um de seus pilares se baseia na programação em pares (pair programming), ou seja, uma aplicação prática do Peer Review ao longo do desenvolvimento do software.



“Em 2016, pesquisadores da Universidade de Georgia fizeram uma pesquisa comparando equipes que utilizaram programação em pares e equipes que possuíam programação “solo”. Os resultados são bem animadores: maior qualidade, melhor produtividade, mais aprendizado para as equipes com programação em pares! Se tiver interesse em se aprofundar no assunto, acesse o trabalho completo por meio das credenciais do Portal CAPES no site da Biblioteca da Universidade Presbiteriana Mackenzie, no seguinte link: <<https://doi.org/10.1109/HICSS.2016.667>>.”



Quando são utilizados métodos ágeis para o desenvolvimento de software, a ênfase está muito mais na geração de artefatos executáveis, ou seja, no código-fonte, do que na geração de outros artefatos – mesmo assim, existem artefatos importantes, como as histórias do usuário e a arquitetura de software. Nesse caso, além das técnicas tradicionais de revisão que já foram apresentadas, as metodologias ágeis adaptaram muito bem as revisões para que sejam feitas em código-fonte. Por isso, é necessário um conjunto de ferramentas de automação e construção de software, como os controladores de versão. Entre essas ferramentas, podemos citar aquelas que implementam os padrões de *git* (como GitHub, GitLab, entre outras), além dos processos de controle de mudança de código-fonte, como o GitFlow e OneFlow, entre outras ferramentas que avaliam a qualidade do código-fonte.



“Você precisa entender como funciona a construção de software usando git. Para iniciar seus estudos, indicamos o livro mais conhecido (e oficial), disponível em <<https://git-scm.com/book/>>, além das boas práticas de evolução do software chamadas de GitFlow, disponível em <<https://nvie.com/posts/a-successful-git-branching-model/>> e OneFlow, em <<https://www.endoflineblog.com/oneflow-a-git-branching-model-and-workflow>>.”

Apesar da quantidade de ferramentas envolvidas, ainda assim a revisão de código é fundamental. Veja, por exemplo, o trecho de código da Figura 4a. A função deveria encontrar o maior elemento de um conjunto de dados.

**Figura 4 – Exemplos de funções que procuram o maior elemento de um conjunto de dados**

<pre>int getMax(int values[]) {     int max = 1000;     for(int value : values) {         if(value &gt; max) {             max = value;         }     }     return max; }</pre>	<pre>int getMax(int values[]) {     int max = values[0];     for(int value : values) {         if(value &gt; max) {             max = value;         }     }     return max; }</pre>
(a)	(b)

Fonte: Elaborada pelo autor.

Uma revisão nesse código permitiria detectar, por exemplo, que ele somente funciona se os dados do conjunto forem menores do que um valor adotado (no caso, 1000). Se esse defeito não for corrigido o mais rápido possível, seu mau funcionamento será propagado para outras partes do *software*, o que desencadearia um investimento maior de tempo e recursos para procurar a origem do problema. A Figura 4b apresenta uma versão melhorada para o algoritmo.




Será que você consegue encontrar mais um problema no código apresentado na **Figura 4b**? Dica: verifique a quantidade de elementos armazenadores no conjunto de dados e se eles funcionariam para qualquer caso – inclusive para um conjunto vazio.”

Um outro exemplo que também justifica a revisão de código é o desempenho de seu algoritmo. Mesmo sabendo que desempenho é algo percebido durante a execução de um software, existem indícios no código-fonte de que o software apresentará problemas no futuro. Um exemplo simples de desempenho são os algoritmos de ordenação, conforme apresenta a **Figura 5**.

**Figura 5 – Algoritmos de ordenação: quicksort vs. bubblesort**

<pre>int partition(int values[], int s, int e) {     int pivot = values[e];     int i = (s-1);     for (int j = s; j &lt; e; j++) {         if (values[j] &lt;= pivot) {</pre>	<pre>void bubblesort(int values[]) {     for(int i=0; i&lt;values.length; ++i) {         for(int j=0; j&lt;values.length; ++j) {             if(values[i] &gt; values[j]) {                 int temp = values[i];</pre>
<pre>                i++;                 int temp = values[i];                 values[i] = values[j];                 values[j] = temp;             }         }          int temp = values[i+1];         values[i+1] = values[e];         values[e] = temp;     } }</pre>	<pre>                values[i] = values[j];                 values[j] = temp;             }         }     } }</pre>

<pre> return i+1; }  void quicksort(int values[]) {     sort(values, 0, values.length-1); }  void sort(int values[], int begin, int end) {     if (begin &lt; end) {         int partition =             partition(values, begin, end);          sort(values, begin, partition-1);         sort(values, partition+1, end);     } } </pre>	
(a)	(b)

Fonte: Elaborada pelo autor.

Fica evidente que os dois algoritmos possuem um desempenho semelhante quando o conjunto de dados está em uma distribuição ruim para o algoritmo da Figura 5a. O pivô escolhido é sempre o menor elemento do subconjunto. Porém, considerando os casos gerais de distribuição dos elementos no conjunto de dados, o código da Figura 5a sempre será mais rápido que o código da **Figura 5b**.



“Caso você queira se lembrar sobre o funcionamento dos algoritmos de ordenação, veja as explicações e o comparativo visual no site <<https://www.toptal.com/developers/sorting-algorithms>>.”

Sendo possível, inclusive, utilizar um checklist para analisar diversos pontos do software, como, por exemplo, no código:

Sendo possível, inclusive, utilizar um checklist para analisar diversos pontos do software, como, por exemplo, no código:

- a inicialização correta das variáveis;
- o uso definido dos limites inferior/superior das estruturas em laços de repetição;
- a correta definição da expressão condicional nas diversas estruturas de controles;
- as variáveis utilizadas nas entradas dos dados são as corretas;
- a ordem certa dos parâmetros na chamada de uma função/método;
- o tratamento dos erros de possíveis condições do código (veja a Figura 4b, por exemplo);
- o entendimento do desempenho dos algoritmos;
- a adoção dos padrões de convenção de código da empresa;
- outros itens que podem ser aproveitados ou definidos pela empresa ou pelos membros da equipe de desenvolvimento de software.

Assim, fica evidente que o processo de revisão é fundamental não só em artefatos estáticos, como em documentos, modelos, conjuntos de testes etc., mas também no código-fonte do software.

## REFERÊNCIAS

GONÇALVES, P. et al. *Testes de software e gerência de configuração*. Porto Alegre: Sagah, 2019.

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

SOMMERVILLE, I. *Engenharia de Software*. 10. ed. São Paulo: Pearson Education do Brasil, 2018.

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