



---

# VALIDAÇÃO DO SOFTWARE: TESTES DE SOFTWARE E APLICAÇÕES DE SEGURANÇA NO SISTEMA



**Luís Otávio Toledo Perin**

**VALIDAÇÃO DO SOFTWARE: TESTES DE  
SOFTWARES E APLICAÇÕES DE SEGURANÇA NO  
SISTEMA**

1ª edição

Londrina  
Editora e Distribuidora Educacional S.A.  
2020

© 2020 por Editora e Distribuidora Educacional S.A.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Editora e Distribuidora Educacional S.A.

**Presidente**

Rodrigo Galindo

**Vice-Presidente de Pós-Graduação e Educação Continuada**

Paulo de Tarso Pires de Moraes

**Conselho Acadêmico**

Carlos Roberto Pagani Junior  
Camila Braga de Oliveira Higa  
Carolina Yaly  
Giani Vendramel de Oliveira  
Henrique Salustiano Silva  
Mariana Gerardi Mello  
Nirse Ruscheinsky Breternitz  
Priscila Pereira Silva  
Tayra Carolina Nascimento Aleixo

**Coordenador**

Henrique Salustiano Silva

**Revisor**

Rafael Maltempe da Vanso

**Editorial**

Alessandra Cristina Fahl  
Beatriz Meloni Montefusco  
Gilvânia Honório dos Santos  
Mariana de Campos Barroso  
Paola Andressa Machado Leal

Dados Internacionais de Catalogação na Publicação (CIP)

---

P445v Perin, Luís Otávio Toledo.  
Validação do software: testes de softwares e aplicações  
de segurança no sistema/ Luís Otávio Toledo Perin, –  
Londrina: Editora e Distribuidora Educacional S.A. 2020.  
44 p.

ISBN 978-65-5903-032-3

1. Validação 2. Software 3. Segurança I. Título.

CDD 005

---

Raquel Torres - CRB: 6/2786

2020  
Editora e Distribuidora Educacional S.A.  
Avenida Paris, 675 – Parque Residencial João Piza  
CEP: 86041-100 — Londrina — PR  
e-mail: editora.educacional@kroton.com.br  
Homepage: <http://www.kroton.com.br/>

## SUMÁRIO

Qualidade do software, documentação de testes e o modelo TMMi_ 05	
Gerenciamento dos testes de software com ferramentas CASE open source _____	23
Procedimentos para garantir a segurança do software _____	39
Ferramentas CASE para testes de segurança do software _____	55

# Qualidade do software, documentação de testes e o modelo TMMi.

Autoria: Luís Otávio Toledo Perin

Leitura crítica: Rafael Maltempe da Vanso



## Objetivos

- Definir o que é qualidade de software.
- Compreender o processo de documentação de testes de software.
- Apresentar o modelo TMMi para testes de software.



# 1. A excelência desejada

## 1.1 Introdução

Nesta leitura, será abordada a qualidade de software, seu processo de documentação e o modelo TMMi. Não pense que esses processos representam um mero complemento à criação ou concepção do produto software propriamente dito, visto que o sucesso ou fracasso de um sistema informatizado depende de suas boas práticas durante todo o desenvolvimento, além de garantir que aquilo que foi produzido não apresentará problemas ou, caso apresente, que sejam prontamente resolvidos.

A importância de conhecer como a qualidade pode ser mensurada é algo que será abordado e detalhado ao longo desta leitura e, para darmos início a nossa jornada, pensemos na seguinte questão: você já percebeu como o uso da tecnologia tem crescido nos últimos tempos? Caso não tenha percebido, saiba que isso é normal, pois é algo que se tornou um hábito e já é parte integrante da vida de todos.

Quem nunca utilizou um aplicativo *mobile* para resolver um problema bancário? Muitas pessoas também utilizam para conversar, via *streaming*, com um amigo que mora do outro lado do país. Toda essa facilidade e precisão das informações, enviadas e recebidas, só é possível com o desenvolvimento de softwares e aplicativos que tenham uma excelente gestão em sua engenharia de concepção e construção, alinhados a práticas de testes com métricas e normalizações que o mercado exige. Veremos, então, sobre esse tema fantástico!

## 1.2 A qualidade e sua importância


Refleta: será que a qualidade é importante até na construção de aplicativos? Onde e como empregá-la? Como sabemos que algo possui ou não a qualidade desejada? Para responder esses questionamentos, devemos, primeiramente, entender o que é qualidade. Para Pressman (2011):

No desenvolvimento de software, a qualidade de um projeto engloba o grau de atendimento às funções e características especificadas no modelo de requisitos. A qualidade de conformidade focaliza o grau em que a implementação segue o projeto e o sistema resultante atende suas necessidades e as metas de desempenho. (PRESSMAN, 2011, p. 359)

Desse modo, mensurar aquilo que já foi documentado torna-se um processo menos complicado, entretanto, não é tão simples, já que diversos fatores estão envolvidos, como características, funcionalidades e até a própria usabilidade. Para Pfleeger (2001, p. 657), “avaliar a qualidade de produto de um software vai muito além da preocupação com defeitos de funcionamento”.

Contar com o processo de qualidade, por equipes de desenvolvimento de software, pode ser um fator decisivo no sucesso ou fracasso do que se espera ter como produto, além de assegurar possíveis clientes. Isso tudo se deve ao fato do mercado estar mais atento e exigente ao que se tem de mais novo e confiável nesse meio, fazendo com que empresas adotem padrões e até modelos de maturidade, até então desnecessários, mas que se tornaram peça-chave na competitividade.

Segundo Pressman (2006), há uma conexão entre o grau de importância do software e suas exigências, com o papel que a engenharia de software representa, já que são necessárias técnicas e abordagens a altura daquilo que se quer produzir, fazendo com que a qualidade se equipare ao mesmo nível de suas cobranças.



Para auxiliar nessa busca por qualidade, a engenharia de software conta com mecanismos, sendo o processo de desenvolvimento de software um dos responsáveis por sua estrutura. Segundo Delamaro, Maldonado e Jino (2007), para se planejar, criar, desenvolver, implantar e manter um produto de software, são necessários políticas, estruturas organizacionais, procedimentos, artefatos e a tecnologia propriamente dita. Tudo isso para assegurar que a melhor técnica e procedimento empregado possa gerar um software com os padrões de exigência do mercado.

Cada etapa adotada, durante o processo de desenvolvimento de software, devolve para o próprio projeto implementações e revisões sistêmicas que fazem com que a aplicação se torne mais assertiva naquilo que foi planejado, ou seja, durante a fase de análise, com os requisitos funcionais e não funcionais. Entretanto, para que isso possa ocorrer de maneira coerente e seguindo o fluxo de trabalho, um bom planejamento é necessário, principalmente, quando falamos em avaliar a qualidade daquilo que foi desenvolvido e até daquilo que foi anteriormente planejado e documentado.

Outro ponto fundamental é poder contar com modelos de qualidade, que definem padrões de funcionamento, além de serem bem definidos e estruturados. Nesse aspecto, o modelo proposto pela ISO 9126 é o mais adequado e difundido na indústria, com divisões do produto software em qualidade interna, externa e em uso. Seu foco é na qualidade do produto, propondo atributos de qualidade, distribuídos em seis características principais, cada uma com suas subcaracterísticas.

Veja o Quadro 1, que apresenta as características e seus objetivos, além de citar as subcaracterísticas:



**Quadro 1 – Características ISSO 9126**

Característica principal:	Explicação:	Subcaracterísticas:
Funcionalidade.	Capacidade do software promover as funcionalidades que atendam as necessidades do usuário.	<ul style="list-style-type: none"><li>• Adequação.</li><li>• Acurácia.</li><li>• Interoperabilidade.</li><li>• Segurança.</li></ul>
Confiabilidade.	Software mantém nível de desempenho em condições estabelecidas.	<ul style="list-style-type: none"><li>• Maturidade.</li><li>• Tolerância a falhas.</li><li>• Recuperabilidade.</li></ul>
Usabilidade.	Capacidade do software ser entendido e compreendido da forma como foi desenvolvido e projetado.	<ul style="list-style-type: none"><li>• Inteligibilidade.</li><li>• Apreensibilidade.</li><li>• Operacionalidade.</li><li>• Atratividade.</li></ul>
Eficiência.	Verifica se a execução e os recursos utilizados são compatíveis com o nível de desempenho do software.	<ul style="list-style-type: none"><li>• Comportamento.</li><li>• Utilização de recursos.</li></ul>

Manutenibilidade.	Capacidade do software sofrer manutenção, ou seja, ser modificado com melhorias ou criação de novas funcionalidades.	<ul style="list-style-type: none"> <li>• Analisabilidade.</li> <li>• Modificabilidade.</li> <li>• Estabilidade.</li> <li>• Testabilidade.</li> </ul>
Portabilidade.	Capacidade do sistema ser portado, ou seja, migrado para outra plataforma e ambiente, por exemplo.	<ul style="list-style-type: none"> <li>• Adaptabilidade.</li> <li>• Capacidade para ser instalado.</li> <li>• Coexistência.</li> <li>• Capacidade para substituir.</li> </ul>

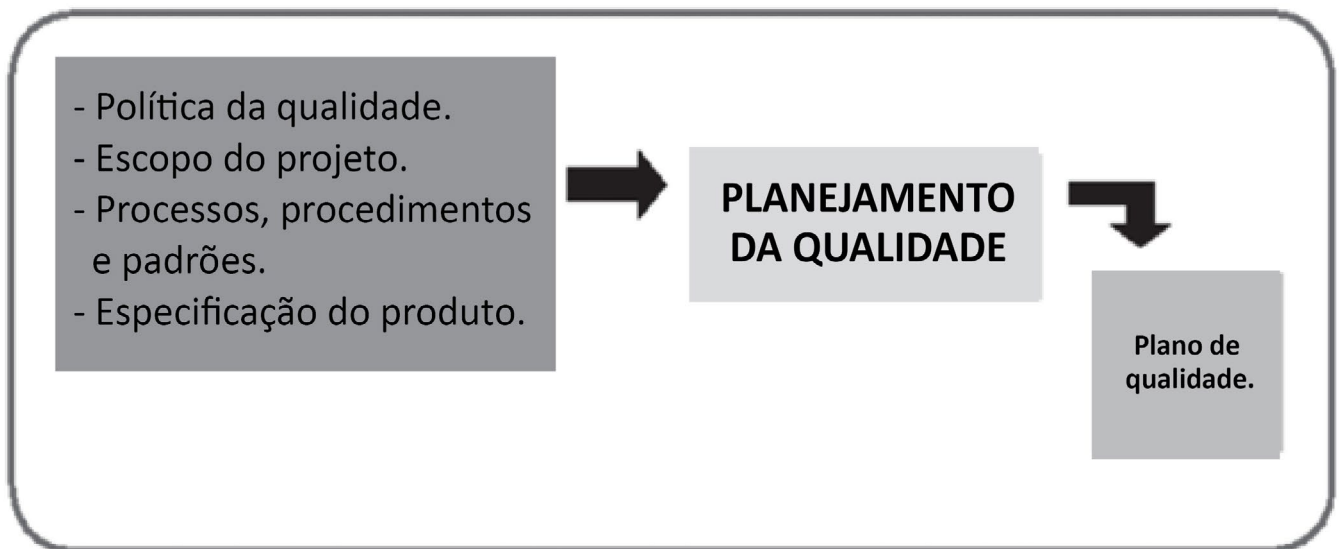
Fonte: ISO/IEC 9126 (1991).

Observe a Figura 1, abaixo, que mostra um modelo genérico a ser seguido, mas que pode ser enriquecido de acordo com a equipe ou o projeto, onde deve ocorrer o planejamento de qualidade de software, com o mínimo necessário para que algo possa ser analisado com destreza. Nesse sentido, devemos entender alguns pontos antes do planejamento propriamente dito, que envolve aquilo que foi definido como qualidade com o cliente, chamado de política de qualidade.

Depois, é de suma importância conhecer e entender o escopo do projeto, ou seja, quais as necessidades e os problemas que o sistema deve resolver para aquilo que foi planejando. Compreender os processos, procedimentos e padrões também são relevantes para o bom desenvolvimento, já que tornam a aplicação mais próxima da realidade. Por fim, mas não menos importante, conhecer as especificações do produto, para transparecer segurança e conhecimento daquilo que se

quer implantar e, desse modo, é essencial conhecer o que o sistema faz e o que não faz, principalmente para a qualidade.

**Figura 1 - Planejamento de qualidade**



Fonte: Vasconcelos *et al.* (2006, p. 83).

Importante também ressaltar que as técnicas de prevenção e detecção de erros não podem ficar de fora do projeto, já que tais situações podem ocorrer e nenhum software está isento de apresentar falhas, visto que a busca pela qualidade não é sinônimo de perfeição, mas de contínuo aperfeiçoamento daquilo que está se desenvolvendo.

Apenas planejar a qualidade não é suficiente, devemos garantir que essa qualidade ocorra de maneira satisfatória e ininterrupta, que tem por objetivo assegurar que o software possua os padrões básicos de qualidade, sendo segurança, eficiência, confiabilidade, integridade e até a usabilidade. Todos esses fatores, juntamente com algumas atividades a serem desenvolvidas, atestam a garantia do produto, focando, desse modo, na prevenção de defeitos e problemas que podem surgir, segundo Vasconcelos *et al.* (2006).

A Figura 2 retrata esse processo de garantia de qualidade de software, que exemplifica como deve ocorrer a busca pela mesma, retratando

genericamente os passos de como isso deverá ser feito. Com isso, as possibilidades para a melhoria do software já se tornam possíveis, uma vez que, após planejar e possuir o plano de qualidade, que é o escopo do projeto devidamente documentado e com todas as técnicas de prevenção, as métricas devem ser aplicadas, ou seja, realizar uma análise para medir os resultados daquilo que foi desenvolvido frente à realidade do usuário, fazendo com que haja a efetivação da garantia da qualidade.

**Figura 2 – Garantia de qualidade de software**

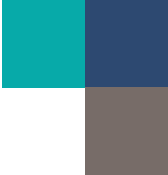


Fonte: Vasconcelos *et al.* (2006, p.83).

## ➤ 2. Documentação de testes de software

A busca incansável da engenharia de software está diretamente relacionada a entregar produtos cada vez mais competitivos no mercado, seja pela qualidade ali atribuída ou pelas novas tecnologias que diariamente surgem. Nesse universo plural, tudo é muito intuitivo e tecnológico, bem como as novas maneiras e concepções de desenvolver algo, além de padrões e processos que são reinventados com o passar do tempo.

A busca por uma documentação simples e clara também não fica de fora desse processo. Imagine que seja preciso realizar testes em um




determinado software e, para facilitar, recorreremos a documentação de tal sistema. Para surpresa, o que parecia fácil acabou ficando mais difícil. É exatamente isso que ocorre quando métricas e processos precisos não são utilizados durante o desenvolvimento do processo de software. Desse modo, tudo fica obscuro e impreciso, fazendo com que falhas e situações, que levem ao erro, se tornem inevitáveis. Diante disso, a padronização e utilização de métricas já testadas e aprovadas são de essencial importância para o bom desenvolvimento das atividades executadas, pois o processo como um todo é afetado, gerando, assim, uma falha setorial que pode inferir no resultado.

Nesse contexto, temos uma organização sem fins lucrativos, que é responsável por promover o conhecimento em diversas áreas, como, por exemplo, a computação. O *Institute of Electrical and Electronic Engineers* (IEEE) define diversos padrões e práticas presentes na engenharia de software, que contribuem significativamente, principalmente, para a documentação em testes de software. Esses testes partem de unitários até de aceitação, além de incluir documentos consistentes e adequados para definir, registrar e prover condições de análise dos resultados obtidos ao longo do processo.

Para Koscianski e Soares (2007, p. 360), a norma IEEE 829 “é um modelo que define o padrão de documentação onde as informações serão registradas”. Ao todo, existem oito documentos norteadores:

- a. Plano de testes.
- b. Especificação do projeto de teste.
- c. Especificação de caso de testes.
- d. Especificação do procedimento de testes.
- e. Relatório de transição de itens de testes.
- f. Relatório de log de teste.
- g. Relatório de incidentes.
- h. Relatório final (sumário).



Todos esses documentos têm o intuito de auxiliar a equipe de qualidade a mensurar, por meio de planos e especificações já conhecidas, o software a ser testado, garantindo, assim, um padrão de qualidade. O plano de teste, como o próprio nome diz, é um plano, que prepara para a elaboração e execução dos testes, juntamente com os itens e as funcionalidades a serem testadas, além de identificar as tarefas e os riscos associados.


Os documentos relacionados à especificação de testes têm por objetivo identificar os casos e os procedimentos de teste, ou seja, é o que determinará o que deve ser inserido ou não como entrada, bem como o resultado esperado. Também determinam como serão os passos para a execução do teste a ser realizado.

Para finalizar esse conjunto de documentos, os relatórios de testes são responsáveis por registrar os detalhes mais importantes, ou seja, situações e eventos que ocorreram durante a execução dos testes e que mereçam atenção. Também são registrados os resultados das atividades desenvolvidas, fazendo com que haja um histórico de todo o processo.

Por meio dessa divisão apresentada acima (plano, especificação e relatório de teste), a execução das atividades pode ser mais bem compreendida, já que há uma separação lógica em pré-teste, teste e registro apresentado. Além de ser válido para todo e qualquer tamanho e complexidade de software, já que a qualidade não está relacionada ao tamanho do produto, mas em entregar aquilo que foi planejado.

A norma IEEE 829 também apresenta um método para a implantação do processo de teste de software, que se dividem em:

- a. Guia para elaboração de documentos de teste de software.
- b. Processos para a elaboração de documentos de teste de software.



Enquanto o primeiro tem o objetivo de servir como referência para criação de documentos de teste, o segundo se preocupa com a preparação, a execução e o registro dos resultados do teste, estabelecendo uma orientação geral sobre o item a ser verificado.

Contudo, independentemente de quais documentos ou a forma como serão utilizados, é de extrema importância que haja planejamento, projeto, casos de teste e procedimentos de teste. Isso assegurará que o processo foi feito seguindo todas as métricas e que a documentação seguiu os padrões solicitados, assegurando, desse modo, a gestão da qualidade.

## **2.1 Casos de testes**

Assim como os passos anteriores pra a execução da documentação, outro fator importante é realizar o caso de teste, que é um conjunto de testes, ou seja, por meio dele, vários itens em uma determinada tela são averiguados, devendo sempre responder a perguntas como: o que será testado? O que será testado para determinar se o programa é bom ou não?

Sobre quais são os testes que o sistema deverá ser submetido, ou a melhor metodologia para isso, o (a) analista deverá decidir sobre e, posteriormente, repassar à equipe. Além disso, designará integrantes que podem ou não utilizar o mesmo caso para testar certa funcionalidade, ou até comparar os resultados obtidos em casos de testes distintos. Isso é importante porque cada um tem uma forma de realizar um processo e mesmo que a descrição do passo a passo a ser seguida seja a mesma, resultados diferentes podem ocorrer.

Observe Quadro 2, que retrata um modelo simples de um caso de teste tradicional, conforme o padrão IEEE 829, instituto internacional que regulamenta casos de testes. Note que, neste caso, o que se busca saber

é se um triângulo é realmente equilátero, ou seja, se possui as mesmas medidas em todos os lados.

### Quadro 2 – Modelo de caso de teste tradicional

Caso de teste	
ID:	TCS-111.
Nome:	Verificar um triângulo equilátero.
Ambiente:	RVM-1.87.
Ator:	Aluno de matemática.
Precondições:	Não existir nenhum triângulo antes da realização do teste.
Procedimento:	<ol style="list-style-type: none"><li>1. Abrir aplicação.</li><li>2. Solicitar a consulta a um tipo de triângulo.</li><li>3. O sistema deve solicitar as medidas do triângulo.</li><li>4. Informar 7 para o lado A.</li><li>5. Informar 7 para o lado B.</li><li>6. Informar 7 para o lado C.</li><li>7. O sistema retorna a informação de que o triângulo é equilátero.</li></ol>
Pós-condições:	Existir um triângulo equilátero.

Fonte: adaptado de Vasconcelos (2006).

Note que na coluna à esquerda foram elencados vários itens, que são passos e procedimentos para a execução do teste, sendo:

- ID: identificador do teste.
- Nome: descreve o que o teste fará.
- Ambiente: em qual local ou ambiente o teste deverá ser realizado. Nesse exemplo, RVM-1.87 refere-se à ferramenta *Ruby Manager*.
- Precondições: cenário que deve ou não existir antes da realização do teste. Fundamental para validar ou não o teste.



- Procedimento: execução do teste passo a passo.
- Pós-condições: resultado esperado após a execução do teste.


A partir dessas definições, a execução pode ocorrer com métricas definidas, fazendo com que o teste fique preciso e conciso, gerando, assim, relatórios e estatísticas satisfatórias, que auxiliarão na tomada de decisão.

Contudo, os casos de testes representam uma etapa fundamental na área da qualidade, além de possuírem diversas especificações, que podem se resumir em formais e informais. O primeiro, com foco na usabilidade do sistema, ou seja, focado nos objetivos que o sistema deve atender. O segundo tem um caráter mais revisório, ou seja, avalia as funcionalidades do sistema e se baseia em requisitos não funcionais.

### 3. O modelo TMMi

A indústria tecnológica tem crescido muito nos últimos anos, especialmente no setor de desenvolvimento de software, que exige novas metodologias e métricas de trabalho, especialmente na qualidade de seus produtos. Dessa maneira, modelos já consagrados pelo mercado, como *capability maturity model and integration* (CMMi) e melhoria de processo do software brasileiro (MPS.BR), vêm sendo utilizados como forma de melhoria dentro do processo de desenvolvimento. Tais modelos têm por objetivo medir a eficiência e eficácia de seus produtos desenvolvidos, elencando pontos fracos e fortes, bem como os que podem ser melhorados. É nesse meio que o processo de testes é inserido, ou seja, durante todo seu ciclo de desenvolvimento e após, com a manutenção do código.

Desse modo, mesmo com vários benefícios dos modelos CMMi e MPS.BR, a área necessitava de um melhor suporte quanto a avaliação



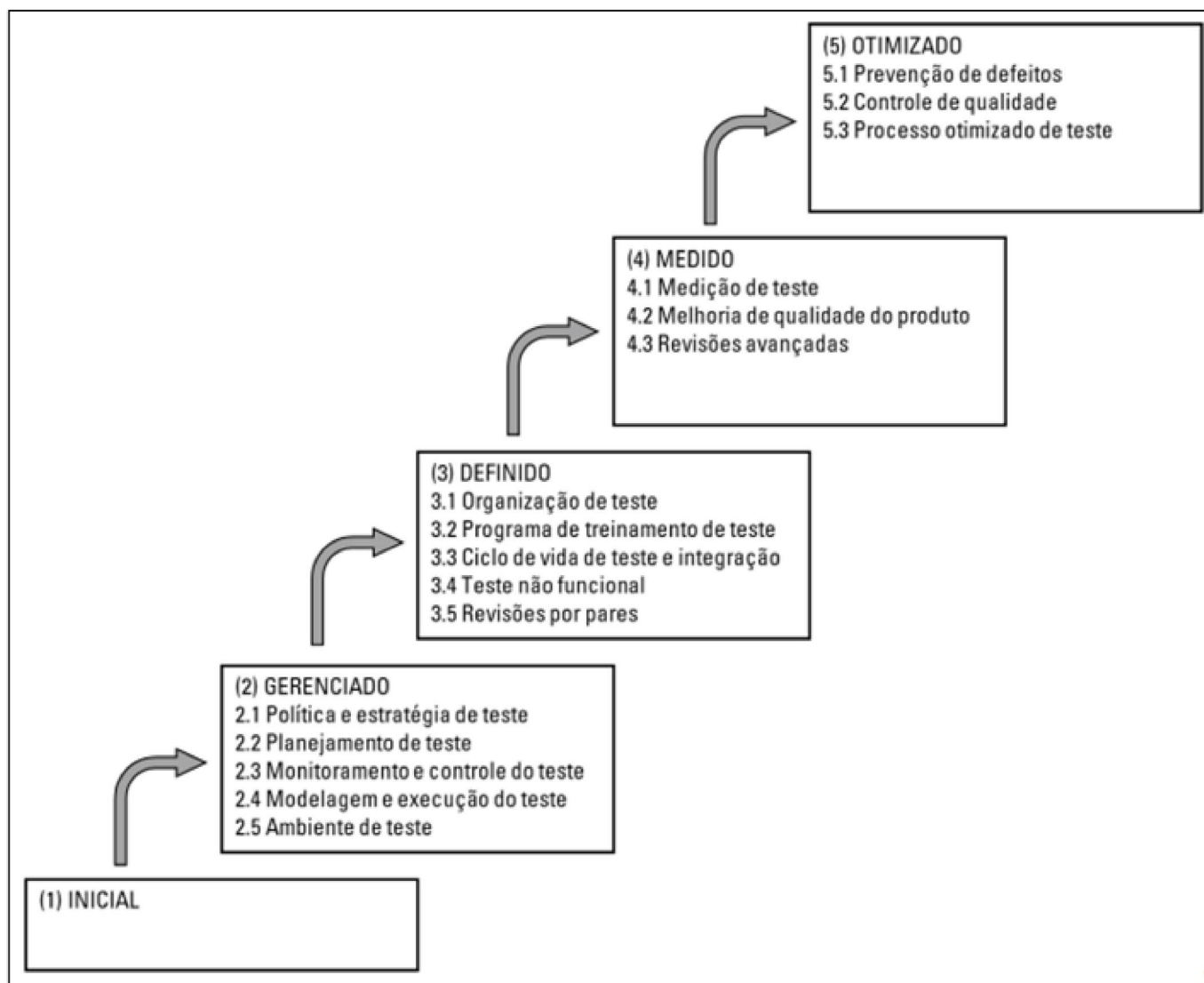
e melhoria de seus processos. Dessa forma, o TMMi *Foundation* desenvolveu o *test maturity model and integration* (TMMi), que vem para somar com o CMMi, que nasceu para implementação e melhoria dos testes de software. Esse modelo se baseia em classificações e níveis, ou seja, a organização vai se classificando e evoluindo dentro da atividade avaliada, podendo evoluir conforme o cumprimento de processos para atingir determinado nível de maturidade. Com a implantação do TMMi, o número de defeitos e falhas tende a diminuir antes, durante e após o ciclo de desenvolvimento, já que gera um produto com melhor qualidade.

Entretanto, assim como todo novo processo e metodologia, este deve ser bem estudado para a efetiva implementação, já que a equipe, inicialmente, pode se sentir desconfortável, visto que é algo novo e depende de entendimento, aceitação e um fluxo que efetivamente funcione, assim como a motivação da equipe em querer reorganizar algo já em andamento.

Estruturalmente, o TMMi é composto por práticas de teste que podem ser aplicadas e melhoradas, isso para dar suporte ao processo de teste de qualidade, principalmente, em etapas incrementais.

Ao todo, o TMMi é composto por cinco níveis que uma organização pode ser classificada, evoluindo conforme o nível gerencial e organizacional da mesma. Cada nível tem seus requisitos, que devem ser atingidos para que possa progredir para o próximo. Esse processo pode ser visto na Figura 3.


### **Figura 3 – Níveis de maturidade e áreas de processos do TMMi**



Fonte: TMMi (2020).

Como pode ser visto na estrutura acima, há a existência de níveis, áreas de processos, objetivos e práticas. Para que a organização possa avançar, ou seja, sair de um nível e progredir para o próximo, é necessário que todas as áreas de processos do referido nível estejam cumpridas. Essa verificação é possível por meio de uma avaliação, aplicada, e o TMMi *assessment method application requirements* (TAMAR) avaliará se isso é possível ou não.

Para que esse processo possa ser melhor compreendido, segue, resumidamente, o que cada nível de maturidade exige:

- 
- a. Nível 1 – Inicial: não existe uma métrica definida, desse modo, o processo de teste é caótico e faz parte da depuração do desenvolvimento. Testes são desenvolvidos de forma *ad-hoc* e somente executados após a finalização do código, não existindo diferença entre a prática de testes e a depuração do código. Como o objetivo é apenas provar que o software é executado, o produto pode não atender às expectativas e extrapolar o orçamento.
  - b. Nível 2 – Gerenciado: há um processo de teste gerenciado, que está separado da depuração do código, o que ajuda a garantir a integridade em períodos de estresse. Há uma estratégia de teste para toda a empresa, planos de teste, técnicas de gerenciamento de riscos, compromisso com os *stakeholders*, monitoramento e controle de testes são implementados neste nível. O objetivo deste nível é verificar se o produto satisfaz seus requisitos, entretanto, os testes ocorrem no momento tardio ao ciclo de vida do software.
  - c. Nível 3 – Definido: os testes passam a ser totalmente integrados ao ciclo de vida do desenvolvimento e aos marcos associados a ele. O planejamento de testes passa a ser realizado no momento inicial do projeto, com o plano de teste mestre, partindo do conhecimento do nível 2. Profissões da área de testes, como engenheiro de testes e analistas de testes, são reconhecidos, e um programa de treinamento em testes é definido. Neste nível, os métodos e técnicas de testes utilizados passam a cobrir também os testes não funcionais.
  - d. Nível 4 – Medido: neste nível, o processo de testes se torna bem definido, fundamentado e mensurado. Há uma leitura estatística quanto a qualidade do produto e o desempenho do processo de testes, já que há objetivos quantitativos para eles. Os produtos são avaliados, usando critérios quantitativos para atributos de qualidade, tais como confiabilidade, usabilidade e facilidade de manutenção. Neste nível, o teste também é percebido como uma avaliação que consiste em todas as atividades do ciclo de vida dos testes relacionadas à validação e verificação de um produto.

- e. Nível 5 – Otimizado: neste nível, o processo de teste é completamente definido e é capaz de controlar os custos e a efetividade dos testes. Há a melhoria contínua de seus processos, prevenção de defeitos e controle de qualidade são introduzidos como áreas de processo. Amostragem estatística, medições dos níveis de confiança e confiabilidade conduzem o processo de teste, além do uso de ferramentas que auxiliam durante os testes.

Contudo, pode-se notar que a estrutura do TMMi é rica em processos e competências que devem ser alcançadas para que o próximo nível seja possível. Todo esse procedimento tende a criar software com um nível alto de qualidade, fazendo com que o usuário não seja afetado com problemas ou falhas que poderiam ser evitadas ou até inexistirem durante seu desenvolvimento.

## Referências Bibliográficas

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **NBR ISO/IEC 9126-1**. Engenharia de software – Qualidade do produto. Parte 1: modelo de qualidade. Disponível em: [https://jkolb.com.br/wp-content/uploads/2014/02/NBR-ISO\\_IEC-9126-1.pdf](https://jkolb.com.br/wp-content/uploads/2014/02/NBR-ISO_IEC-9126-1.pdf). Acesso em: 2 out. 2020.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao teste de software**. Rio De Janeiro: Campus, 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, 2007.

PFLEEGER, S. L. **Software engineering**: theory and practice. 2. ed. New Jersey: Prentice Hall, 2001.

PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. 7. ed. Porto Alegre: AMGH, 2011.

PRESSMAN, R. **Software engineering – A practitioner´s approach**. 6. Ed., McGraw-Hill Professi, 2004.

TMMI FOUNDATION. Disponível em: <https://www.tmmi.org/>. Acesso em: 29 set. 2020.



VASCONCELOS, A. M. L. de *et al.* **Introdução à engenharia de software e à qualidade de software.** Tese de pós-graduação, curso de Melhoria de Processos de Software, Fundação de Apoio ao Ensino, Pesquisa e Extensão, Universidade Federal de Lavras. Lavras, 2006.

# Gerenciamento dos testes de software com ferramentas *CASE open source*.

Autoria: Luís Otávio Toledo Perin

Leitura crítica: Rafael Maltempe da Vanso



## Objetivos

- Definir o que é teste de software.
- Compreender o processo de teste de software.
- Apresentar ferramentas *CASE open source* para testes de software.



## 1. Teste de software

### 1.1 Introdução

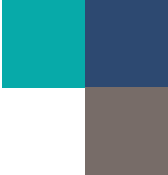
Neste tema, o assunto tratado será gerenciamento dos testes de **software**, por meio de ferramentas CASE *Open Source*. Testar parte de um processo ou todo o produto não é mais uma opção, mas um requisito a ser cumprido. Diante do aumento de empresas de TI, seja porque estamos em um mundo mais tecnológico ou porque as pessoas estão consumindo mais tecnologia em seu cotidiano, testar, com métricas e padrões aquilo que se produz, é indispensável para que o usuário não enfrente problemas em sua utilização, além de garantir a maturidade do software desenvolvido.

Quando se fala em testar o software, o que você pensa? Um grande algoritmo complexo que realize essa ação? Uma verificação manual de tela por tela ou função por função? Uma aplicação que demanda muito tempo e esforço para ser compreendida? Pensa uma aplicação recém-lançada que mereça atenção? Pode ser tudo isso e mais um pouco, ou apenas parte disso. Esse universo de testes e situações demanda uma atenção muito grande, apesar de ser relativamente simples, pois o fato de testar algo é atestar que aquilo vai funcionar quando estiver em produção, ou seja, que o cliente ou o usuário vão poder usufruir de todos os recursos de maneira satisfatória, sem apresentar problemas que o impeça de realizar o que precise.

### 1.2 A qualidade

Antes de entrar no assunto teste de software, reflita: apenas testar por testar é válido? Há alguma utilidade em valer-se de inúmeros testes, a fim de gerar relatórios estatísticos ou parâmetros de comparação, se não é considerado o fator qualidade? Caso isso ocorra, todos os dados





gerados não têm serventia, visto que o teste de software, por si só, não consegue resolver os problemas, mas mostrar onde estão as falhas e prever situações para que possam ser corrigidos e aprimorados, respectivamente.

Todo o ciclo de desenvolvimento de um projeto de sistema de informação passa por várias etapas até chegar a produção, ou seja, a versão final para que o usuário possa utilizá-lo. Dentro dessas etapas, existem métricas e métodos que devem ser seguidos, além de ferramentas que auxiliam a equipe nas diversas fases, como análise, prototipação, desenvolvimento, testes e implantação. Entretanto, mesmo com a utilização de tais recursos, erros ainda podem ocorrer, ocasionando, assim, a elevação de custos, o atraso de prazos ou até mesmo o insucesso do produto, segundo Pressman (2016).

Com o intuito de minimizar tais possibilidades, práticas de VV&T (verificação, validação e teste), são executadas durante todo o ciclo de desenvolvimento do software, garantindo, assim, que a menor quantidade de falhas possa ocorrer. De acordo com Sommerville (2018):

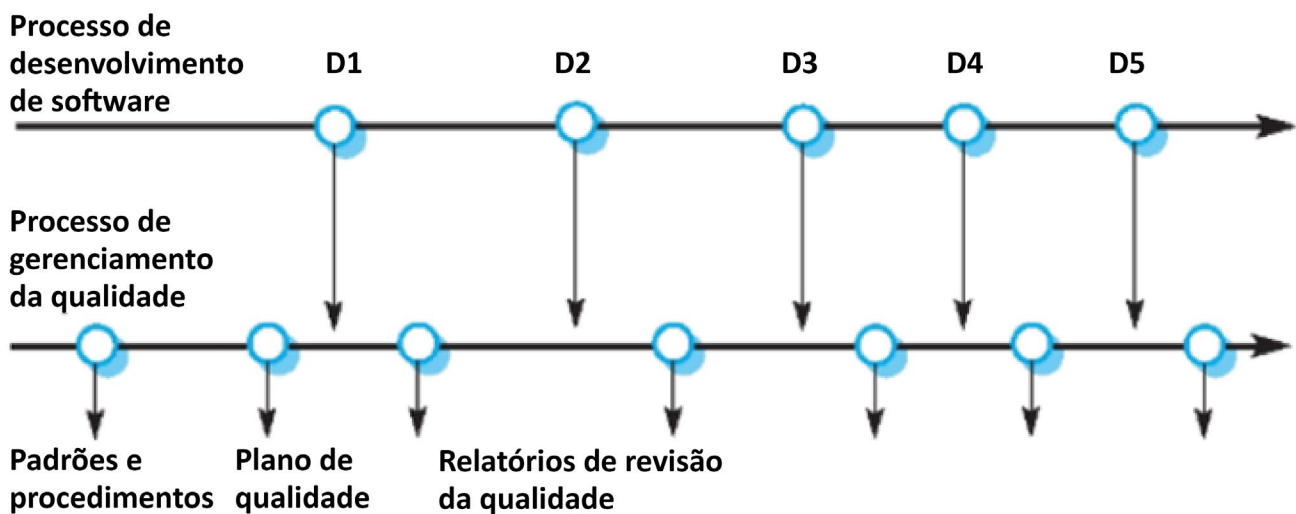
Na indústria de software, algumas empresas veem a garantia de qualidade como a definição de procedimentos, processos e padrões para garantir que a qualidade do software seja atingida. Em outras empresas, a garantia de qualidade também inclui todas as atividades de gerenciamento de configuração, de verificação e de validação aplicadas depois que um produto foi entregue por um time de desenvolvimento. (SOMMERVILLE, 2018, p. 664)

No quesito qualidade, o termo garantia é muito valorizado, já que o processo de verificação e certificação será efetuado nesta fase. Metodologias cada vez mais precisas e elaboradas têm feito com que a qualidade do produto desenvolvido tenha valor agregado, ou seja, produzir com padrões e métricas vai além de se certificar que o produto passou por um rigoroso controle de qualidade, com muitas etapas e

checagens, mas de saber que o cliente poderá utilizar a aplicação com segurança.

De maneira independente ao desenvolvimento do software, o gerenciamento da qualidade passa a verificar se os requisitos foram fielmente cumpridos, certificando-se, assim, dos padrões e metas organizacionais propostos. Paralelamente a isso, avalia a documentação gerada por cada time durante o processo, garantindo, assim, a integridade da informação entre as equipes, além do produto concebido, segundo Sommerville (2018). Observe a Figura 1, que esboça o descrito acima.

**Figura 1 – Gerenciamento da qualidade e desenvolvimento de software**



Fonte: Sommerville (2018, p. 664).

Entretanto, para que haja essa independência, é preciso também que as equipes ou times sejam independentes, ou seja, desenvolvimento e qualidade não podem estar interligados quanto a pessoal ou subordinação. Esse é um fator fundamental para que o processo ocorra com a maior transparência e tenha uma visão objetiva sobre aquilo que está se testando.

Outro fator que pode afetar a qualidade, já que é considerado um dos procedimentos mais caros dentro do ciclo de desenvolvimento de software, é o cronograma de entrega do produto, segundo Sommerville (2018). Para mantê-lo, o gerente do projeto pode suprimir alguns processos e métricas para cumprir o cronograma, fazendo com que a equipe de qualidade não atinja todas as metas organizacionais desejáveis. Essa atitude tende a comprometer toda a integridade e segurança do sistema, visto que não foi amplamente testada e está suscetível a falhas que poderiam ser evitadas.

Contudo, entre todas as boas práticas para obter a qualidade daquilo que se deseja, o teste se destaca, já que é o responsável por avaliar se os requisitos elencados com o cliente estão sendo atendidos ou não, só sendo possível com padrões preestabelecidos.

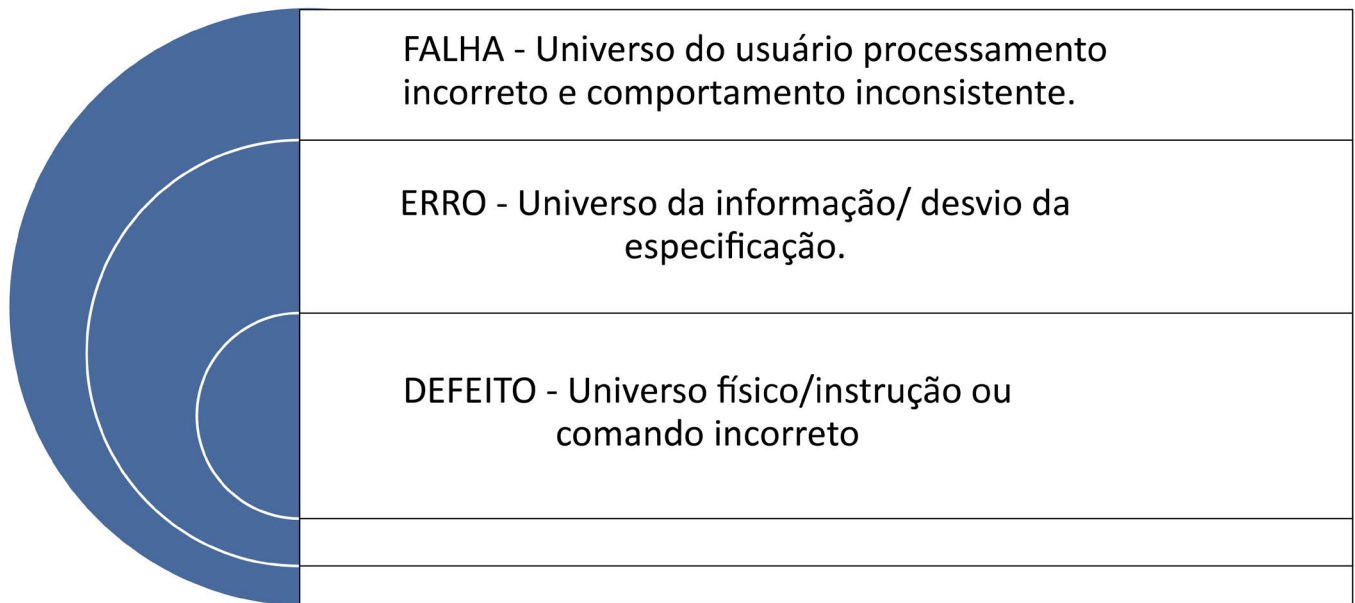
## 2. O processo de teste de software

O processo de teste de software visa elaborar métricas para que essa atividade possa ser bem executada. Antes, reflita: já solicitaram a você para testar alguma coisa? Pode ser um software ou qualquer outra coisa. Provavelmente sim, e podem ter sido feitos, ao menos, esses dois questionamentos: como devo testar? Qual o resultado esperado?

Questões como essas são comuns e importantes para nortear aquilo que se deseja testar, já que o teste tem por finalidade averiguar se aquilo que foi projetado para ser feito realmente está de acordo, além de descobrir problemas ou defeitos antes do usuário, garantindo, assim, o entendimento e efetivação do processo. Durante essa atividade, dados fictícios são inseridos no software, e os resultados obtidos são utilizados como parâmetros em busca de anormalidades.

Quando falamos em testar algo, logo surgem as palavras, defeito, erro ou falha. Todas representam um problema que deve ser solucionado, entretanto, cada um tem uma tratativa específica, assim como mostra a Figura 2.

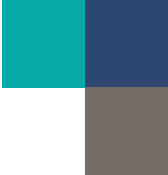
**Figura 2 – Defeito versus erro versus falha**



Fonte: elaborada pelo autor.

Segundo o *International Software Testing Qualifications Board* (ISTQB), temos o erro como falha humana, que produz um resultado incorreto, podendo ser um erro de sintaxe, por exemplo. O defeito, por sua vez é resultado de um código mal escrito, causando mau funcionamento no sistema, conhecido também como bug. Por fim, a falha é resultado de um código que apresenta defeito e foi executado, causando funcionamento incoerente das funções do software, sendo o usuário diretamente atingido, já que é visível ao mesmo.

O testador de software busca, com esse processo, basicamente duas coisas: demonstrar que o software atende aos requisitos solicitados pelo cliente; e falhas ou erros em processos ou situações desenvolvidas durante a criação dele. Para o primeiro caso, há uma divisão entre



software customizado e implementações genéricas, onde o primeiro deve possuir, ao menos, um teste para cada customização, já no segundo, para cada característica implantada. Quanto a falhas ou erros durante o processo de criação, devem ser identificados e repassados ao desenvolvimento, já que dizem respeito ao processo em si, ou seja, houve um erro no algoritmo e deve ser corrigido.

## 2.1 Tipos de testes

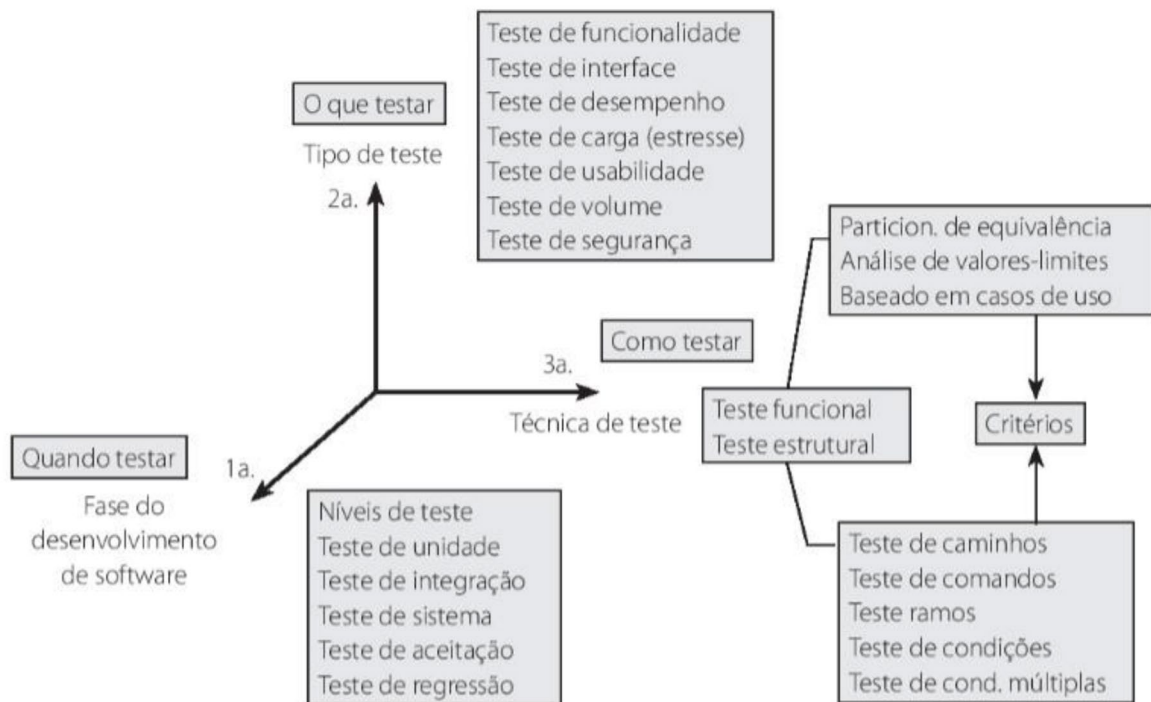
Testar um software é importante para seu bom funcionamento, ou seja, para que haja qualidade, bem como para que o cliente não tenha problemas ou contratempos com sua utilização. Agora, veremos sobre o ato de testar demanda técnica, e que existem diversos testes e formas que podem ser executados, cada qual com seu objetivo e finalidade específica.

Há uma grande divisão nos testes, que se liga a relação temporal, ou seja, o momento mais adequado para determinado teste serem executados, segundo Crespo *et al.* (2004). Isso porque dependendo da fase em que o produto se encontra, certo teste é mais adequado e rápido de ser executado, além de se considerar que o erro pode ocorrer a qualquer momento e, por esse motivo, devem ser efetuadas validações constantes.

Observe a Figura 3, que representa a relação entre os níveis, tipos e técnicas de testes. Alguns direcionamentos devem ser efetuados nesse processo, como quando, o que e e como testar. Todas as técnicas norteiam quanto às boas práticas de teste, já que estão divididas entre as fases do ciclo de desenvolvimento do software, como o desenvolvimento propriamente dito, os tipos de testes existentes e as técnicas que cada uma demanda. Todo esse processo deve se repetir antes, durante e após a concepção do produto, garantindo, assim, que o modelo foi corretamente executado, além de atribuir maturidade e

longevidade a aplicação, visto que os processos e técnicas tendem a evoluir com o tempo, algo que favorece a manutenção do software.

**Figura 3 – Relação entre níveis, tipos e técnicas de teste**



Fonte: Crespo *et al.* (2004, p. 4).


Sabemos da importância de todos os tipos de testes, mas sua utilização, bem como o momento ideal para que isso ocorra é algo que deverá ser determinado pelo analista, que tem conhecimento técnico para a tomada de decisão.

Por existir uma infinidade de testes, vamos direcionar essa leitura apenas para os principais: unitário, de integração, de sistema, de aceitação, de regressão, de integridade de dados, de configuração e instalação, de performance e não funcionais. Abaixo, você verá um resumo das principais funcionalidade de cada teste.

- Teste unitário: trabalha com a menor parte do software e é um dos mais básicos da lista. Tem por objetivo avaliar cada módulo do sistema de maneira individual, averiguando se sua funcionalidade

- está correta. Também pode ser feito em trechos de códigos específicos, segundo Wazlawick (2011).
- b. Teste de integração: seu foco é avaliar se a comunicação entre os módulos da aplicação é feita do modo certo. Agrupa os módulos com funções específicas, formando, assim, um sistema completo, e, dessa maneira, integra e testa, avaliando se há erros de interface entre eles, segundo Wazlawick (2011).
  - c. Testes de sistema: após a integração do sistema, já é possível tê-lo por completo e, assim, simular o ambiente de uso final, ou seja, do usuário. Desse modo, pode-se averiguar a existência de falhas em aspectos gerais do sistema, além da qualidade da interface gráfica, o funcionamento das funções do sistema e se os requisitos elencados no começo do projeto foram cumpridos, segundo Wazlawick (2011).
  - d. Teste de aceitação: similar aos testes de sistema, mas se diferencia por estabelecer uma relação focada entre o usuário final e o programa, ou seja, averiguar se há aceitação do público com aquilo que o programa executa. Geralmente, se utiliza de um grupo de usuários selecionados para esse tipo de verificação, em versões de software do tipo teste ou *beta*, que tem por objetivo avaliar a usabilidade de um produto por usuários finais, entretanto, com a aplicação não oficial. Podem também se subdividir em: teste de aceitação formal, teste de aceitação informal ou teste *beta*.
  - e. Teste de regressão: basicamente, é uma replicação dos testes vistos anteriormente, mas com em nova versão do sistema. Essencial para averiguar a existência de novos erros após os anteriores terem sido identificados e tratados, já que uma correção pode afetar outras áreas do sistema. Portanto, este teste é uma nova verificação, uma espécie de avaliação mais rigorosa no sistema.
  - f. Teste de integridade de dados: focado no processo de validação da confiabilidade e integridade dos dados de um software. É



- 
- realizado em diversas fases do desenvolvimento para averiguar a robustez do programa, ou seja, sua capacidade de resistir a falhas.
- g. Teste de configuração e instalação: ambos os testes avaliam a capacidade do software em funcionar corretamente em diversas plataformas. Mais especificamente, o de instalação avalia possíveis erros em vários cenários de instalação, como hardware, espaço em disco, entre outros. Já o de configuração, avalia a capacidade do programa ser executado e ofertar todos os recursos em diferentes plataformas.
  - h. Teste de performance: destinado a avaliar a performance da aplicação, ou seja, o nível de excelência daquilo que foi desenvolvido. Neste teste, existem três subdivisões, sendo: teste de estresse, de carga e de estabilidade. O primeiro diz respeito a expor o sistema a ambientes aos quais chega próximo à sua capacidade de projeto, ou seja, os requisitos mínimos, como memória, sistema operacional, entre outros. O segundo, trata da capacidade do software em relação aos dados ali inseridos, ou seja, avaliar os limites que consegue operar de modo constante e normal. Por fim, o de estabilidade visa avaliar o software no quesito temporalidade, ou seja, por quanto tempo um sistema se mantém estável, funcionando de forma normal e correta, sem sofrer anomalias e alterações imprevistas.
  - i. Testes não funcionais: testes diretamente relacionados aos requisitos não funcionais. Podemos citar como exemplo os testes de performance, confiabilidade, usabilidade e recuperação, entre outros. Todos têm um único objetivo, assegurar a funcionalidade e segurança para o usuário que utiliza o sistema testado.

O Quadro 1 tem por objetivo, mostrar, de maneira resumida os tipos de testes citados acima, além de sua funcionalidade.



## Quadro 1 – Tipos de testes e suas funcionalidades

Teste	Funcionalidade
Unitário.	Avaliar cada módulo do sistema de maneira individual, averiguando se sua funcionalidade está correta.
De integração.	Avaliar se a comunicação entre os módulos da aplicação é feita do modo correto.
De sistema.	Com o software integrado, simular o ambiente de uso final, ou seja, do usuário
De aceitação.	Averiguar se há aceitação do público com aquilo que o programa executa.
De regressão.	Após os erros encontrados serem corrigidos e uma nova versão do sistema ser gerada, novos testes são executados para eliminar possíveis falhas.
De integridade de dados.	É realizado em diversas fases do desenvolvimento para averiguar a robustez do programa, ou seja, sua capacidade de resistir a falhas.
De configuração e instalação.	De configuração avalia a capacidade do programa ser executado e ofertar todos os recursos em diferentes plataformas. O de instalação avalia possíveis erros em vários cenários de instalação, como hardware, espaço em disco, entre outros.
De performance.	Avalia a performance da aplicação, ou seja, o nível de excelência daquilo que foi desenvolvido.
Não funcionais.	Relacionado aos requisitos não funcionais.

Fonte: elaborado pelo autor.

Como visto até o momento, quando falamos em testar uma aplicação, estamos indo além de apenas utilizar o sistema por um tempo e verificar se há ou não erros. Há métricas e técnicas de devem ser seguidas, além de cada situação a ser testada demandar um tipo específico de teste, fazendo com que o processo passe a ser padronizado e confiável.



### 3. Ferramentas CASE *open source*

Desde sua existência, a engenharia de software tem como meta aprimorar seus processos e etapas dentro do ciclo de desenvolvimento do software. Isso se torna visível com as diversas técnicas e práticas já mencionadas nesta leitura. No quesito teste de software, o esforço está na prática com mais agilidade, eficácia e confiabilidade em sua execução, garantindo, assim, o funcionamento adequado do sistema.

Para minimizar os problemas durante a execução dos testes, ferramentas que auxiliam o trabalho são necessárias, diminuindo, assim, as possibilidades de erros em sua execução, bem como melhor captura daquilo que se deseja analisar. Também estão relacionadas ao avanço que a engenharia de software tem buscado ao longo dos últimos anos, ou seja, melhoria contínua do processo, segundo Sommerville (2018).

Outro fator importante para a utilização de ferramentas e padronizações de práticas no ambiente de teste de software diz respeito ao custo, já que tendem a reduzir o valor do processo como um todo, segundo Maldonado *et al.* (1998). Esse fator deve ser levado em consideração pelo gerente de projeto, já que o controle de qualidade exige alto custo, se comparado com as etapas anteriores, isso pelos inúmeros processos e métricas que são exigidos.

Para que haja essa redução, ferramentas *open source*, podem e devem ser utilizadas durante as fases de teste. O termo *open source* não diz respeito apenas ao seu código livre para modificações e leitura do público em geral, mas também à padrões que a comunidade exige, segundo OSI (2012).


### 3.1 Ferramentas mais utilizadas

Sabemos que o mercado tecnológico é vasto em software e apresenta infinitas possibilidades, partindo de aplicações mais simples até as mais robustas. Nesse aspecto, foram separados alguns sistemas que auxiliam a equipe de teste de software neste processo. Vejamos:

- a. TestLink: seu foco é auxiliar o testador durante a realização do teste de software, bem como manter uma organização daquilo que já foi verificado. Sua criação data de 2005 e, atualmente, mantida pela *TestLink Community*, sendo desenvolvida em linguagem web PHP, possuindo código aberto (*open source*). O fato de ser web torna sua utilização mais expansiva, ou seja, equipes espalhadas em diversos locais do país e até do mundo podem colaborar com o projeto, favorecendo, assim, o dinamismo entre os envolvidos, segundo Caetano (2008).

O analista de teste ou gerente que optar pela ferramenta, terá à disposição a possibilidade de criar e executar casos de testes para sua equipe, favorecendo, desse modo, o gerenciamento de projetos. Também é possível elencar os testes que foram bem sucedidos ou não durante sua realização, com a possibilidade de gerar relatórios, fato que servirá como parâmetro para futuras análises. Por fim, de maneira administrativa, é possível gerenciar as permissões de acesso de cada usuário, garantindo que cada um tenha acesso apenas a sua atividade desenvolvida, segundo Bittencourt (2013) e Reis (2012).

No quesito integração com ferramentas externas de gerenciamento de defeitos (*bug tracking*), o *TestLink* mostra-se muito colaborativo, visto que trabalha de forma integrada com ferramentas como *Mantis* ou *Bugzilla*, por exemplo (Testlink, 2020).

- 
- b. Rth: similar a anterior, também possui foco na análise de testes, ou seja, permite a criação de casos de testes, além de contar com o monitoramento da execução de cada atividade ali pensada e desenvolvida pelo analista.

Sua licença é *General Public License* (GNU) e é construída em linguagem PHP, sendo baseada na web, desse modo, ao utilizá-la de maneira on-line, a integração com membros da equipe pode ser feita de remotamente, gerando mais uma opção de trabalho.

Como diferencial, permite exportar os resultados obtidos para arquivos com extensão *.x/sx*, já que não possui uma integração com ferramentas externas, o que pode facilitar a importação para sistemas de terceiro ou geração de relatórios pontuais.

- c. TestMaster: diferente das citadas acima, essa ferramenta é baseada na web, mas escrita em PERL e utilizando o Apache como servidor web. Foi criada com o intuito de auxiliar o processo de teste de software, tendo como foco a criação e execução de casos de testes, que podem ser mapeados durante a execução, sendo identificados como bem-sucedidos ou não, além de emitir estatísticas e relatórios daquilo que foi testado e do resultado obtido.

Como diferencial, destacam-se funções de automação de teste, que podem contribuir e muito com a assertividade e agilidade durante o processo. Também é possível realizar a importação e exportação de informações para documentos *.doc* e arquivos do tipo *Comma Separated Values* (CSV).

Contudo, nota-se a importância da utilização de ferramentas para a realização de casos de testes ou para o gerenciamento do processo de testes. São de suma importância para que o procedimento possa ficar informatizado e menos suscetível a falhas, já que um software

gerenciador realizaria essa organização, bastando apenas ser alimentado e utilizado para a extração da informação que se pretende.

## Referências Bibliográficas

BITTENCOURT, M. **Gestão de teste e defeitos com Testlink e Mantis**. 1. ed., v.1, 2013.

CAETANO, C. Gestão de teste: ferramentas open source e melhores práticas na gestão de teste. **Revista Engenharia de Software Magazine**, ano I, 3. ed., Brasil, 2008.

CRESPO, A. N. *et al.* **Uma metodologia para teste de software no contexto da melhoria de processo**. Centro de Pesquisas Renato Archer (CenPRA), Divisão de Melhoria de Processos de Software (DMPS). Campinas, 2004. Disponível em: [https://www.researchgate.net/publication/237497188\\_Uma\\_Metodologia\\_para\\_Testes\\_de\\_Software\\_no\\_Contexto\\_da\\_Melhoria\\_de\\_Processo](https://www.researchgate.net/publication/237497188_Uma_Metodologia_para_Testes_de_Software_no_Contexto_da_Melhoria_de_Processo). Acesso em: 5 out. 2020.

MALDONADO, J. C.; VINCENZI, A. M.; BARBOSA, E. F. *et al.* **Aspectos teóricos e empíricos de teste de cobertura de software**. VI Escola de Informática da SBC-Regional Sul, 1998.

OPEN SOURCE INITIATIVE (OSI). **Open Source Initiative**, 2012. Disponível em: <https://opensource.org/>. Acesso em: 5 out. 2020.

PRESSMAN, R. **Engenharia de software**. São Paulo: McGraw-Hill, 2006.

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

REIS, M. F. S. **VVTESTE**: ambiente de geração e gerenciamento de testes e de defeitos como apoio aos processos de verificação e validação do MPS. 2012.


REQUIREMENT SAND TESTING HUB. **RTH Blog**: RTH – Requirements and Testing Hub. Disponível em: <https://requirementsandtestinghub.wordpress.com/>. Acesso em: 29 set. 2020.

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

TESTLINK. **TestLink Open Source Test Management**. Disponível em: <http://testlink.org/>. Acesso em: 29 set. 2020.

TESTMASTER. **Testmaster-Source Forge**. Disponível em: <http://testmaster.sourceforge.net/>. Acesso em: 29 set. 2020.

TORRES, G.; BRAGA, P. **Open source**: uma nova desconstrução de paradigmas na educação moderna. Universidade Federal de Minas Gerais, 2015. Disponível em:



<http://www.periodicos.letras.ufmg.br/index.php/ueadsl/article/view/8629>. Acesso em: 01 agos 2020.

WAZLAWICK, R. S. **Análise e projeto de sistemas de informação orientados a objetos**. 2. ed. Rio de Janeiro: Campus, 2011.

# Procedimentos para garantir a segurança do software

Autoria: Luís Otávio Toledo Perin

Leitura crítica: Rafael Maltempe da Vanso



## Objetivos

- Definir o que é segurança de *software*.
- Compreender o processo de segurança de *software*.
- Apresentar normas de segurança de *software*.



## 1. Segurança no software

### 1.1 Introdução

Neste tema, veremos sobre os procedimentos para garantir a segurança do software. Além disso, as informações aqui contidas permitirão a atualização sobre o assunto, essencial na era digital em que vivemos, bem como aprimoramento de seus conhecimentos.

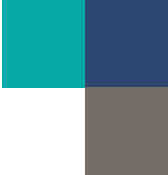
Já parou para pensar que toda vez que acessamos algum site, aplicativo ou até um sistema, estamos expostos a todo tipo de perigo? O que podemos fazer para melhorar nossa segurança? Será que deixar de acessar ou restringir o acesso seria uma alternativa de prevenção? Essa ideia seria pouco usual, já que controlar o acesso não é a melhor solução. Então, o que pode ser feito para que a segurança no software seja melhorada? Que práticas ou técnicas podemos aderir para tornar nossas aplicações mais confiáveis a quem irá utilizá-las? Perguntas como essas serão respondidas ao decorrer de nosso texto.

Sempre que desenvolvemos algo, ou alguma coisa em específica, devemos pensar no produto e em sua utilização, ou seja, em quem utilizará e como aquilo será benéfico para o ambiente. Todo o ciclo de desenvolvimento de software deve estar voltado, único e exclusivamente, para projetar e conceber aplicações que facilitem a vida do usuário, além de possuir um nível satisfatório de segurança, já que dados estão em jogo e a proteção deles é importantíssima para a continuidade dos negócios.

### 1.2 Qualidade e segurança

Devemos entender que a qualidade e segurança devem caminhar lado a lado durante o ciclo de desenvolvimento de software, visto que técnicas





e métricas bem aplicadas, ou não, durante a gestão da qualidade afetam o quesito segurança, pois repercutem na usabilidade final do produto.


Para McGraw (2005) essa relação é algo que complementa o software:

A segurança de software se relaciona inteira e completamente à qualidade. Devemos nos preocupar com a segurança, a confiabilidade, a disponibilidade e a dependência — nas fases inicial, de projeto, de arquitetura, de testes e de codificação, ao longo de todo o ciclo de vida [qualidade] de um software. Até mesmo pessoas cientes do problema de segurança têm se concentrado em coisas relacionadas ao ciclo de vida do software. O quanto antes descobrirmos um problema de software, melhor. E existem dois tipos de problemas de software. O primeiro são os bugs, que são problemas de implementação. Os demais são falhas de software — problemas arquiteturais do projeto. As pessoas prestam muita atenção aos bugs, mas não o suficiente às falhas. (MCGRAW, 2005, p. 369)

Em um mundo cada vez mais on-line, ou seja, com sistemas interligados a diversas aplicações e com dados disponíveis na nuvem (servidores e serviços na web), a preocupação com a segurança deve ser redobrada, já que a rede mundial de computadores é aberta, o que demanda cuidados extras. Nessa dinâmica, o software que possui alta qualidade está mais suscetível a cópias, ou seja, ser pirateado. Por outro lado, o que apresenta baixa qualidade pode sofrer com a segurança, expondo seus dados e informações a terceiros, segundo Pressman (2011).

Focar na qualidade é essencial para eliminar as falhas de um sistema e, por consequência, torná-lo mais seguro ao seu usuário. Esse é um dos motivos que a arquitetura de software é tão importante durante o ciclo de desenvolvimento, já que, eliminando falhas arquiteturais, se garante a melhora da qualidade, assegurando que os invasores tenham mais dificuldade em acessá-lo.

Para toda e qualquer atividade a ser desenvolvida, a utilização de padrões, métricas e métodos são importantes para se manter uma



organização dentro daquilo que se esteja trabalhar. No quesito desenvolvimento de software, a linha de pensamento é a mesma, ou seja, a adesão às métricas, para que haja um melhor aproveitamento no ciclo de desenvolvimento de software, é de fundamental importância para a competitividade e permanência no mercado, além de garantir a integridade e confiabilidade durante o processo de concepção do produto.

Qual a melhor metodologia ou prática de segurança a ser escolhida? Para Ferreira e Araújo (2008) são várias as possibilidades e depende da necessidade da empresa em selecionar qual será a que melhor se adeque ao modelo de negócio trabalhado. Outro fator que deve ser levado em consideração na escolha, diz respeito a tecnologia empregada, já que este é um fator muito flutuante, visto que há atualizações constantes nessa área.

Neste aspecto, a segurança é de fundamental importância para a integridade dos dados que o sistema deverá receber. Dessa forma, algumas propriedades básicas devem ser seguidas, como menciona Oliveira (2001, p. 9):

- a. Confidencialidade: dados e informações confidenciais devem estar protegidos de acesso não permitido ou tentativa dele.
- b. Disponibilidade: garantir que informações e serviços estejam disponíveis quando requeridos.
- c. Integridade: garantir a manutenção de informações e sistemas computadorizados, com total integridade dos dados ali presentes.

Além das propriedades citadas acima, Azevedo (2008) inclui novas, fazendo com que o quadro se torne ainda mais completo e seguro:

- a. Autenticação: capacidade de garantia que determinado usuário ou informação alimentada seja realmente verifica.


- b. Não repúdio: capacidade do sistema comprovar que um usuário executou determinada ação no sistema.
- c. Legalidade: capacidade do sistema estar de acordo com a legislação vigente.
- d. Privacidade: capacidade do sistema manter a privacidade do usuário em suas ações, impossibilitando a ligação direta da identidade do usuário com as ações, por este, realizadas.
- e. Autoria: capacidade do sistema validar e comprovar as ações realizadas no sistema por determinado usuário, ou seja, auditar tudo o que foi realizado pelo usuário, detectando fraudes ou tentativas de ataque.

Com o intuito em auxiliar as empresas no gerenciamento e na organização, padrões reconhecidos mundialmente foram criados, sendo: NBR ISO/IEC 27001; NBR ISO/IEC 27002; e ISO/IEC 15408. Esses padrões estabelecem normas e técnicas padronizadas para que os processos possam ser melhor trabalhados, cada qual com sua particularidade. Adiante, será exposta cada norma e suas particularidades.

### **1.2.1 NBR ISO/IEC 27001 e NBR ISO/IEC 27002**

O objetivo de tais normas é “estabelecer diretrizes e princípios gerais para iniciar, implementar, manter e melhorar a gestão da segurança da informação em uma organização” (ABNT NBR ISO/IEC 27017, 2016, [s.p.]). Essas duas normas, se trabalhadas em conjunto, tornam-se o mais completo padrão da segurança da informação, já que tem por base padrões, normas e controles internacionais, o que garante um aperfeiçoamento do processo.


Inicialmente, foi criada pela *British Standard Institute* e denominada como BS 23 7799. Com o passar do tempo, tornou-se ISO/IEC 1779, adotada pela ISO e, mais tarde, em 2001, adotada pelo Brasil com o nome NBR ISO/IEC 17799, por meio da Associação Brasileira de Normas



Técnicas (ABNT). Ao longo do tempo, a norma sofreu atualizações, sendo a primeira em 2005 e, por esse motivo, é conhecida como parte um, recebendo novos capítulos e chamando-se NBR ISO/IEC 27002. A parte dois denomina-se NBR ISO/IEC 27001 e abrange o ciclo PDCA (*Plan-Do-Check-Act*) (FERREIRA; ARAÚJO, 2008).

Estruturalmente, a norma NBR ISO/IEC 27002 está distribuída da seguinte forma (ABNT NBR ISO/IEC 27002, 2013):

1. Escopo.
2. Termos e definições.
3. Estrutura da norma.
4. Avaliação e tratamento dos riscos.
5. Política de segurança da informação: criação de um documento norteador sobre o assunto, onde deve conter o comprometimento com a política adotada, bem como a estrutura para controle, análise e avaliação do gerenciamento de riscos, além de normas e requisitos específicos para a organização.
6. Organizando a segurança da informação: estabelecimento de uma estrutura para implementar a segurança da informação, onde as atividades devem ser coordenadas por representante de diversos setores da organização.
7. Gestão de ativos: como ativo, temos qualquer coisa que tenha valor para a organização e, desse modo, devem ser classificados e protegidos, fazendo com que regras documentais passem a existir.
8. Segurança em recursos humanos: a contratação de novos colaboradores, bem como toda a manutenção do quadro corporativo deve ser executado com extrema atenção, além de normas e práticas a serem seguidas.
9. Segurança física e do ambiente: instalações de processamento de informação devem operar em local seguro, além de possuir um controle de acesso.
10. Gestão das operações e comunicações: definir procedimentos e responsabilidades, pela gestão e operação de todos os recursos



de processamento das informações, é de extrema importância para a organização. Também é recomendado, por segurança, que não apenas uma pessoa execute todo o processo, mas parte dele, dificultando o risco de mau uso ou uso indevido dos sistemas.

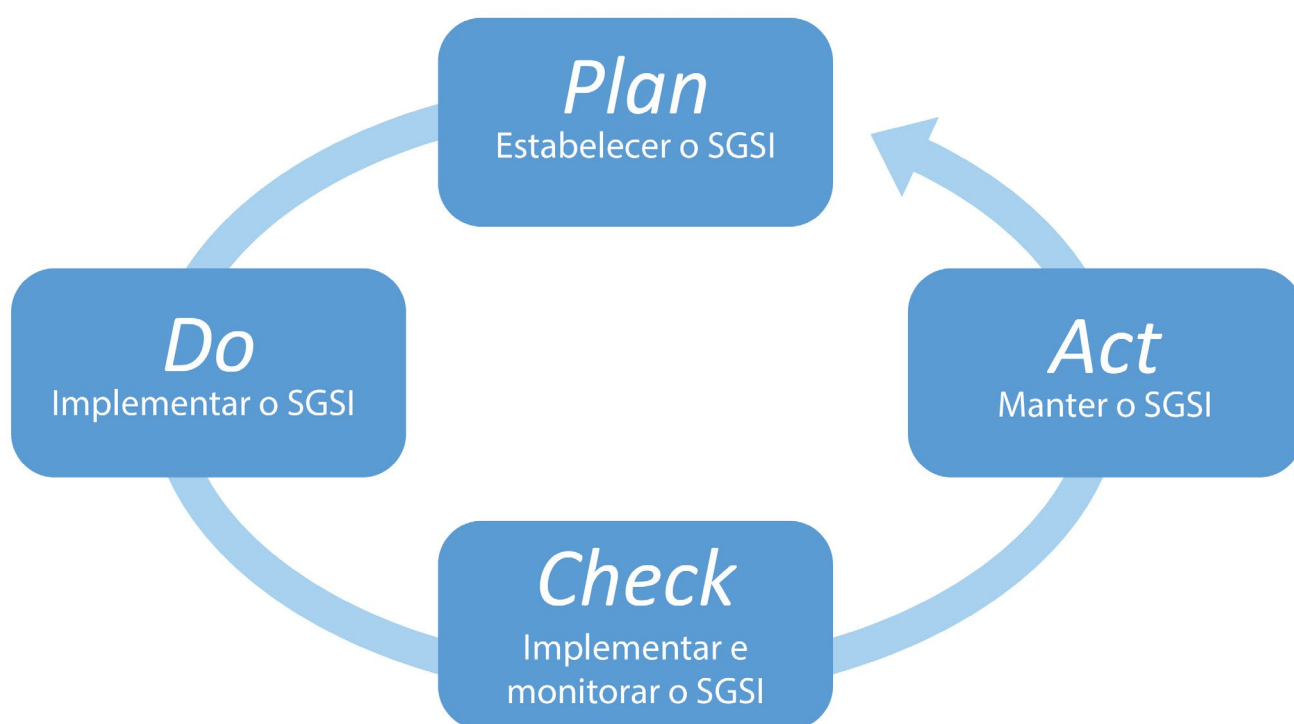
11. Controle de acesso: por meio dos requisitos de negócio e da segurança da informação, o acesso à informação deve ser controlado. Garantir que apenas usuários devidamente cadastrados acessem a informação é essencial para a garantia da segurança dos dados.
12. Aquisição, desenvolvimento e manutenção de sistemas de informação: antes do desenvolvimento propriamente dito, ou de sua implementação, os requisitos de segurança devem ser elencados e estudados, garantindo, assim, maior proteção às informações ali armazenadas.
13. Gestão de incidentes de segurança da informação: incidentes quanto à segurança da informação deve ser comunicada o mais rápido possível, a fim de que a ação corretiva possa ser aplicada o quanto antes. Para que isso ocorra, devem ser estabelecidos procedimentos formais de registro e escalonamento, para que todos os envolvidos conheçam os procedimentos de notificação.
14. Gestão da continuidade do negócio: deve-se impedir a interrupção das atividades do negócio e proteger os processos críticos contra defeitos e/ou falhas, não deixando que isso interfira na atividade a ser desenvolvida, ou que retorne o mais rápido possível.
15. Conformidade: lei criminal, civil, estatutos ou qualquer outra forma de regimento deve ser preservado, garantindo, assim, sua violação.

Seguindo a estrutura elencada acima, o modelo a ser aplicado tende a ser bem estruturado e definido, fazendo com que haja uma organização e gerenciamento dentro da instituição. Dessa maneira, teremos como resultado um *information security management system* (ISMS) ou sistema de gestão de segurança da informação (SGSI). Seu objetivo é definir

ações planejadas que contemplem práticas, diretrizes, procedimentos, modelos e qualquer outro tipo de medida que vise a reduzir os riscos para a segurança da informação.

Um ISMS ou SGSI pode ser construído com o padrão do ciclo PDCA, destinado à segunda parte da NBR ISO/IEC 27001, onde visa, por meio de suas métricas, um planejamento preciso e confiável. Veja na Figura 1 como essa gestão é realizada.

**Figura 1 – Sistema de Gestão de Segurança da Informação (SGSI)**



Fonte: elaborada pelo autor.

De acordo com Ferreira e Araújo (2008), o ciclo PDCA visa nortear os passos a serem seguidos, possuindo cada um sua especificação. Observe o Quadro 1, que relata o que cada fase é responsável em executar.

## Quadro 1 – Ciclo PDCA, implementação de um SGSI

Fase I – <i>Plan</i>	Fase II – <i>Do</i>	Fase III – <i>Check</i>	Fase IV – <i>Act</i>
<ul style="list-style-type: none"> <li>• Estruturação do SGSI.</li> <li>• Plano diretor de segurança.</li> <li>• Diagnóstico de segurança.</li> <li>• Avaliação, tratamento dos riscos e seleção dos controles de segurança.</li> <li>• Declaração de aplicabilidade (<i>statement of applicability</i>).</li> </ul>	<ul style="list-style-type: none"> <li>• Comitê de segurança da informação.</li> <li>• Política de segurança.</li> <li>• Classificação da informação.</li> <li>• Plano de continuidade dos negócios e de TI;</li> <li>• Treinamento e conscientização.</li> <li>• Implementação dos controles especificados na declaração de aplicabilidade.</li> </ul>	<ul style="list-style-type: none"> <li>• Monitoração dos controles de segurança.</li> <li>• Gestão de incidentes.</li> <li>• Revisão do nível de risco residual.</li> <li>• Auditoria interna do SGSI.</li> </ul>	<ul style="list-style-type: none"> <li>• Implementação de melhorias.</li> <li>• Ações corretivas e preventivas.</li> <li>• Comunicação das ações e resultados para alta administração e partes interessadas.</li> <li>• Assegurar que as melhorias foram implementadas e atenderam às expectativas.</li> </ul>

Fonte: Ferreira e Araújo (2008).

Contudo, com a utilização de todos os itens do quadro acima, a garantia do gerenciamento de riscos está assegurada, além dos objetivos de controle estarem implementados, formando, dessa maneira, um SGSI. Algo importante, quando o assunto é gerenciamento de equipes e projetos, visto que o sucesso ou fracasso do produto depende das diversas etapas que antecedem a completude do projeto.

## 1.2.2 ISO/IEC 15408

Por meio do documento norteador existente no mercado, o *common criteria for information technology security evaluation* (critério comum para avaliação de segurança de tecnologia da informação), que a norma ISO/IEC 15408 foi criada, podendo ser conhecida também como *common criteria*.


A partir de conjuntos de critérios fixos, que permitem a garantia da segurança, por meio de especificações do ambiente da aplicação, a norma é estruturada e pode ser aplicada. Para que isso ocorra, é necessário que haja uma avaliação que determine o nível de segurança, que devem ser atendidos pela funcionalidade e pelos respectivos produtos.

Com base nos resultados obtidos, é possível determinar se tais produtos atendem ou não suas necessidades de segurança, podendo ser utilizadas como guia para o desenvolvimento, avaliação ou aquisição de produtos de TI voltados para a segurança.

Na concepção de Albuquerque e Ribeiro (2002), a norma foi implementada com o intuito da melhoria do processo, pois o sistema deve ser desenvolvido seguindo tais regras, posteriormente, a execução dos referidos testes em um laboratório credenciado e, por último, a emissão de um selo, que atesta que os requisitos de segurança foram cumpridos.

Devemos entender que para um software ser considerado seguro, de acordo com essa norma, os requisitos de segurança precisam existir e serem executados. Agregado a isso, o perfil de proteção, que pode ser aplicado a toda uma classe de sistemas, também deve possuir resultados satisfatórios.





A norma ISO/IEC 15408 possui sete níveis para a garantia da segurança, denominamos *evaluation assurance level* (EAL), entretanto, por questões de flexibilidade, apenas os quatro primeiros níveis são considerados, já que os demais são rígidos e acabam tornando-se inviáveis para sua aplicabilidade (ALBUQUERQUE; RIBEIRO, 2002).

O *Common Criteria Portal* (2015) define os sete níveis EAL, sendo:

- a. EAL 1 = Funcionalmente testado: certifica que o TOE teve seu funcionamento testado.
- b. EAL 2–Estruturalmente testado: estabelece que o sistema teve sua estrutura testada. Envolve a cooperação do fabricante.
- c. EAL 3–Metodicamente testado e verificado: certifica que o TOE foi metodicamente testado e checado.
- d. EAL 4–Metodicamente projetado, testado e verificado: define que o sistema foi metodicamente projetado, testado e checado.
- e. EAL 5–Semiformalmente projetado e testado: prevê que o sistema seja implementado e testado de maneira não formal.
- f. EAL 6–Semiformalmente projetado, testado e verificado: certifica que o TOE foi projetado, verificado e testado de maneira não formal.
- g. EAL 7–Formalmente projetado, testado e verificado: define que o sistema foi projetado, verificado e testado de maneira formal.

Contudo, a utilização da norma para a padronização, no quesito segurança, é de suma importância para a maturidade do produto, além de garantir que o ciclo de desenvolvimento cumpra com todos os passos propostos, assegurando maior confiança na aplicação.



## 2. Modelo de segurança

Com o intuito de focar na segurança daquilo que é projetado e desenvolvido, modelos foram criados para auxiliar e nortear, antes, durante e após o desenvolvimento do produto. Práticas como essa, enriquecem o mercado de desenvolvimento de software, uma vez que a cada novo método desenvolvido, a possibilidade de reorganizar processos, que até então não tinham uma padronização, tornam-se possíveis, fazendo com que todos ganhem, ou seja, desde a equipe que trabalha diretamente na concepção do produto até o usuário final.

Garantir que as organizações ou empresas se adequem a modelos que visam e buscam as boas práticas de desenvolvimento de seus produtos, além de focarem na qualidade e segurança da aplicação, faz com que cada vez mais padrões e métodos passem a ser criados e/ou aprimorados. Todos devem ter como fundamento que um sistema deve passar por muitas formas e maneiras de teste logo no início de sua concepção, e não apenas ao término do processo, garantindo, assim, que caso algo ocorra de errado ou que possa ser melhorado, que seja feito antes ou durante a criação, e não após.

### 2.1 Modelo SDL

Com foco na segurança para os produtos desenvolvidos, a Microsoft, em 2002, traça um modelo denominado *Security Development Lifecycle* (SDL), que passa a ser implementado em suas aplicações. Envolve uma série de atividades e produtos com foco no software seguro, que são trabalhados durante a fase de desenvolvimento do software. Para Maristela e Jorge (2011), alguns aspectos importantes desse modelo merecem destaque, como:

- a. Criação de modelos de ameaças durante o design do software.

- b. Utilização de ferramentas de verificação de código de análise estática durante a implementação.
- c. Realização de revisões de código e testes de segurança.

É com base na integração de medidas e adição de pontos de verificação, que o SDL propõe modificar a maneira como o software é comumente desenvolvido, tendo como resultado produtos com alto padrão de segurança.

Nessa perspectiva, podemos determinar que o modelo compõe uma série de atividades, que merecem atenção especial, segundo De Win *et al.* (2009): o aumento da qualidade, com foco na segurança da funcionalidade do produto; os processos que são bem definidos em estágios, distribuídos ao longo da fase de criação do produto; a facilidade nas orientações a serem seguidas, já que o método é pragmático e simples de ser compreendido; e, por fim, a descrição das atividades, que torna a gestão do desenvolvimento menos complexa de ser entendida, facilitando o gerenciamento da segurança como uma qualidade do produto.

A Quadro 2 demonstra a sequência de fases a serem seguidas durante a execução do modelo SDL.

## Quadro 2 – Processo de desenvolvimento SDL da Microsoft

Treinamento	Requisitos	Design	Implementação	Verificação	Liberação	Resposta
Segurança principal Treinamento	Estabelecer requisitos de segurança  Criar protões de qualidade/ barras de erros	Estabelecer requisitos de design  Analisar superfícies de ataque  Modelagem de ameaças	Utilizar ferramentas aprovadas  Desaprovar funções não seguras  Análise estática	Análise dinâmica  Teste de fuzzing  Revisão da superfície de ataque	Plano de resposta de incidentes  Revisão final de segurança  Liberar arquivo	Executar plano de resposta de incidentes

Fonte: elaborado pelo autor.

Para que haja melhor entendimento sobre o processo, veremos o que cada fase compreende e o que é destinado a cada uma:

1. Treinamento de segurança: com o intuito de conceituar e formar toda a equipe de desenvolvimento sobre aspectos como princípios, tendências em segurança e privacidade, são aplicados treinamentos de nível básico e avançado sobre o tema. De acordo com Microsoft (2010), esses princípios compreendem: desenho seguro para redução da superfície de ataque; defesa em profundidade; privilégio mínimo; *defaults* seguros; modelagem de ameaças; codificação segura; teste de segurança e questões relacionadas a privacidade de informações pessoais.
2. Requisitos: são analisados com maior atenção os requisitos de segurança e privacidade da aplicação, bem como a realização de uma análise de custo para averiguar se os recursos necessários estão de acordo com projetado. Também é de responsabilidade dessa fase o rastreamento dos bugs, que deve elencar suas causas ou origem do defeito.
3. Design: realizado análise da superfície de ataque da aplicação e posterior modelo de ameaças dela.
4. Implementação (codificação segura): por meio da utilização de linguagem de programação, a produção de código executável

passa a ser possível. Nesse sentido, a produção de codificação segura se faz necessária com a adoção de padrões de codificação segura, que evitam que ataques sejam realizados no software. Outra maneira de proteção é quanto ao uso de ferramentas de análise estática de código, que têm por objetivo identificar a codificação propensa à introdução de vulnerabilidades.

5. Verificação: esta fase é destinada a testes, inspeções de código e análise da documentação do software. Recomenda-se a utilização de ferramentas automatizadas, já que tendem a serem mais práticas e precisas, evitando eventuais falhas humanas.
6. Release (liberação de versões): nesta fase é produzido um plano de ação, que tem por objetivo nortear a equipe de tratamento de incidentes sobre eventuais falhas de segurança no software.
7. Resposta: tem por objetivo nortear quanto ao tratamento de incidentes relacionados à aplicação.

Por fim, você pode perceber que o assunto segurança, voltado ao desenvolvimento de software, trata de ações em diversas etapas durante o ciclo de desenvolvimento de software, onde técnicas, modelos e métricas favorecem as boas práticas de construção e criação de aplicações. Também é importante manter toda a equipe alinhada com o modelo a ser executado, visto que são eles quem farão o processo ocorrer, e isso está diretamente ligado ao sucesso ou fracasso do produto a ser concebido.

## Referências Bibliográficas

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **ABNT NBR ISO/ IEC 27002 – Tecnologia da informação – Técnicas de segurança – Código de prática para a gestão de segurança da informação**. ABNT, 2016.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS (ABNT). **ABNT NBR ISO/IEC 27002 – Tecnologia da informação — Técnicas de segurança — Código de prática para controles de segurança da informação**. ABNT, 2013. Disponível em: <http://www.professordiovani.com.br/AdmRedes/NBRISO-IEC27002.pdf>. Acesso em: 29 set. 2020.

ALBUQUERQUE, R.; RIBEIRO, B. **Segurança no desenvolvimento de software**: como garantir a segurança do sistema para seu cliente usando a ISO/IEC. Rio de Janeiro: Editora Campus, 2002.

AZEVEDO, D. **Metodologias de segurança de sistemas**. São Paulo: IFSP, 2008.

COMMON CRITERIA PORTAL. **Common Criteria**, Publications. Disponível em: <http://www.commoncriteriaportal.org/cc>. Acesso em: 29 set. 2020.

DE WIN, B.; SCANDARIATO, R.; BUYENS, K. *et al.* Wouter-On the secure software development process: CLASP, SDL and Touchpoints compared. **Journal of Information and Software Technology**, v. 51, p. 1152-1171, 2009.

FERREIRA, F. N. F.; ARAÚJO, M. T. de. **Política de Segurança da Informação – Guia prático para elaboração e implementação**. 2. ed., rev. Rio de Janeiro: Editora Ciência Moderna Ltda, 2008.

HOLANDA, M. T de; FERNANDES, J. H.; CABRAL, J. H. **Segurança no desenvolvimento de aplicações**. Gestão da Segurança da Informação e Comunicações, UNB, 2011. Disponível em: [https://cic.unb.br/~jhcf/MyBooks/cegsic/2009\\_2011/GSIC701\\_Seguranca\\_Desenvolvimento\\_Aplicacoes.pdf](https://cic.unb.br/~jhcf/MyBooks/cegsic/2009_2011/GSIC701_Seguranca_Desenvolvimento_Aplicacoes.pdf). Acesso em: 5 out. 2020.

MCGRAW, G. **Bridging the gap between software development and information security**. IEEE Security and Privacy, 2005.

OLIVEIRA, W. J. de. **Segurança da informação**: técnicas e soluções. Florianópolis: Visual Books, 2001.

PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. 7. ed., Dados eletrônicos. Porto Alegre: AMGH, 2011.

# Ferramentas CASE para testes de segurança do software

Autoria: Luís Otávio Toledo Perin

Leitura crítica: Rafael Maltempe da Vanso



## Objetivos

- Definir o que é segurança em testes de software.
- Compreender os principais testes de segurança em software.
- Apresentar ferramenta CASE para testes de segurança em software.



# 1. A segurança e o produto software

## 1.1 Introdução

Neste tema, veremos como as ferramentas CASE auxiliam os testes de segurança do software, fazendo com que o processo fique mais confiável e ágil durante seu ciclo de desenvolvimento.

Vamos refletir sobre algumas perguntas, pois esse processo é essencial para que você possa compreender a dimensão de um assunto tão importante em nossa era digital. Já parou para pensar como o uso da tecnologia foi se transformando com o passar dos anos? Quanta tecnologia foi produzida nessa última década, por exemplo? Quantas foram as invenções? E os serviços criados? Todos esses questionamentos servem de parâmetro para analisarmos que estamos vivendo em uma era tecnológica, digital e conectada, e que, com sua evolução, o mundo transformou-se, assim como seus usuários de gerações diferentes.

Assim como o software sofreu evolução com o passar dos anos, toda sua cadeia de desenvolvimento, que abrange a concepção, o desenvolvimento propriamente dito e a manutenção, também passou por avanços significativos, fazendo com que o produto se tornasse mais sólido e competitivo no mercado. Novos processos e práticas foram criados para atender a essa nova tecnologia, mais especificamente no quesito segurança, já que com processos na nuvem e acesso a todo e em qualquer lugar, algumas metodologias tiveram que se adaptar para ofertar toda a integridade necessária ao usuário.

Tudo isso está relacionado, direta ou indiretamente, com o assunto que abordaremos. Vejamos, então, sobre as ferramentas CASE para testes de segurança do software.



## 1.2 O produto software


Você já deve ter ouvido, inúmeras vezes, a palavra software, inclusive aqui nesta leitura. Entretanto, entender o que é um software, como é projetado, desenvolvido e mantido ao longo do tempo, é de grande valia para compreender essa dimensão, principalmente na área da segurança, já que processos tendem a sofrer mudanças constantes.

Para Pressman (2011), a definição de software é dividida em três, sendo:

Software consiste em: (1) instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenho desejados; (2) estruturas de dados que possibilitam aos programas manipular informações adequadamente; e (3) informação descritiva, tanto na forma impressa como na virtual, descrevendo a operação e o uso dos programas. (PRESSMAN, 2011, p. 32)

Como visto acima, a definição de software vai além de um programa computacional desenvolvido para uma determinada atividade, ou seja, diversos outros fatores estão agregados àquela prática, como a inserção de dados para organização do negócio ou a exibição e impressão de relatórios gerenciais para a tomada de decisão, por exemplo. Tudo isso é parte do produto software, que passa por diversas etapas até sua finalização.

Com o avanço da tecnologia, novas aplicações passaram a ser desenvolvidas, bem como a preocupação da engenharia de software, mais especificadamente do engenheiro, em organizar e manipular todos os possíveis tipos de software que o mercado comporta. Dessa forma, Pressman (2011, p. 34) elenca sete grandes categorias de software, possuindo cada uma sua particularidade e especificação técnica, o que deve ser levado em consideração durante seu desenvolvimento, teste e manutenção da aplicação: de sistema; de aplicação; científico/ de engenharia; embutido; para linha de produtos; para a web; e de inteligência artificial.



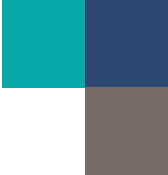
Dentro do princípio da segurança da informação, manter a informação segura é parte inerente das atribuições que um software deve possuir, garantindo que não haja vulnerabilidade, mas proteção contra invasores e acesso não autorizado, sendo essa eventual vulnerabilidade oriunda da forma como são armazenados ou transmitidos os dados, por exemplo, segundo Goertzel *et al.* (2006).

Com o avanço da tecnologia e sua alta disseminação, sistemas e aplicações tornam-se mais comuns em nosso dia a dia. Garantir a proteção a sistemas que armazenam, transmitem e processam a informação, tem se tornado um grande desafio para os engenheiros e desenvolvedores de software, já que inúmeras são as ameaças, bem como a vasta gama de recursos computacionais, sendo: a Internet como serviço principal para comunicação e troca de informações e dados a nível mundial; a Intranet como um serviço local de rede única ou múltipla; a computação móvel como grande disseminadora da tecnologia na palma das mãos; a computação ubíqua, que é o fato da tecnologia estar onipresença no cotidiano das pessoas.

Conforme Goertzel *et al.* (2006) relata, um sistema de informação tem o objetivo de auxiliar a tomada de decisão, sendo baseado em computador, mas com pessoas, software, recursos e hardware, o que pode afetar o aspecto segurança, já que leva em consideração os aspectos relevantes do sistema e do ambiente ao qual está inserido.

Como o fator humano se torna evidente neste processo, atos inseguros podem ocorrer de maneira intencional ou não, devendo a segurança do software preferencialmente evitar, ou minimizar possíveis ataques ou tentativas deles. Para que isso seja possível, controles e funções devem ser criadas dentro do ciclo de desenvolvimento do software, que buscam garantir a integridade da aplicação e dos dados ali contidos.

Dentre os fatores que colaboram com a exploração de sistemas, podemos citar, segundo Goertzel *et al.* (2006):

- 
1. A exposição da aplicação é umas das principais janelas de invasão, já que o número de ameaças tende a crescer e explorar a vulnerabilidade de tais softwares.
  2. A complexidade e tamanho do software, além das diversas interfaces que pode possuir. Tais fatores podem dificultar a realização e validação de testes corretos e precisos, o que pode abrir uma brecha de segurança.
  3. A terceirização durante o desenvolvimento do software, ou seja, a utilização de desenvolvedores de fora da empresa e do ambiente corporativo, além do uso de software não verificado para sua construção.
  4. A mescla de software legado para outras aplicações e em novos ambientes. Tal fator pode acarretar consequências não tão positivas, além do aumento do número de vulnerabilidades.

O software necessita ser seguro devido a sua função, ou seja, é uma ferramenta de apoio para o usuário e, desse modo, não pode permitir que invasões sejam bem-sucedidas, além de se preocupar com as práticas desenvolvidas por seu executor, já que não pode permitir adulteração de dados ou coleta de informações não autorizadas. Com isso, utilizar práticas de desenvolvimento seguro aumentam a chance da aplicação ser segura e diminuem possíveis invasões durante sua utilização.

Contudo, vimos que existem diferentes categorias de software e cada qual tem suas particularidades, fazendo com que seja necessário possuir um conhecimento mais direcionado sobre o que se pretende trabalhar, além da equipe estar alinhada com práticas e métricas de desenvolvimento que busquem a segurança da aplicação, e serem condizentes com o mercado.

## 1.2.1 A segurança

Assim como em qualquer outro produto, a segurança é essencial para que haja aceitação por parte do consumidor, nesse caso, do usuário ou do cliente, visto que a segurança dos dados manipulados é de extrema importância para a organização. Desse modo, garantir que o sistema possa ser acessado em todo e qualquer lugar (on-line), e com segurança, é um requisito não funcional que deve ser cumprido à risca.

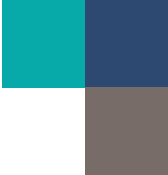
De acordo com McGraw (2005), a preocupação com a segurança deve existir desde a concepção do projeto:

[...]Devemos nos preocupar com a segurança, a confiabilidade, a disponibilidade e a dependência — nas fases inicial, de projeto, de arquitetura, de testes e de codificação, ao longo de todo o ciclo de vida [qualidade] de um software [...] (MCGRAM, 2005, p. 369).

Essa preocupação deve estar alienada à manutenção do software, ou seja, deve ser constantemente verificada e analisada quanto a anomalias ou qualquer outro tipo de ameaça, a fim de assegurar não só a segurança, mas todos os outros fatores agregados, como confiabilidade e disponibilidade, por exemplo.

O software considerado seguro, ou seja, resistente à invasão, é aquele que reduz a probabilidade de um ataque ser bem-sucedido, e, caso ocorra, que os danos possam ser suavizados, sendo a aplicação menos afetada possível. Entretanto, isso só será possível se os conceitos de segurança de software forem aplicados de maneira eficiente durante o ciclo de desenvolvimento do mesmo, segundo Paul (2011).

O software seguro deve ser concebido desde o primeiro planejamento efetuado para sua construção, garantindo, assim, que seja desenhado, construído, configurado e suportado para que, mesmo sob ataque, opere corretamente e resista a explorações do invasor ou ainda que limite os danos gerados pela invasão, segundo Goertzel *et al.* (2007).



Ainda para Goertzel *et al.* (2007), as propriedades a seguir caracterizam a segurança do software:

- a. Falhas exploráveis e outras fraquezas são evitadas por desenvolvedores bem-intencionados.
- b. A probabilidade de que desenvolvedores mal-intencionados implantem falhas exploráveis ou lógicas maliciosas no software é baixa ou nula.
- c. O software será resistente, tolerante ou resiliente a ataques.

Compreenda que para o software ser seguro, é essencial incorporar esse conceito desde a primeira fase do projeto, ou seja, no levantamento de requisitos, no projeto, no desenvolvimento, na fase de testes e na implementação, assim como durante a manutenção que deverá sofrer em toda sua vida útil, já que mudanças e ajustes sempre são necessários para a boa usabilidade do produto.

Nesse sentido, preocupar-se com a aplicação para que seja segura, deve ser um dos principais requisitos elencados pela equipe do projeto, e algumas propriedades de dependabilidade devem ser levadas em consideração, como as seis propriedades fundamentais elencadas por Knight (2002):

- a. Confiança: manterá o funcionamento correto quando estiver em uso.
- b. Disponibilidade: estará operacional quando necessário.
- c. *Safety*: sua operação não trará perigo ao usuário ou operador.
- d. Confidencialidade: não haverá revelação não autorizada da informação.
- e. Integridade: não haverá modificação não autorizada da informação.
- f. Manutenibilidade: facilidade de modificação do software, com objetivo de corrigir defeitos, se adequar a novos requisitos ou se ajustar a um ambiente novo.

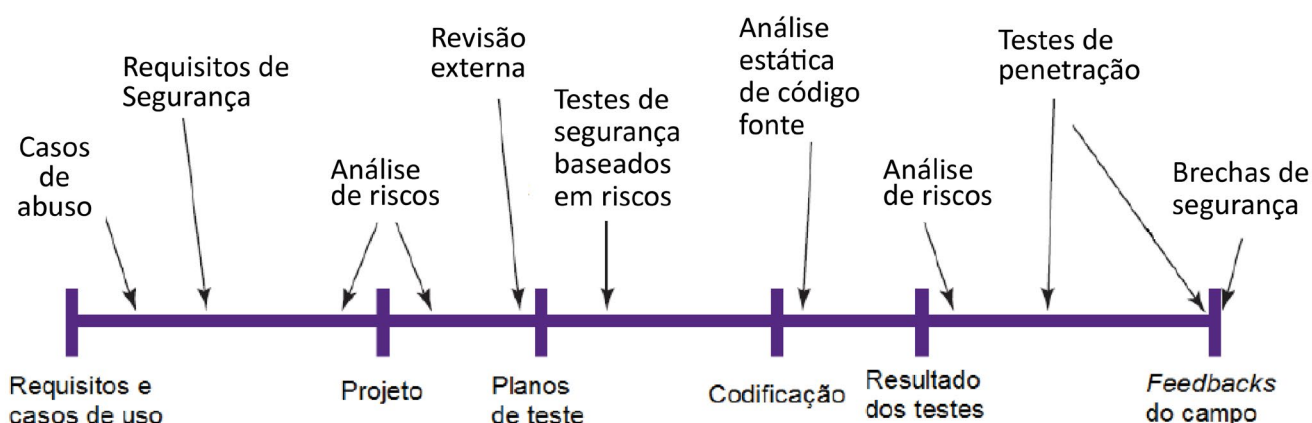
Contudo, com a utilização de tais propriedades e a adoção de boas práticas durante o desenvolvimento do software, a chance de problemas com segurança ocorrerem, com a aplicação, diminuem de maneira significativa, já que todos os cuidados foram tomados para que o produto possuísse a melhor qualidade possível.

## ➤ 2. Ciclo de vida do software e suas atividades

Sabemos que o ciclo de desenvolvimento do software é composto por várias etapas, que devem ser seguidas por toda a equipe, a fim de construir um produto com a máxima qualidade possível. E se durante esse ciclo de vida algumas atividades forem realizadas? Com a adoção de boas práticas voltadas para a segurança durante todas as fases do ciclo de vida da aplicação, torna-se mais preciso e seguro o desenvolvimento do produto software.

A Figura 1 exemplifica como tais atividades ocorreriam durante o ciclo de vida do software, segundo McGraw (2006):

**Figura 1–Atividades de segurança no ciclo de vida do software**



Fonte: McGraw (2006 p. 82).

Para McGraw (2006), cada atividade desenvolvida ao longo do ciclo de vida do software é importante para a segurança conforme segue:

- a. Casos de abuso: atividade voltada para avaliar se algum padrão de ataque se encaixa no sistema ou em seus requisitos, bem como para realizar uma relação entre os problemas e a análise de risco. Também é a fase para se projetar cenários de vulnerabilidade que podem ser explorados nas fases seguintes, como revisão de código ou teste de penetração.
- b. Requisitos de segurança: fase focada no levantamento das necessidades de segurança do software, além do mapeamento para garantir sua correta implementação. É importante que os requisitos de segurança atendam aos requisitos funcionais e de segurança, a fim de coibir possíveis padrões de ataque.
- c. Análise de risco arquitetural: fase relacionada com o levantamento dos riscos de segurança que podem estar presentes na arquitetura da aplicação que será construída. É dedicada também ao processo de gerenciamento de risco para garantir a aplicabilidade de normas de segurança.
- d. Teste de segurança baseado em riscos: os testes desenvolvidos precisam absorver dois itens principais, sendo os testes dos requisitos de segurança com técnicas de teste para requisitos funcionais e testes de segurança baseados em riscos, que foram anteriormente levantados pelos casos de abuso e pela validação dos padrões de ataque.
- e. Revisão de código fonte: após o desenvolvimento das linhas de código, e antes da fase de testes, é importante revisar a codificação desenvolvida para averiguar se os requisitos de segurança foram bem implementados, além das vulnerabilidades listadas na análise de casos de abuso. Fica a critério quanto ao formato da revisão, ou seja, automatizado ou manual, entretanto, o processo automatizado pode não contemplar todos os cenários, criando lacunas na verificação.
- f. Testes de penetração: fase direcionada para testar, de maneira dinâmica, falhas de projeto ou vulnerabilidades, que a aplicação possa ter. Importante para assegurar que nenhum problema



em potencial, que possa ser explorado durante a execução do software, passe despercebido.

- g. Operação segura: fase destinada a assegurar que as atividades desenvolvidas pelo usuário possam ser rastreadas, além de protegê-las em um eventual ataque, fazendo com que estejam em segurança.

Por fim, manter a integridade do software, assim como focar na segurança durante seu ciclo de desenvolvimento, é de fundamental importância para uma construção íntegra e concisa, além de não depender diretamente do tipo de ciclo de vida adotado, sendo uma prática universal a qualquer metodologia escolhida.

### 3. Teste de segurança

O teste de segurança é importante para detectar possíveis vulnerabilidades presentes no software. Assim como afirma Myers (1979 p. 23), “testar é analisar um programa com a intenção de descobrir erros e defeitos”. Nesse aspecto, é importante garantir que toda e qualquer vulnerabilidade seja identificada, documentada e corrigida, mas sempre devemos saber que um software perfeito nunca existirá, ou seja, com a ausência de defeito. Por isso, é imprescindível que normas, técnicas e diretrizes sejam utilizadas para que o sistema permaneça dentro dos padrões desejados, fazendo com que a análise de riscos possa ser efetuada.

Testar a segurança de um software é de extrema importância para prever o comportamento da aplicação quanto a possíveis ataques de invasores, além de averiguar qual será o comportamento frente ao inimigo, já que o sistema deve estar preparado para suportar a invasão, bem como assegurar que os dados estejam protegidos.



## 3.1 Técnicas de teste de segurança em redes e aplicações

A fim de auxiliar no processo de verificação de testes de segurança, o *National Institute of Standards and Technology* (NIST), agência não regulatória do departamento de comércio dos Estados Unidos, tem disponibilizado documentos a respeito do teste de segurança. O documento *Special Publication 800-115: Technical Guide to Information Security Testing and Assessment [NIST 2008]* elenca as etapas básicas a serem seguidas, sendo dividido em três grandes grupos: técnicas de revisão; técnicas de identificação e análise; técnicas de validação e vulnerabilidades. Também define técnicas de teste de segurança de rede, organizado em oito seções:

- Seção 1: introdução, que exemplifica o propósito e o escopo do documento.
- Seção 2: exhibe a visão geral das avaliações de segurança da informação, bem como políticas, papéis e responsabilidades, metodologias e técnicas.
- Seção 3: fornece descrição detalhada das várias técnicas de avaliação, incluindo análise de documentação, análise de logs, interceptação de dados na rede e verificação de integridade de arquivos.
- Seção 4: relata técnicas para a identificação de alvos e procedimentos para mapear possíveis vulnerabilidades.
- Seção 5: explica as técnicas usadas para validar a existência de vulnerabilidades.
- Seção 6: apresenta abordagem e processo de planejamento de uma avaliação de segurança.

- Seção 7: discute os fatores fundamentais para a execução de avaliações de segurança, incluindo coordenação, a própria avaliação, análise e manipulação de dados.
- Seção 8: descreve sobre a apresentação dos resultados da avaliação e fornece uma visão geral das ações para mitigar os riscos.

Tais técnicas são de extrema importância para a construção de um produto software com a mais alta qualidade e segurança, coisa que o mercado já vem solicitando nesta área. Também é importante para garantir a integridade dos dados ali inseridos e gravados no banco de dados, visto que toda a aplicação realiza a manipulação de inúmeras informações ao longo do tempo, e devem possuir a integridade possível, fazendo com que seu acesso seja por vias legais.

A técnica *Penetration Testing* da *Special Publication 800-115 do NIST* deve ser precedida por uma análise dos riscos envolvidos, para que os componentes que necessitam de maior atenção na aplicação possam ser identificados e priorizados. A técnica também orienta que deva existir no mínimo três fases de avaliação de segurança, sendo: planejamento, execução e avaliação. A primeira tem por objetivo analisar como será a execução do processo de auditoria; a segunda para identificar ou validar possíveis vulnerabilidades; a terceira, o processo de avaliação para identificar a origem de tais vulnerabilidades, além de tentar solucioná-las.

É de extrema importância que a execução de testes de segurança ocorra desde as primeiras etapas do projeto, evitando, assim, que vulnerabilidades possam ser deixadas sem seu devido tratamento. Outro fator que deve ser levado em consideração é o custo, já que, para tratá-lo, o valor sofre aumento significativo, visto que está à nível de projeto, envolvendo a arquitetura do sistema.



## 4. Metodologia *Secure Tropos*


Apesar do ciclo de desenvolvimento de software contar com boas práticas e métricas que seguem um direcionamento quanto a construção de um sistema, o assunto segurança não é frequentemente tratado com a devida importância que deveria ter. Isso pode estar relacionado com o tempo de desenvolvimento, além das diversas fases empregadas durante a criação, que acabam se sobressaindo frente à segurança.

Nesse sentido, a metodologia *Secure Tropos*, que é uma extensão de *Tropos*, que por sua vez é um *framework* de análise com preocupação na segurança ao longo das fases de desenvolvimento do sistema, é apresentada com a finalidade de modelar problemas de segurança durante todo o ciclo de desenvolvimento do produto software, segundo Mouratidis (2009). Consiste em um conjunto de conceitos de requisitos da engenharia de domínio e engenharia de segurança, formando, assim, parâmetros de restrição de segurança, objetivos de segurança e ataques.

Esse processo visa a análise das necessidades de segurança, identificando seus objetivos e, conseqüentemente, implementando mecanismos e recursos de segurança para auxiliar e agregar nas métricas do software. Seu conceito é baseado em restrições de segurança, ou seja, durante a fase de análise e desenvolvimento, alguns requisitos podem sofrer alteração, deixarem de existir ou serem implementados, já que o foco é manter a segurança da aplicação.

A metodologia *Secure Tropos* possui quatro fases básicas, sendo:

1. Em um primeiro momento, durante a análise de requisitos, as restrições de segurança são expostas como declarações de alto nível para serem analisadas posteriormente.

- 
2. Após a fase de análise de requisitos, as restrições são impostas ao sistema a ser desenvolvido, sendo analisado o objetivo de segurança e entidade necessária para o sistema garantir os requisitos de segurança.
  3. Durante a fase de designer, pode surgir novos atores que serão analisados. O designer do sistema é definido por meio dos requisitos de segurança do sistema.
  4. Por fim, durante a fase de projetos, os itens identificados nas fases anteriores são concebidos com a ajuda do *Agente Unified Modeling language* (Auml).

Além das fases, o *Secure Tropos* possui várias etapas durante seu processo de desenvolvimento, sendo:

1. Modelagem de atores.
2. Modelagem de dependências.
3. Identificação de restrições de segurança.
4. Delegação de restrições de segurança e objetivos.
5. Modelagem de objetivos.
6. Modelagem de restrições de segurança.
7. Modelagem de entidades de segurança.
8. Análise de ameaças.
9. Análise de vulnerabilidades.
10. Modelagem de atacantes.
11. Seleção de provedor de nuvem.

Além de contar com todas essas fases, a metodologia também apresenta cinco visões:

1. Visão organizacional.
2. Visão dos requisitos de segurança.
3. Visão de componentes de segurança.
4. Visão de ataques à segurança.
5. Visão de análise de nuvem.

## 4.1. Ferramentas CASE

Uma das ferramentas CASE, que suporta a metodologia *Secure Tropos*, é a SecTro2, estando em sua versão 2.0 e sendo disponível apenas para Windows. Como requisito, é preciso possuir um banco de dados para armazenamento dos projetos.

Também existem alternativas, como pode ser visto no Quadro 1, que existem para dar suporte a metodologia, e algumas ainda fazem extensões a mesma, acrescentando outros componentes.

**Quadro 1 – Ferramentas CASE de suporte a metodologia *Tropos***

Ferramenta	Mantenedor	Descrição
STS-Tool.	DISI-University of Trento.	Ferramenta de apoio para a linguagem de modelagem de segurança sócio-técnica.
Si* Tool.	DISI-University of Trento.	Suporta raciocínio e planejamento de risco.
TAOM4E.	Fondazione Bruno Kessler.	Geração de código para JADEX.
GR-Tool.	DISI-University of Trento.	Ferramenta para raciocínio de objetivos.
T-Tool.	DISI-University of Trento.	Ferramenta para <i>Tropos Formal</i> .
DW-Tool.	DISI-University of Trento.	Projeto de <i>Data Warehouse</i> .
OpenOME.	University of Toronto.	Modelagem dirigida a requisitos, modelando os <i>stakeholders</i> .

DESCARTES.	Université catholique de Louvain.	Projeto de sistemas, ambientes, técnicas e repositórios orientado a agentes.
SecTro.	University of East London.	Ferramenta de modelagem automatizada, que fornece suporte para a metodologia <i>Secure Tropos</i> .

Fonte: adaptado de [www.troposproject.eu/](http://www.troposproject.eu/). Acesso em: 1 out. 2020.

## Referências Bibliográficas

COMPUTER WORLD. Willoughby, M. Q&A: Quality Software Means More Secure Software,” Computerworld, March 21, 2005. Disponível em: <https://www.computerworld.com/article/2563708/q-a—quality-software-means-more-secure-software.html>. Acesso em: 5 out. 2020.

GOERTZEL, K. M. *et al.* **Software Security Assurance**: a State of the Art Report (SOAR). Information Assurance Technology Analysis Center (IATAC), 2007.

GOERTZEL, K. M.; WINOGRAD, T.; MCKINLEY, H. L. *et al.* **Security in the Software Life Cycle**. v. 1.2 (Draft), 2006.

MCGRAW, G. **Software security**: building security in. Boston: Addison Wesley Professional, 2006.

MOURATIDIS, H. Secure Tropos: an agent oriented software engineering methodology for the development of health and social care information systems, **International Journal of Computer Science and Security**, n. 3, p. 241-271, 2004. Disponível em: [https://www.researchgate.net/publication/41822939\\_Secure\\_Tropos\\_An\\_Agent\\_Oriented\\_Software\\_Engineering\\_Methodology\\_for\\_the\\_Development\\_of\\_Health\\_and\\_Social\\_Care\\_Information\\_Systems](https://www.researchgate.net/publication/41822939_Secure_Tropos_An_Agent_Oriented_Software_Engineering_Methodology_for_the_Development_of_Health_and_Social_Care_Information_Systems). Acesso em: 5 out. 2020.

MYERS, G. **The art of software testing**. Wiley, 1979.

PAUL, M. **Official (ISC)2-Guide to the CSSLP**. 1. ed. Florida: CRC Press, 2011.

PRESSMAN, R. S. **Engenharia de software**: uma abordagem profissional. 7. ed. Dados eletrônicos. Porto Alegre : AMGH, 2011.



Tropos. Disponível em: <<http://www.troposproject.eu/>>. Acesso em 22 jul. 2020.

VIENNA. **SecTro2–User manual for “SecTro2” v 2.0**. Disponível em: <http://vienna.omilab.org/repo/files/SecTro/User%20Manual.pdf>. Acesso em: 29 set. 2020.



**Bons estudos!**