

**Pratique,  
aprenda,  
conquiste.**

 **Proz**  
Viva sua profissão!

Disciplina | Qualidade de Software

**Téc. Sistemas de Desenvolvimento**



**Aqui  
começa  
a sua  
jornada**

**Vamos nessa?**



Disciplina | Qualidade de Software

**Téc. Sistemas de Desenvolvimento**

## SUMÁRIO

QUALIDADE DE SOFTWARE.....	4
<b>TEMA 01.....</b>	<b>5</b>
Introdução a Teste de Software.....	5
<b>TEMA 02.....</b>	<b>12</b>
Conceito de Qualidade de Software.....	12
<b>TEMA 03.....</b>	<b>24</b>
Ciclos de Vida de Software.....	24
<b>TEMA 04.....</b>	<b>34</b>
Rational Unified Process (RUP) e TMap (Testing Management Approach).....	34
<b>TEMA 05.....</b>	<b>47</b>
Processos de revisões.....	47
<b>TEMA 06.....</b>	<b>60</b>
Ambientes de Testes.....	60
<b>TEMA 07.....</b>	<b>77</b>
Virtualização.....	77
<b>TEMA 08.....</b>	<b>86</b>
Teste Funcional.....	86
<b>TEMA 09.....</b>	<b>97</b>
Teste Estrutural.....	97
<b>TEMA 10.....</b>	<b>103</b>
Diagnóstico.....	103

# QUALIDADE DE SOFTWARE



## INTRODUÇÃO

A qualidade de software desempenha um papel crucial na satisfação dos usuários e no sucesso de projetos de desenvolvimento de software. Nesta apostila, exploraremos os principais conceitos e práticas relacionados à qualidade de software e testes. Compreenderemos a importância do teste de software como um componente essencial no ciclo de vida do desenvolvimento de sistemas, além de abordar o conceito abrangente de qualidade de software.

O teste de software é uma parte vital da garantia de qualidade em projetos de software. Ele desempenha um papel fundamental na identificação e correção de defeitos e falhas, garantindo que o software entregue aos usuários finais atenda aos requisitos e funcione de maneira confiável. Entenderemos como o teste de software é uma ferramenta poderosa para garantir que o software seja funcional, seguro e eficiente.

Além disso, exploraremos o que constitui qualidade em software. A qualidade não se limita apenas à ausência de erros, mas também envolve eficiência, segurança e usabilidade. Aprenderemos como avaliar e medir a qualidade de software para alcançar resultados mais eficazes nos projetos.

No contexto dos ciclos de vida de software, analisaremos os diferentes modelos, como Waterfall, Agile e outros. Destacaremos o Rational Unified Process (RUP) e o Testing Management Approach (TMAp) como metodologias que integram testes em todas as fases do desenvolvimento de software. Compreenderemos como a escolha do ciclo de vida pode influenciar a qualidade do produto final.

À medida que avançamos nesta apostila, abordaremos tópicos adicionais, como processos de revisões, ambientes de testes e virtualização. A revisão sistemática de código e documentos contribui para a melhoria da qualidade. A criação de ambientes de testes controlados e a importância da virtualização para economia de recursos e eficiência dos testes também serão discutidas.

Nos capítulos subsequentes, aprofundaremos tópicos específicos, como testes funcionais, testes estruturais, diagnóstico de problemas e muito mais. Ao concluir esta apostila, você estará preparado para compreender, aplicar e valorizar os princípios fundamentais da qualidade de software e testes em projetos de desenvolvimento de software.

## TEMA 01

### Introdução a Teste de Software

#### Habilidades

- Compreender e aplicar as técnicas de qualidade de software;



Disponível em: <[freepik-https://tinyurl.com/3pdwenn5](https://tinyurl.com/3pdwenn5)>. Acesso em 27 ago. 2023.

No mundo atual, onde a tecnologia desempenha um papel fundamental em diversas áreas da sociedade, a qualidade de software é um aspecto crucial para o sucesso de qualquer organização. Um software de qualidade é capaz de atender às necessidades dos usuários, garantir a confiabilidade e a eficiência das operações, e manter a satisfação do cliente em alta. Neste primeiro capítulo, exploraremos os princípios, conceitos, modelos, práticas e metodologias relacionadas à qualidade de software, fornecendo uma base sólida para compreender e aplicar técnicas que promovam a qualidade em todas as fases do ciclo de vida do software.

Nas décadas de 1960 e 1970, os desenvolvedores dedicavam a maior parte dos seus esforços nas atividades de especificação e codificação e nos testes unitários. Estima-se que 80% desse esforço era despendido nessas atividades.

Uma parcela menor era dedicada aos testes de integração dos programas, mas não havia testes de sistema. As atividades de teste eram consideradas um mal necessário para provar aos usuários que os produtos funcionavam e não eram tratadas como um processo formal alinhado com as atividades do processo de desenvolvimento de sistemas (também poucas vezes tratado como processo).

Nesta época os dados de teste eram criados pelo analista de sistema e os resultados eram entregues ao usuário para aprovação. Muitas vezes a massa de teste era proveniente de uma parte dos arquivos usados em produção.

Nos últimos anos, com a utilização da Internet para a realização de negócios, houve uma mudança significativa na abrangência e complexidade das aplicações, onde fatores, tais como segurança e performance passaram a ser relevantes, tornando a atividade de testar cada vez mais especializada.

Antes de mergulharmos mais a fundo no conteúdo recomendo que escaneie o QR Code ao lado para que possa assistir um vídeo de um overview sobre Qualidade de Software e sua atuação no mundo da tecnologia.



Fonte do vídeo: [https://youtu.be/FLX9\\_QqzDAs](https://youtu.be/FLX9_QqzDAs)

## Definição e Importância do Teste de Software:



Disponível em: <<http://guts-rs.blogspot.com/2016/06/evento-de-junho-praticas-de.html>>. Acesso em 27 ago. 2023.

### Existem diversas definições para teste de software:

- Avaliar se o software está fazendo o que deveria fazer, de acordo com os seus requisitos, e não está fazendo o que não deveria fazer;
- Processo de executar um programa ou sistema com a intenção de encontrar defeitos (teste negativo) (Glen Myers-1979);
- Qualquer atividade que a partir da avaliação de um atributo ou capacidade de um programa ou sistema seja possível determinar se ele alcança os resultados desejados (Bill Hetzel-1988).

Muitas outras definições poderiam ser ainda citadas, mas em essência, teste de software é o processo que visa a execução do software de forma controlada, com o objetivo de avaliar o seu

comportamento, baseado no que foi especificado. A execução dos testes é considerada um tipo de validação (testes de aceitação) ou verificação (testes unitários, de integração e de sistema).

Na prática, não se pode testar um programa completamente e garantir que ele ficará livre de "bugs". É quase impossível testar todas as possibilidades de formas e alternativas de entrada de dados, bem como testar as diversas possibilidades e condições criadas pela lógica do programador, assim como as suas diversas combinações.

Segundo uma estimativa de Beizer (1990), a média do número de defeitos em programas liberados para testes é de 1 a 3 por 100 instruções executáveis. Claro, existem diferenças entre programadores, porém uma coisa é certa, todos eles cometem erros em grau maior ou menor.

Se não se pode descobrir todos os defeitos de um programa, em consequência nunca se pode afirmar que ele está 100% correto, por que testar? Porque o propósito dos testes é descobrir e corrigir os problemas e com isto melhorar a sua qualidade. O quanto se quer melhorar dependerá de quanto se deseja investir. O tempo (prazo) dedicado à execução dos testes muitas vezes tem uma relação direta com a quantidade de defeitos que ocorrem na produção. Isto é, se você dedica menos tempo do que o necessário, com toda certeza mais defeitos irão ocorrer na produção.

A qualidade do software depende também do investimento feito no processo de testes. Um software mal testado poderá custar caro (e muito) para a organização. Dentro do processo de teste existem também outras técnicas de verificação, tais como: inspeção, revisão de produtos e "walkthroughs" – passo a passo.

Estas técnicas baseadas em reuniões e "checklists" servem para identificar defeitos de elaboração, descumprimento de padrões e das boas práticas. Devem ser realizados em documentos produzidos, planos, códigos, especificações, requisitos entre outros e preferencialmente antes da execução dos testes. Estas técnicas, usadas de forma combinada com os testes, aumentam sensivelmente a qualidade final dos softwares desenvolvidos.

Em resumo, a validação se refere exclusivamente ao teste de aceitação e verificação aos testes unitário, de integração e de sistemas. Realmente as inspeções técnicas e revisões estão no processo de verificação também sob o enfoque do modelo **CMMI**.

## Processo de Teste de Software

Deve basear-se em uma metodologia aderente ao processo de desenvolvimento, em pessoal técnico qualificado, em ambiente e ferramentas adequadas. A metodologia de teste deve ser o documento básico para organizar a atividade de testar aplicações no contexto da empresa. Como é indesejável o desenvolvimento de sistemas sem uma metodologia adequada, também acontece o mesmo com as atividades de teste. O desenho da Figura abaixo mostra as fases de um ciclo de vida num processo de testes:



Disponível em: <<https://tinyurl.com/mr2x9szs>>. Acesso em 27 ago. 2023.

Conforme Boehm (1976), quanto mais tarde um defeito for identificado mais caro fica para corrigi-lo e mais ainda, os custos de descobrir e corrigir defeitos no software aumentam exponencialmente na proporção que o trabalho evolui através das fases do projeto de desenvolvimento.

Um outro aspecto que devemos considerar é o papel dos testes na manutenção dos sistemas. Uma grande parcela do orçamento de TI das organizações é dedicada à manutenção dos softwares após eles entrarem em produção. A maioria dos testes feitos durante a manutenção são os mesmos que foram feitos durante o desenvolvimento. Neste momento devem ser aplicados os Testes de Regressão, preferencialmente se automatizados, todas as vezes que os programas forem alterados.

### Fica evidente que:

- Quanto melhores forem os testes feitos durante o desenvolvimento, menores serão os custos de manutenção;
- As manutenções solicitadas pelos usuários são fontes de novos defeitos, inclusive gerando problemas em partes do programa que não foram modificados e para identificar estas situações, sempre devem ser aplicados os testes de regressão completos, evitando testar apenas as modificações realizadas;
- Certos testes, tais como o de carga em ambiente Web, só podem ser realizados com auxílio de ferramentas de automação de testes, pois possuem a capacidade de simular o ambiente real, muito difícil de ser realizado por pessoas;
- Quanto mais especializada e independente a equipe de testes, melhor será a qualidade do sistema e menor o custo total.

Abaixo veremos um exemplo de um Ciclo de Vida de Desenvolvimento de Software:

### Procedimentos iniciais

Elaboração do documento Guia Operacional de Testes (GOT), ou seja, o estabelecimento de um acordo entre as partes envolvidas no projeto de teste (usuário, desenvolvimento, teste e produção) para a definição dos seguintes assuntos: objetivo do projeto de teste, pessoal a ser envolvido (desenvolvimento, equipe de testes e usuários), as responsabilidades de cada um, o plano

preliminar de trabalho, a avaliação dos riscos, os níveis de serviço acordados e qualquer item considerado relevante pelo responsável das atividades de teste para garantir o sucesso do projeto. Pode ser considerado o “*kick off*” do projeto de teste ou o seu estudo de viabilidade.

**Planejamento** - Elaboração e revisão da Estratégia de Testes e do Plano de Teste.

**Preparação** - Preparação do ambiente de teste, incluindo equipamentos, rede, pessoal, software e ferramentas.

**Especificação** - Elaboração e revisão dos Casos de Teste, "scripts" ( no caso de uso de ferramentas de automação de testes) e dos Roteiros de Teste e execução dos testes de verificação da documentação do sistema (testes estáticos).

**Execução** - Execução dos testes planejados conforme os Casos de Teste, "scripts" (no caso de uso de ferramentas de automação de testes) e dos Roteiros de Teste com os correspondentes registros dos resultados obtidos.

**Entrega** - Conclusão do processo de testes com a entrega do sistema para o ambiente de produção.

Outro aspecto importante que deve ser considerado é a forma como os processos de desenvolvimento de sistemas e de testes interagem. De qualquer forma, tenha em mente que o ciclo de vida mostrado é apenas um exemplo e que existem outros ciclos de vida para o processo de teste. Ou seja, poderemos encontrar denominações diferentes para as mesmas atividades ou fases.

## MPS.BR (Melhoria de Processos do Software Brasileiro)

Assim como temos o CMMI um modelo referência no mundo para práticas necessárias à maturidade, em vários casos temos semelhantemente em nosso País o MPS.BR que é um modelo de melhoria de processos de software que foi criado para melhorar os processos de software no Brasil. Ele surgiu em 2003, levando em conta como as empresas brasileiras realmente funcionam, e sua missão é apresentar um modelo de processo que ajude a elevar o nível do desenvolvimento de software no país. Em resumo, o MPS.BR veio para impulsionar a qualidade e o aprimoramento dos processos de software feitos aqui no Brasil.

O principal foco do MPS.BR é melhorar a capacidade de desenvolvimento de software, serviços e as práticas de gestão de RH na indústria da tecnologia de informação e comunicação. E para isso foram elaborados três modelos para melhoria nessas áreas respectiva.

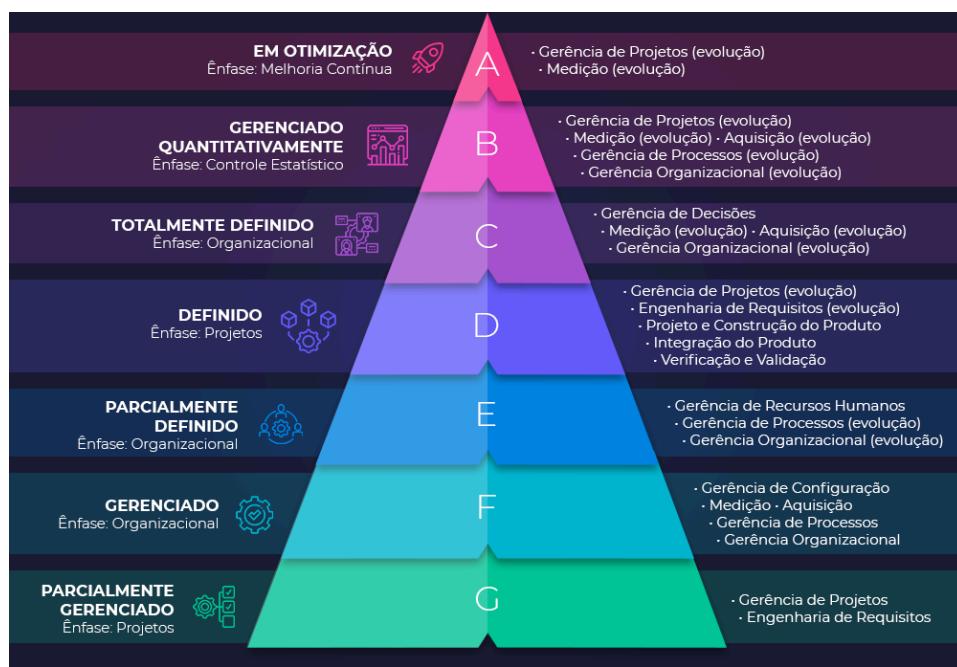


Disponível em: <<https://www.youtube.com/watch?v=SUTecO8YQIs>>. Acesso em 27 ago. 2023.

**MPS SW** - Esse modelo concentra-se na melhoria dos processos específicos relacionados ao desenvolvimento de software. Ele fornece diretrizes e práticas para otimizar a forma como o software é concebido, desenvolvido, testado e mantido. O MPS SW é voltado para empresas que desejam aprimorar seus processos de software, aumentando a eficiência, a qualidade e a produtividade em todas as etapas do ciclo de vida do software.

**MPS SV** - Esse modelo é voltado para a melhoria dos processos relacionados à prestação de serviços, abrangendo áreas como suporte técnico, atendimento ao cliente, consultoria, entre outros. O MPS SV busca fornecer diretrizes e práticas para otimizar a entrega de serviços, melhorando a eficiência, a qualidade e a satisfação do cliente.

**MPS RH** - Esse modelo tem como foco a melhoria dos processos relacionados à gestão de pessoas. Ele busca fornecer diretrizes e práticas para otimizar atividades como recrutamento, seleção, treinamento, avaliação de desempenho, desenvolvimento de carreira e gestão de competências. O MPS RH visa melhorar a eficácia da gestão de recursos humanos, promovendo um ambiente de trabalho mais produtivo e satisfatório.



Disponível em: <<https://tinyurl.com/365638ru>>. Acesso em 27 ago. 2023.

Assim como o Modelo do CMMI, o Modelo do MPS.BR também apresenta níveis de maturidade, começando da Letra G e escalando até a Letra A, assim como podemos ver na imagem acima.



## RESUMO

O teste de software é um elemento fundamental no desenvolvimento de produtos de software de alta qualidade. Envolve uma série de técnicas e abordagens voltadas para a verificação e validação das funcionalidades de um sistema, a fim de identificar defeitos e garantir que o software atenda aos requisitos especificados. A introdução ao teste de software abrange uma variedade de conceitos essenciais para a compreensão desse processo.

No âmbito das técnicas de qualidade de software, o teste desempenha um papel crucial. Ele engloba diferentes abordagens, como testes de unidade, testes de integração, testes de sistema e testes de aceitação. Cada uma dessas fases concentra-se em aspectos específicos do software, garantindo que ele funcione corretamente individualmente e em conjunto. Isso ajuda a identificar erros em estágios iniciais, economizando tempo e recursos no longo prazo.

Além das abordagens técnicas, compreender os princípios por trás do teste de software é fundamental. Isso envolve entender a diferença entre defeitos e falhas, reconhecendo que defeitos são os problemas subjacentes no código, enquanto as falhas são suas manifestações externas. Também é importante aprender a elaborar casos de teste eficazes, que abrangem cenários diversos e representativos do uso real do software.

A introdução ao teste de software não se trata apenas de identificar problemas, mas também de documentar e relatar os resultados de maneira organizada. Isso permite uma comunicação clara entre os desenvolvedores, testadores e demais partes interessadas. Além disso, compreender a importância do ciclo de vida do teste, integrando-o desde as fases iniciais de desenvolvimento até a entrega final, é um aspecto crucial para garantir a qualidade do produto final.

Em resumo, a introdução ao teste de software abrange a compreensão das técnicas e princípios que norteiam a busca por qualidade no desenvolvimento de software. A capacidade de aplicar diferentes abordagens de teste, reconhecer a importância da documentação e compreender como o teste se encaixa no ciclo de vida do desenvolvimento são habilidades essenciais para profissionais que desejam produzir software confiável e de alto desempenho.



## ATIVIDADE DE FIXAÇÃO

1. Qual é a importância da qualidade de software para o sucesso de uma organização?
2. Cite duas áreas em que a qualidade de software é importante?
3. Quantos níveis de Maturidade o modelo MPS.BR tem?

4. Por que não é possível testar completamente um programa e garantir a ausência de defeitos?
  5. Defina qualidade de software em suas próprias palavras.
  6. Quais são as principais atividades do processo de teste de software?
  7. Explique a diferença entre validação e verificação no contexto de testes de software.
  8. Discuta a relação entre investimento no processo de testes e qualidade do software.
  9. Explique o que é O MPS.Br e suas vertentes dentro do modelo.
- 10.** Divida a Turma em Grupos e peça para os grupos pesquisarem sobre CMMI e MPS.BR e apresentarem quais são as diferenças, benefícios e malefícios entre os modelos.

## TEMA 02

### Conceito de Qualidade de Software

#### Habilidades:

- Compreender e aplicar as técnicas de qualidade de software;
- Entender o que são testes funcionais, não funcionais e estruturais em sistemas;
- Executar testes funcionais e estruturais em sistemas;

#### O que é Qualidade de Software?



Disponível em: <rawpixel-<https://tinyurl.com/tsbkexbc>>. Acesso em 27 ago. 2023.

O termo qualidade é utilizado em diversas situações. E, nota-se, sempre, que o que chamamos de qualidade, em qualquer situação, depende de alguns fatores, que, se modificados, podem alterar a nossa percepção da qualidade. Para mim um restaurante de qualidade, por exemplo, tem de possuir bom atendimento, refeições saborosas e bem feitas, tem de possuir um ambiente organizado e higiênico. Agora, dependendo da pessoa, a noção de qualidade de um restaurante pode ser bem diferente da minha. Tendo-se em vista essa variabilidade, a ASQ (American Society for Quality – Sociedade Americana para a Qualidade), apresenta o seguinte conceito:

*"Qualidade – Um termo subjetivo, para o qual cada pessoa, ou setor, tem a sua própria definição. Em sua utilização técnica, a qualidade pode ter dois significados: As características de um produto ou serviço, que dão suporte (ou sustentação), à sua habilidade em satisfazer requisitos especificados ou necessidades implícitas; Um produto ou serviço livre de deficiências."*

Conforme vimos a definição de Qualidade pode variar. Qualidade pode ser vista como a

adequação ao uso. Pode-se aplicá-la, seja a situações do cotidiano, quanto a situações mais específicas, como, por exemplo, num processo de fabricação.

Antes de mergulharmos mais a fundo no conteúdo recomendo que escaneie o QR Code ao lado para que possa assistir um vídeo de introdução sobre Qualidade de Software onde serão apresentados alguns exemplos do dia a dia entre outros detalhes formadores básicos sobre o assunto.



Fonte do vídeo: Gerência e Qualidade de Software - Aula 01 - Visão Geral

Contudo, considerando-se, o Sistema de Gestão da Qualidade, amplamente aplicável, sob os requisitos da ISO 9000, a definição normativa é: **qualidade é o grau no qual um conjunto de características inerentes, satisfaz a requisitos** (ABNT NBR ISO 9000:2005), sendo que características, segundo a mesma norma são propriedades diferenciadoras. As características são de diferentes tipos, como físicas, sensoriais, comportamentais, temporais, ergonômicas ou funcionais.

Definição do Guia PMBOK para Garantia e Controle da Qualidade:

**Garantia da Qualidade (QA):** é o processo de auditoria dos requisitos de qualidade e dos resultados das medições de controle de qualidade, a fim de assegurar a conformidade com os padrões de qualidade e definições operacionais apropriados. Este processo de execução utiliza dados gerados durante a execução do Controle de Qualidade (QC).

**Controle da Qualidade (QC):** é o processo de monitoramento e registro dos resultados da execução das atividades de qualidade para avaliar o desempenho e recomendar as mudanças necessárias.

## História da Qualidade



Disponível em: <[dilokastudio-https://encurtador.com.br/mDV04](https://dilokastudio-https://encurtador.com.br/mDV04)>. Acesso em 27 ago. 2023.

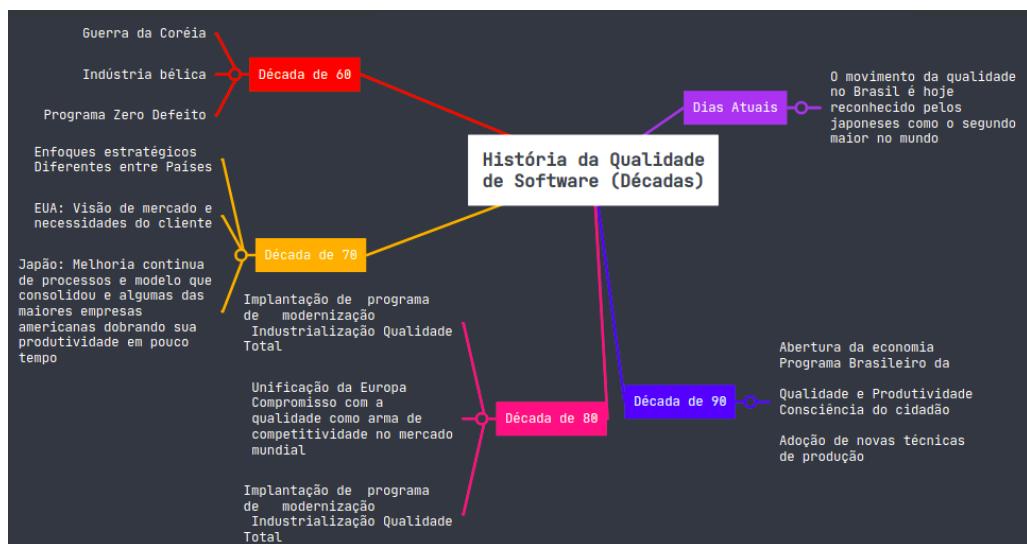
Nos séculos XVIII e XIX, a qualidade era controlada pelos artesãos, que acompanhavam desde a concepção do produto até a sua venda, incluindo as atividades de detecção e correção de erros. Nesta época, a qualidade era associada ao conhecimento individual de cada artesão, facilitada por sua relação com o cliente e com a produção.

Durante a Primeira Guerra Mundial (1914-1918) a situação se agravou. Inúmeros defeitos em produtos militares bélicos foram detectados. Em 1931, a publicação "Economic Control of Manufactured Products", do matemático americano W. A. Shewhart revolucionou os conceitos até então praticados como qualidade.

Pela primeira vez, a qualidade foi abordada com um caráter científico, utilizando-se os princípios da probabilidade e da estatística para inspecionar a produção. No início da Segunda Guerra Mundial (1939-1945), houve uma grande conversão das indústrias para a fabricação de produtos militares com qualidade e dentro dos prazos. Esta época foi considerada o apogeu do controle estatístico da qualidade.

Durante a Segunda Guerra que os japoneses perceberam que seus produtos estavam sucateados e, para não se tornar uma nação extinta, seria preciso partir para a industrialização, importando recursos naturais e exportando produtos manufaturados. Seria necessário qualidade, preço e fabricação eficiente, seria necessário importar conhecimento. Por isso, em 1950, chegou ao Japão o professor W. Edwards Deming levando um método de controle estatístico do processo com o qual treinou milhares de engenheiros e técnicos. Em 1954, o engenheiro Joseph M. Juran também foi ensinar qualidade aos japoneses. Nesta década, foi criada a JUSE (Japanese Union of Scientists and Engineers) para acompanhar e desenvolver as normas de qualidade dentro do Japão.

A figura a seguir representa o Mapa Mental que ilustra a evolução histórica da Qualidade de Software desde a década de 1960 até os dias atuais.



Fonte: Elaborado pelo autor (2023) - Ferramenta Mindmeister.

Assim como vimos no Mapa Mental podemos salientar alguns detalhes sobre cada década em um rápido resumo:

•**Década de 1960** : houve a Guerra da Coréia e a indústria bélica americana se destacou com o programa “zero defeito” criado por Philip Crosby. Nesse período, foram desenvolvidos com sucesso no Japão, por Kaoru Ishikawa, os Círculos de Controle de Qualidade

•**Meados de 1970 a 1980:** os Estados Unidos e o Japão representavam as maiores potências no processo da qualidade, porém defendiam enfoques estratégicos diferentes. Os EUA investiram na visão de mercado e nas necessidades do cliente. O Japão crescia investindo na melhoria contínua de seus processos. O modelo japonês se consolidou e algumas das maiores empresas americanas, que adotaram o método, em pouco tempo dobraram sua produtividade.

•**Meados de 1986 a 1999:** Em 1986, a Europa assinou seu tratado de unificação e os doze países comprometidos com o acordo postaram na qualidade como arma de competitividade no mercado mundial. Estabeleceu-se uma nova ordem para a qualidade no ocidente. Desde então, a qualidade no Japão tornou-se palavra de ordem, e nos últimos anos, o que se vê em todo o mundo, é uma grande disputa pelos produtos japoneses, mesmo considerando sua posição geográfica, a valorização do iene e vários outros motivos. Diante do atual panorama mundial, as organizações têm buscado implantar o Controle de Qualidade Total, as certificações (garantias) e almejados prêmios de qualidade a fim de se garantir no mercado globalizado.

No Brasil a preocupação com a qualidade se deu inicialmente em 1980. Porém, somente com a abertura da economia ao mercado internacional, em 1990, as empresas sentiram a necessidade de garantir sua sobrevivência, sendo de extrema necessidade o aumento da produtividade e da qualidade. A criação do Programa Brasileiro da Qualidade e Produtividade (PBQP), pelo governo, e a consciência do cidadão enquanto consumidor foram decisivos para o desenvolvimento da qualidade no Brasil.

A década de 80 foi um divisor de águas na industrialização brasileira e que, em 1988, foi implantado um radical programa de modernização que, entre outras consequências, motivou, principalmente, no setor privado a adoção de novas técnicas de produção, entre elas, a Qualidade Total.

•**Dias Atuais:** O movimento da qualidade no Brasil é hoje reconhecido pelos japoneses como o segundo maior no mundo. A característica mais importante do caso brasileiro é a adesão de empresas de todos os setores, indústrias de transformação e de construção, comércio, serviços, setor agrícola e, inclusive, o serviço público.

## Qualidade do Processo

A qualidade do processo de desenvolvimento afeta diretamente a qualidade dos produtos fornecidos. Essa suposição é derivada dos sistemas de produção, em que a qualidade do produto está intimamente relacionada ao processo de produção. Na verdade, em sistemas automatizados de produção em massa, uma vez atingido um nível aceitável de qualidade de processo, a qualidade do produto decorre naturalmente.

Existe uma ligação entre a qualidade de processos e o produto em produção, pois o processo é relativamente fácil de padronizar e monitorar. Uma vez ajustados os sistemas de produção eles

podem ser executados novamente diversas vezes a fim de produzir produtos de alta qualidade.

Então se o processo de desenvolvimento de software for bem definido a chance dos produtos que forem produzidos em cima dele terem melhor qualidade é alta. No entanto, se o processo não for bom, sabe-se que o insucesso é certo.

## Certificação da Qualidade



Disponível em: <[fotografia.com.br/kqrK4](https://fotografia.com.br/kqrK4)>. Acesso em 27 ago. 2023.

É o reconhecimento por uma entidade acreditadora da conformidade do sistema da qualidade de uma organização com o referencial normativo que adotar para essa certificação. Essa norma fornece às empresas, um conjunto de regras básicas, que deverão ser adaptadas à sua realidade específica, de forma a garantir ao cliente que a empresa se encontra devidamente organizada e é capaz de produzir um bem ou serviço que esteja de acordo com os requisitos do cliente e normativos aplicáveis.

Basicamente, é uma norma vocacionada para a estruturação da organização da empresa e para a resposta ao cliente. Quem procurar nesta norma indicações precisas dos pormenores da organização desilude-se. A norma é muito generalista, pois foi pensada para ser aplicada numa multidão de empresas diferentes. Dá-nos muito simplesmente princípios orientadores. Por exemplo, devemos manter registros dos contatos e acordos mantidos com os clientes, mas a forma, o meio, o tipo de registros e a informação que deverão conter deve ser da responsabilidade da empresa pois esta é que sabe o que necessita para o desenrolar do seu trabalho.

Como peça de filosofia normativa esta norma é excelente: os princípios orientadores que fornece aplicam-se de uma forma pungente à realidade empresarial. Todas as empresas produzem produtos, que são sujeitos a diversos controles de qualidade em várias fases, que tem todos os seus processos de desenvolvimento, que são sujeitos a requisitos e especificações legais e do cliente, etc..

A aplicação prática destas normas é de uma importância primordial para o sucesso empresarial e para a evolução das empresas e dos produtos e serviços que fornecem.

Primeiramente, pode ser vista como um instrumento para as empresas gerenciarem e garantirem o nível de qualidade de seus produtos e, como segundo objetivo, informar e garantir aos consumidores que os produtos certificados possuem os atributos procurados, atributos, esses, intrínsecos aos produtos. Atributos intrínsecos devem ser entendidos como atributos que não podem ser visualizados e percebidos externamente.

A grande importância da certificação da qualidade para qualquer empresa se dá pelo diferencial de qualidade, que abre as portas do mundo globalizado para as empresas certificadas, uma vez que, ao adquirirem produtos ou serviços dessas empresas o consumidor tem certeza de que existe um sistema confiável de controle das etapas de desenvolvimento, elaboração, execução, e entrega do produto provido de um tratamento formalizado com objetivo de garantir os resultados.

### **Benefícios:**

- Propiciar a concorrência justa;
- Estimular a melhoria contínua da qualidade;
- Informar e proteger o consumidor;
- Facilitar o comércio exterior, possibilitando o incremento das exportações;
- Proteger o mercado interno;
- Agregar valor às marcas.

### **O Sistema de Certificação**

O sistema de certificação varia conforme o que será avaliado e qual o interesse de certificação, aqui vou citar alguns passos que podem ocorrer no processo de certificação:

<b>1.</b> Diagnóstico do Sistema de Qualidade existente	<b>2.</b> Planejamento do processo de certificação
<b>3.</b> Informação do pessoal sobre a certificação;	<b>4.</b> Acompanhamento do processo;
<b>5.</b> Redação do manual da qualidade;	<b>6.</b> Redação dos procedimentos;
<b>7.</b> Formação do pessoal;	<b>8.</b> Aquisições necessárias;
<b>9.</b> Formação dos auditores internos da qualidade;	<b>10.</b> Realização das auditorias internas;
<b>11.</b> Implantação de ações corretivas;	<b>12.</b> Implementação prática do sistema de qualidade;
<b>13.</b> Seleção da entidade certificadora;	<b>14.</b> Realização da auditoria de certificação;
<b>15.</b> Preparação para a auditoria de certificação;	<b>16.</b> Planejamento da manutenção do certificado

O prazo para a implementação é definido de acordo com a necessidade, interesse e/ou

tempo disponível do cliente. Após a certificação feita, são feitas auditorias para verificar se de fato as empresas seguem o que foi certificado. Em quase todas as certificações possuem um prazo de validade.

Abaixo vocês poderão conhecer algumas das normas/modelos voltadas à qualidade de software:

Normas/modelos	Descrição/Características
<b>ISO 9126</b>	É uma norma ISO para qualidade de produto de software, que se enquadra no modelo de qualidade das normas da família 9000.
<b>ISO 12119</b>	Esta Norma foi publicada em 1994 e trata da Avaliação de Pacotes de Software, também conhecidos como "Softwares de Prateleira".
<b>ISO 12207</b>	É a norma que define processo de desenvolvimento de software.
<b>ISO/IEC 14598 ABNT-NBR ISO 9001</b>	Esta norma fornece métodos para medida, coleta e avaliação da qualidade de produtos de software.
<b>NBR ISO 9000-3</b>	"Fornece orientações para a aplicação da ISO 9001 ao projeto, desenvolvimento, fornecimento, instalação e manutenção de software".
<b>NBR ISO 10011 CMM-CMMI</b>	Diretrizes para auditoria de sistemas da qualidade. Capability Maturity Model Integration é um modelo de referência que contém práticas (Genéricas ou Específicas) necessárias à maturidade em disciplinas específicas (Systems Engineering (SE), Software Engineering (SW), Integrated Product and Process Development (IPPD), Supplier Sourcing (SS)). Desenvolvido pelo SEI (Software Engineering Institute) da Universidade Carnegie Mellon, o CMMI é uma evolução do CMM e procura estabelecer um modelo único para o processo de melhoria corporativo, integrando diferentes modelos e disciplinas.
<b>MPT.BR / MPS.BR</b>	É um modelo para Melhoria do Processo de Teste concebido para apoiar as organizações de software através dos elementos essenciais para o desenvolvimento da disciplina de teste inserido no processo de desenvolvimento de software.
<b>ISO/IEC 15504 - SPICE</b>	Também conhecida como SPICE, é a norma ISO/IEC que define o processo de desenvolvimento de software. Ela é uma evolução da ISO/IEC 12207 mas possui níveis de capacidade para cada processo assim como o CMMI.
<b>ISO/IEC 25000</b>	Organizações com interesse na área de Engenharia de Software e que utilizam as normas das séries ISO/IEC 9126 e ISO/IEC 14598 (ou na versão ABNT NBR ISO/IEC 9126 e 14598) como referência para especificação e avaliação da qualidade de produto de software devem estar recebendo informações sobre o mercado SQuare. Este modelo, que é um acrônimo de Software Quality Requirements and Evaluation, foi desenvolvido pelo grupo de trabalho

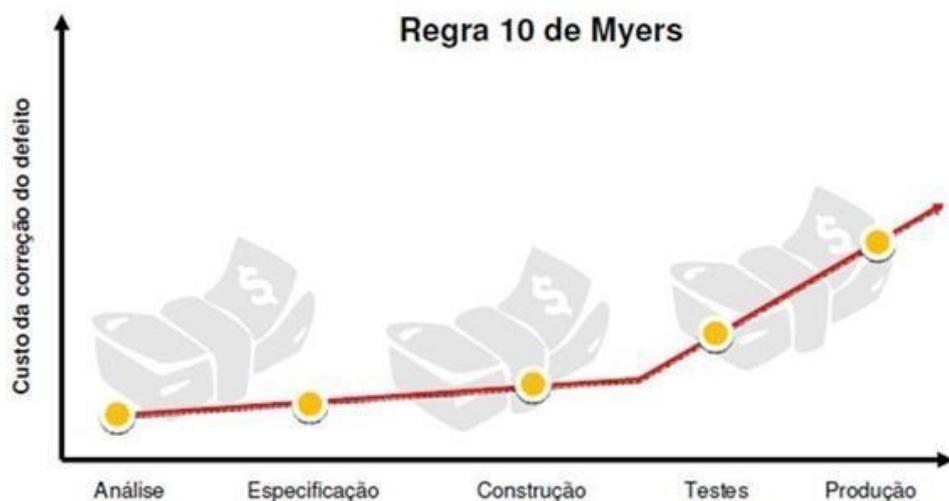
	WG6 do Subcomitê de Sistemas e Software (SC7) da ISO/IEC. O grupo de trabalho WG6 é responsável pela elaboração de normas internacionais que tratam da especificação, medição e avaliação da qualidade de produtos de software. Este modelo, que é um acrônimo de Software Quality A definição da arquitetura de normas SQuaRE teve início em 1999 e vem orientando a revisão das normas já publicadas pela ISO, bem como a criação de novas normas que atendem aos requisitos do mercado e a evolução da Engenharia de Software.
--	---

## Custos da Qualidade

Engloba todos os custos ocorridos no ciclo de vida de um produto, sejam estes custos de conformidade ou de não conformidade. E deve sempre ser visto como um investimento.

Em 1979, Myers, que é um grande convedor da área de teste, escreveu o livro "A arte do teste de software". Nesse livro, ele introduziu a Regra 10 de Myers, um dos conceitos mais importantes já definidos:

Quanto mais cedo descobrimos e corrigimos um defeito, menor é o seu custo para o projeto. Defeitos encontrados nas fases iniciais da etapa de desenvolvimento do software são mais baratos de serem corrigidos do que aqueles encontrados na etapa de produção.



Disponível em: <<https://encurtador.com.br/dCP78>> Acesso em 27 ago. 2023

### Custos da Conformidade:

- Custos de prevenção
- ✓ Treinamento, planejamento, revisões, homologações.
- Custos de inspeções
- ✓ Inspeção, testes laboratoriais, controle da qualidade.

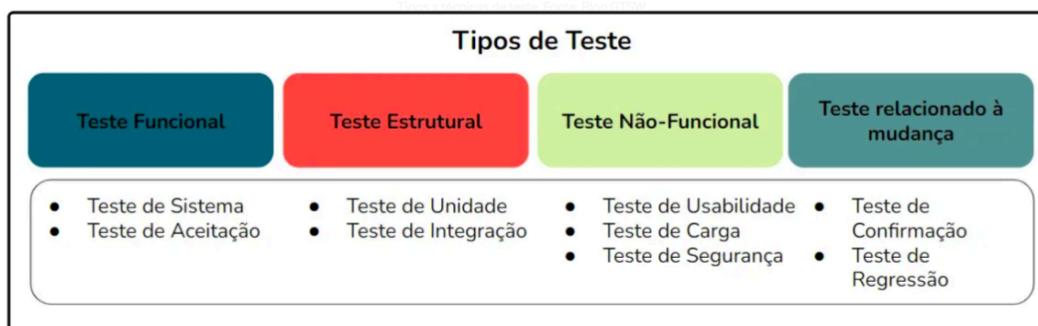
### Custos da Não-Conformidade:

- Custos de falhas internas

- ✓ Retrabalho, refugo, análise de falhas, downtime (tempo de parada), ações corretivas no produto ou serviço, atrasos nos cronogramas
- Custos de Falhas Externas:
- ✓ Reparo durante as garantias, análise reclamações do cliente, visita a clientes, devoluções.

## Testes em Qualidade

São essenciais para assegurar a qualidade de um software. Existem várias abordagens e técnicas para testar um sistema, que podem variar de acordo com sua natureza e objetivo, iremos abordar de forma enxuta 4 categorias que estão representadas na Figura abaixo.



Fonte: Elaborado pelo autor (2023).

**Teste Funcional:** Basicamente, podemos entender que os Testes Funcionais são testes cujo principal objetivo e propósito é verificar se o software cumpre os requisitos específicos de funcionamento, ou seja, se ele executa as funções esperadas.

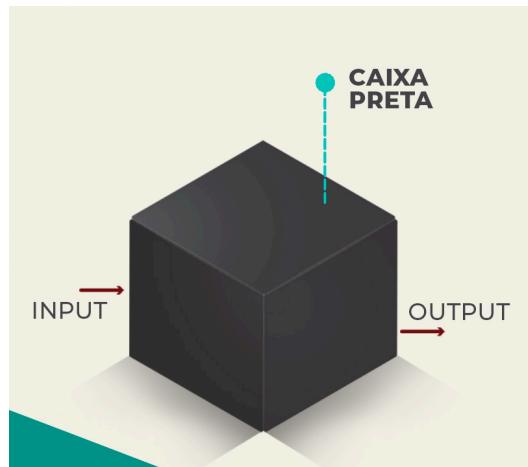
**Teste Estrutural:** Está relacionado à verificação da estrutura interna do software. Ele envolve a análise do código-fonte e a execução de testes com base no conhecimento da implementação interna do sistema. O objetivo é garantir que todas as partes do código sejam testadas e que não haja falhas lógicas ou erros na lógica do programa.

**Teste Não Funcional:** O teste se concentra em avaliar os atributos não funcionais do software, como desempenho, segurança, usabilidade e escalabilidade. Ele verifica se o sistema atende aos requisitos de desempenho, capacidade de resposta, confiabilidade e outras características não relacionadas às funcionalidades específicas. O teste não funcional é crucial para garantir uma experiência de usuário satisfatória e o bom desempenho do sistema em diferentes cenários..

**Teste Relacionado a Mudança:** Este teste é realizado quando ocorrem alterações significativas no software, como correções de defeitos, atualizações de versões ou introdução de novas funcionalidades. Esse tipo de teste visa verificar se as mudanças implementadas não introduzem novos problemas ou afetam negativamente o funcionamento do sistema. Ele envolve a execução de testes regressivos para garantir que as partes existentes do software não tenham sido afetadas pelas alterações realizadas.

Vale mencionar aqui para vocês duas técnicas que são muito utilizadas e são elas a Técnica da Caixa Branca e a Técnica da Caixa Preta que respectivamente abaixo iremos abordá-las.

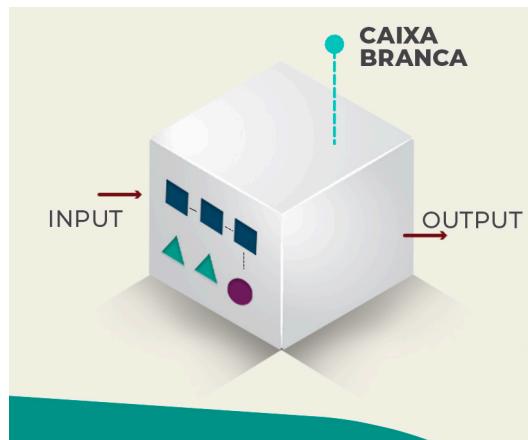
## Técnica da Caixa Preta



Disponível em: <<https://encurtador.com.br/gml13>>. Acesso em 27 ago. 2023.

A técnica da caixa preta é baseada na análise do software sem conhecimento do seu funcionamento interno. O testador trata o sistema como uma "caixa preta", ou seja, apenas observa as entradas fornecidas e as saídas geradas pelo software, sem considerar sua estrutura interna. O objetivo principal é verificar se o software atende aos requisitos funcionais e se executa as funções esperadas pelo usuário. Os casos de teste são projetados com base nas especificações do sistema e nos requisitos do usuário. Exemplos de técnicas da caixa preta incluem teste de equivalência, teste de limite, teste de caso de uso e teste de regressão.

## Técnica da Caixa Branca



Disponível em: <<https://encurtador.com.br/gml13>>. Acesso em 27 ago. 2023.

Nesse tipo de teste, o testador tem acesso ao código-fonte, à estrutura interna e à lógica do software. Com esse conhecimento, são criados casos de teste que visam exercitar caminhos específicos do código, como funções, ramificações condicionais e loops. A intenção é garantir uma

cobertura adequada do código e identificar erros ou falhas na implementação. Exemplos de técnicas da caixa branca incluem o teste de caminho básico, teste de ramificação, teste de condição e teste de fluxo de dados

Material Complementar: Automação de Testes com Selenium, Curso gratuito pelo Youtube sobre automatização de Testes com o Software Selenium.



Fonte do vídeo: [Automação em Teste de Software - Curso Gratuito de Selenium #05 | Inove Teste](#)



## RESUMO

O conceito de Qualidade de Software é de fundamental importância no desenvolvimento de sistemas, visando garantir que o produto final atenda às expectativas e necessidades dos usuários. Isso envolve a aplicação de técnicas específicas para assegurar que o software seja confiável, eficiente e livre de defeitos. Entre essas técnicas, destacam-se os testes funcionais, não funcionais e estruturais, que desempenham papéis distintos na avaliação do sistema.

Os testes funcionais concentram-se na verificação das funcionalidades do software de acordo com os requisitos definidos. Esses testes avaliam se o sistema se comporta conforme o esperado, executando ações específicas e validando os resultados obtidos. Já os testes não funcionais exploram aspectos como desempenho, segurança e usabilidade. Eles têm como objetivo medir a performance do sistema sob diferentes condições e assegurar que ele seja seguro e responsivo.

Por sua vez, os testes estruturais têm como foco a análise interna do software. Eles examinam a estrutura do código-fonte em busca de possíveis falhas, como erros de lógica, inconsistências e vulnerabilidades. A execução de testes funcionais e estruturais contribui para a melhoria contínua da qualidade do software, identificando defeitos tanto a nível de comportamento quanto de implementação. Dessa forma, os desenvolvedores podem corrigir problemas antes que eles afetem a experiência do usuário e comprometam a integridade do sistema.

Em resumo, o conceito de Qualidade de Software envolve a aplicação de técnicas de teste, como funcionais, não funcionais e estruturais, para avaliar diferentes aspectos do sistema. Essas abordagens garantem que o software atenda aos requisitos, seja seguro, eficiente e confiável. A combinação desses testes contribui para a entrega de um produto final de alta qualidade, minimizando riscos e maximizando a satisfação do usuário. Portanto, compreender e aplicar essas técnicas é essencial para garantir o sucesso no desenvolvimento de software no cenário atual.



## ATIVIDADE DE FIXAÇÃO

1. Segundo a ASQ, como a qualidade é definida? Cite os dois significados do termo.
2. Qual é a definição normativa de qualidade de acordo com a ISO 9000?
3. Quais são os benefícios da certificação da qualidade?
4. Explique o que é o sistema de certificação e descreva alguns passos que podem ocorrer nesse processo.
5. Quais são os custos da qualidade? Explique os custos de conformidade e os custos de não conformidade.
6. Cite três categorias de testes em qualidade de software
7. Explique a diferença entre teste funcional e teste estrutural.
8. O que são testes não funcionais? Dê exemplos de atributos não funcionais que podem ser testados.
9. O que é teste relacionado à mudança? Em que situações esse tipo de teste é realizado?
10. Verifique no QR code do Material complementar nesta página, pesquise sobre o Software Selenium, teste o software e explique sobre ele.

## TEMA 03

# Ciclos de Vida de Software

### Habilidades:

- Compreender os métodos de ciclo de um desenvolvimento de software;
- Verificar e validar correspondência entre especificação e o produto testado;

### Processo de desenvolvimento de software



Disponível em: <[macrovector-https://encurtador.com.br/m1789](https://macrovector-https://encurtador.com.br/m1789)>. Acesso em 27 ago. 2023.

Um processo de software é um conjunto de atividades e resultados associados que levam a produção de um produto de software. Esse processo pode envolver o desenvolvimento de software desde o início, embora, cada vez mais, ocorra o caso de um software novo ser desenvolvido mediante a expansão e a modificação de sistemas já existentes, isso é o tão famoso reuso.

Os processos de software são complexos e, como todos os processos intelectuais, dependem de julgamento humano. Por causa da necessidade de utilizar o julgamento e a criatividade, tentativas de automatizar processos de software têm tido sucesso limitado.

Uma razão pela qual existe uma abordagem limitada para a automação de processos é a imensa diversidade dos processos de software. Não há um processo ideal, e diferentes organizações

desenvolveram abordagens inteiramente diferentes para o desenvolvimento de software. Os procedimentos se desenvolveram para aproveitar o potencial das pessoas em uma empresa, bem como as particularidades dos sistemas em desenvolvimento. Portanto, mesmo dentro de uma única organização, é possível encontrar uma variedade de abordagens e métodos empregados.

Embora existam muitos processos de software diferentes, há atividades fundamentais comuns a todos eles, como:

**Especificação:** é preciso definir a funcionalidade do software e as restrições em sua operação.

**Projeto e implementação:** deve ser produzido o software de modo que cumpra sua especificação.

**Validação:** o software precisa ser validado para garantir que ele faça o que o cliente deseja.

**Evolução:** o software precisa evoluir para atender às necessidades mutáveis do cliente.

## Especificação de software

Destina-se a estabelecer quais funções são requeridas pelo sistema e as restrições sobre operação e o desenvolvimento do sistema. Essa atividade, atualmente, é frequentemente chamada de engenharia de requisitos; ela é um estágio particularmente importante do processo de software, uma vez que erros nesse estágio inevitavelmente produzem problemas posteriores no projeto e na implementação do sistema.

Esse processo leva a produção de uma documentação de requisitos, que é a especificação para o sistema. Os requisitos geralmente são apresentados em dois níveis de detalhes nesse documento. Os usuários finais e os clientes necessitam de uma declaração de alto nível dos requisitos; os desenvolvedores de sistema precisam de uma especificação mais detalhada do sistema.

Dentro deste processo existem quatro fases:

- Estudo de viabilidade;
- Levantamento e análise de requisitos;
- Especificação de requisitos;
- Validação de requisitos.

Naturalmente, as atividades no processo de requisitos não são realizadas simplesmente em uma sequência rigorosa. A análise de requisitos continua durante a definição e a especificação, e novos requisitos surgem ao longo do processo. Portanto, as atividades de análise, definição e especificação são intercaladas.

## Projeto e implementação

É o processo de conversão de uma especificação de sistema em um sistema executável. Esse estágio sempre envolve processos de projeto e programação de software, mas, se uma abordagem evolucionária de desenvolvimento for utilizada, ele poderá também envolver o aperfeiçoamento da especificação de software.

Um projeto de software é uma descrição de estrutura de software a ser implementada, dos dados que são parte do sistema, das interfaces entre os componentes do sistema e, algumas vezes, dos algoritmos utilizados. Os projetistas não conseguem que um projeto seja concluído

imediatamente, mas desenvolvem o projeto interativamente por meio de uma série de diferentes versões. O processo de projeto envolve acrescentar formas e detalhes, à medida que o projeto é desenvolvido, com 'retornos' constantes, a fim de corrigir projetos anteriores.

Uma especificação para o próximo estágio é a saída de cada atividade do projeto. Essa especificação pode ser uma especificação de como parte do sistema é implementada. À medida que o processo de projeto continua, essas especificações se tornam mais detalhadas. Os resultados finais de processo são especificações precisas de algoritmos e estruturas de dados a serem implementadas.

As atividades de processo de projeto são:

- Projeto de arquitetura;
- Especificação abstrata;
- Projeto de interface;
- Projeto de componentes;
- Projeto de estrutura de dados;
- Projeto de algoritmos.

## Métodos do projeto

Em muitos projetos de desenvolvimento de software, o projeto ainda é um processo utilizado à medida que for necessário. Partindo de um conjunto de requisitos, geralmente em linguagem natural, é preparado um projeto informal. A codificação se inicia, e o projeto é modificado à medida que o sistema é implementado. Existe pouco ou nenhum controle formal dessas modificações ou do gerenciamento de projeto.

Quando o estágio de implementação se completa, o projeto normalmente foi tão modificado, depois de sua especificação inicial, que o documento original de projeto é uma descrição incoerente e incompleta do sistema.

Uma abordagem mais metódica do projeto é proposta pelos "métodos estruturados", que são conjuntos de notações e diretrizes para o projeto de software.

O uso de métodos estruturados normalmente envolve a produção de modelos gráficos de sistema e resulta em grandes quantidades de documentação de projeto. Não se pode demonstrar que um método é melhor ou pior do que outros.

O sucesso ou fracasso dos métodos depende, com frequência, de sua adequação ao domínio de uma aplicação.

Um método estruturado inclui um modelo de processo de projeto, notações para representar o projeto, formulários de relatórios, regras e diretrizes do projeto. Embora haja um grande número de métodos, eles têm muito em comum. Os métodos estruturados podem aceitar alguns ou todos os seguintes modelos de sistema:

- Um modelo de fluxo de dados;
- Um modelo de relacionamento de entidades;
- Um modelo estrutural; Métodos orientados a objetos.

Métodos específicos complementam esses com outros modelos de sistemas, como os diagramas de transição de estado.

Na prática, a orientação dada pelos métodos é informal e, assim, diferentes projetistas desenvolveram diferentes projetos. Esses métodos nada mais são que boas práticas a serem seguidas.

## Programação e depuração

O desenvolvimento de um programa para implementar o sistema parte naturalmente dos processos de projeto de sistemas. Embora algumas classes de programas, como os sistemas em que a segurança é fundamental, sejam projetadas detalhadamente, antes que qualquer implementação tenha início, é mais comum que os estágios posteriores de desenvolvimento de projeto e de programa sejam intercalados.

A programação é uma atividade pessoal, e não existe um processo geral que seja normalmente seguido. Alguns programadores começarão com os componentes que eles compreendem bem e então prosseguirão com os componentes não tão fáceis.

Normalmente, programadores realizam alguns testes do código que eles mesmos desenvolveram. Isso, muitas vezes, revela defeitos dos programas que devem ser removidos.

Esse processo é chamado depuração. O depurador deve gerar hipóteses sobre o comportamento observável do programa e, então testá-las, na esperança de encontrar o defeito que causou a anomalia.

## Validação de Software

A validação, ou de modo mais geral, verificação e validação (V & V), destina-se a mostrar que um sistema está de acordo com suas especificações e que ele atende às expectativas do cliente comprador do sistema.

Esse processo envolve verificar processos, como por meio de inspeções e revisões, em cada estágio do processo de software. Desde a definição dos requisitos até o desenvolvimento do programa. A maior parte dos custos de validação, contudo, é observada depois da implementação, quando o sistema é testado.

## Documentos de Teste

Assim como o desenvolvimento deve planejar e documentar a sua parte, a equipe de testes deve fazer o mesmo. Abaixo uma pequena lista dos documentos mais utilizados:

**Plano de Teste:** É um documento com uma abordagem sistemática para o teste de sistemas como hardware ou software. Ele geralmente consiste numa modelagem detalhada do fluxo de trabalho durante o processo.

O plano de teste é um dos oito documentos descritos na IEEE 829, uma norma que especifica a forma e o conjunto de artefatos no teste de software. De acordo com ela, a estrutura do plano de teste consiste de uma série de seções descritas a seguir.

O documento começa com um identificador do documento de plano de teste, seguido do escopo, abordagem, recursos e cronograma das atividades de teste que se destina. Ele identifica, entre outros, itens de teste, recursos a serem testados, tarefas de teste, executor de cada tarefa, grau de independência do testador, o ambiente de teste, as técnicas de projeto de teste e critérios de entrada e de saída a serem usados, as razões de sua escolha, e os eventuais riscos que exigem planos

de contingência. É um registro do processo de planejamento de teste.

**Casos de Teste:** É um conjunto de condições usadas para teste de software. Ele pode ser elaborado para identificar defeitos na estrutura interna do software, através de situações que exercitem adequadamente todas as estruturas utilizadas na codificação; ou ainda, garantir que os requisitos do software que foi construído sejam plenamente atendidos. Podemos utilizar a ferramenta de casos de uso para criar e rastrear um caso de teste, facilitando assim a identificação de possíveis falhas.

No caso de teste deve conter precondições de execução, ações e valores de entrada, resultados esperados e pós-condições de execução desenvolvidas para um determinado objetivo ou condição de teste, tais como para exercitar o caminho de um determinado programa ou verificar o atendimento a um requisito específico.

**Reporte de Incidentes (bugs):** Descreve os bugs encontrados juntamente com os passos para reproduzir. Nele devem constar o escopo (o que realmente deveria ter ocorrido), e o impacto que o mesmo causa.

**Métricas, estatísticas e resumos:** Transmitem o progresso dos testes. São utilizados gráficos, tabelas e relatórios escritos. Geralmente utilizados para o gerenciamento.

**Material Complementar:** Processo de Desenvolvimento de Software - Acompanhe o Vídeo para saber mais detalhes sobre como é implementado o Desenvolvimento em si através de Fluxos e Gerenciamento de Processos.



Fonte do vídeo: [Processo de desenvolvimento de software](#)

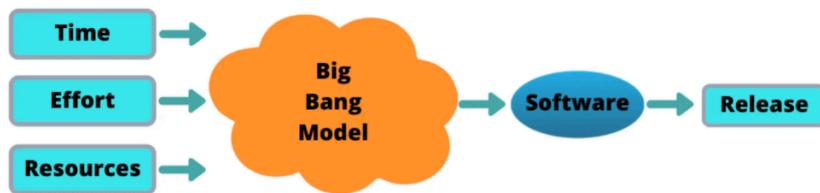
## Modelos de Ciclo de Vida de Desenvolvimento de Software

Por mais simples ou complexo que possa parecer, um modelo de ciclo de vida de um projeto é, de fato, uma versão simplificada da realidade. É suposto ser uma abstração e, tal como todas as boas abstrações, apenas a quantidade de detalhe necessária ao trabalho em mãos deve ser incluída. Qualquer organização que deseje por um modelo de ciclo de vida em prática irá necessitar adicionar detalhes específicos para dadas circunstâncias e diferentes culturas. Por exemplo, a Microsoft quis manter uma cultura de pequena equipe e ao mesmo tempo tornar possível o desenvolvimento de grandes e complexos produtos de software.

Existem alguns modelos mais frequentemente utilizados, a maioria dos outros são apenas variações e aperfeiçoamento destes:

- Modelo Big Bang;
- Modelo Constrói e corrige;
- Modelo Cascata;
- Modelo Espiral;
- Métodos ágeis.

## Modelo Big Bang

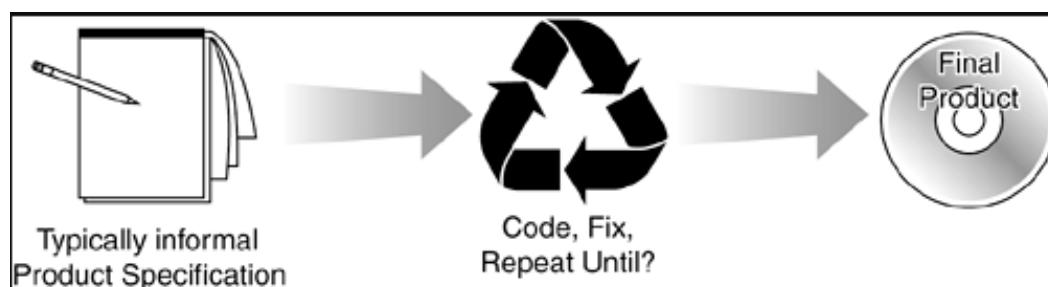


**Big Bang SDLC MODEL**

Disponível em: <<https://encurtador.com.br/apy79>>. Acesso em 27 ago. 2023.

Como na teoria de criação do universo, o software é produzido a partir de uma 'explosão'. Este modelo não possui planejamento, cronograma ou processo formal de desenvolvimento. Todo o esforço é gasto com codificação. Não existe também a fase de teste. Quando executado, é feito no final, pouco antes da entrega. O analista de testes tem a missão de encontrar as falhas e reportar. Mas estas falhas não serão necessariamente corrigidas. Muitas vezes o resultado do produto desenvolvido não será o esperado pelo cliente.

## Modelo Constrói e Corrige



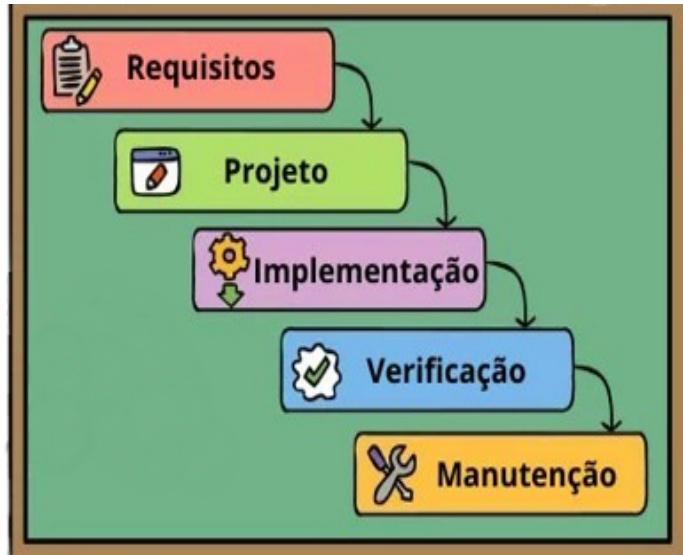
Disponível em: <<https://encurtador.com.br/apy79>>. Acesso em 27 ago. 2023.

Derivado do modelo Big Bang, neste modelo é feito um projeto simples do software e então é iniciado o desenvolvimento. Após uma fase de testes, normalmente informal, onde as falhas são encontradas, existe a fase de correção. Este ciclo se repete até que o software seja considerado satisfatório para ser entregue.

Idealmente, deve ser utilizado em projetos pequenos que serão descartados em seguida tais

como protótipos e demos. No entanto, muitas empresas o utilizam como modelo default, um caso que geralmente há sucesso na entrega, porém alto custo de desenvolvimento.

## Modelo Cascata



Fonte: Elaborado pelo autor (2023).

O modelo de ciclo de vida Cascata é um dos mais antigos e tradicionais para o desenvolvimento de software. Ele é caracterizado por uma abordagem sequencial e linear, onde cada fase é realizada de forma ordenada, seguindo uma sequência fixa.

No modelo Cascata, o processo de desenvolvimento de software é dividido em etapas distintas que incluem análise de requisitos, design, implementação, testes e manutenção. Cada fase é iniciada somente após a conclusão da fase anterior, formando uma cascata descendente de atividades.

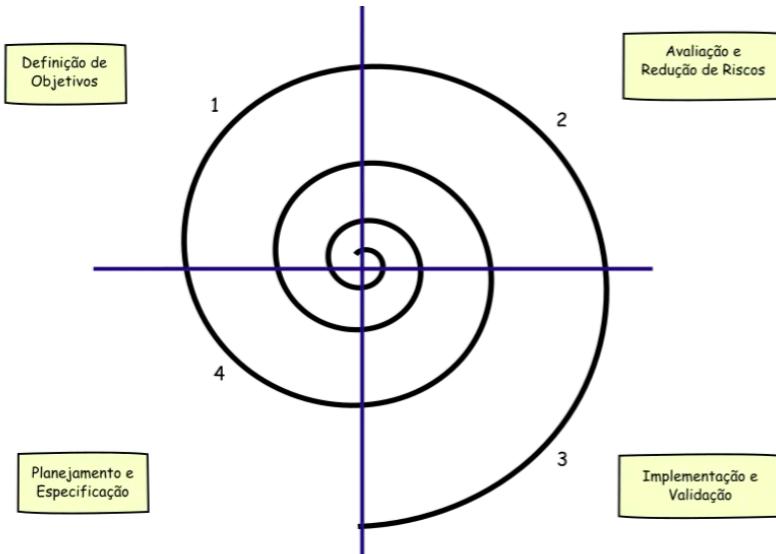
Na fase de análise de requisitos, os requisitos do software são coletados e documentados. Com base nesses requisitos, o design do sistema é elaborado na próxima fase, definindo a arquitetura, estrutura e funcionalidades do software. Em seguida, a implementação ocorre, onde o código-fonte é escrito e o software é construído.

Após a implementação, a fase de testes é realizada para verificar se o software atende aos requisitos e funciona corretamente. Nessa etapa, são identificados e corrigidos eventuais erros e falhas encontradas. Por fim, na fase de manutenção, o software é aprimorado, corrigido e atualizado ao longo do tempo para atender às necessidades em constante evolução.

O modelo Cascata é conhecido por sua abordagem estruturada e linear, o que permite uma visão clara do progresso do projeto. No entanto, uma desvantagem é que ele pressupõe que todos os requisitos sejam conhecidos e definidos desde o início, o que nem sempre é realista. Alterações ou ajustes nos requisitos podem ser difíceis de serem incorporados após a conclusão de uma fase.

Por isso, o modelo de ciclo de vida em cascata é mais adequado para projetos com requisitos estáveis e bem definidos, onde a linearidade e a sequencialidade são vantajosas.

## Modelo Espiral



Disponível em: <<https://medium.com/contexto-delimitado/o-modelo-em-espiral-de-boehm-ed1d85b7df>>. Acesso em 27 ago. 2023.

Este modelo resolve muitos dos problemas dos modelos anteriores. No início, apenas as funcionalidades importantes são definidas e testadas. Somente após a finalização dos testes e correção das falhas o próximo ciclo se inicia definindo-se mais algumas funcionalidades. Estes ciclos se repetem até que o software esteja pronto.

Cada ciclo do espiral possui seis passos:

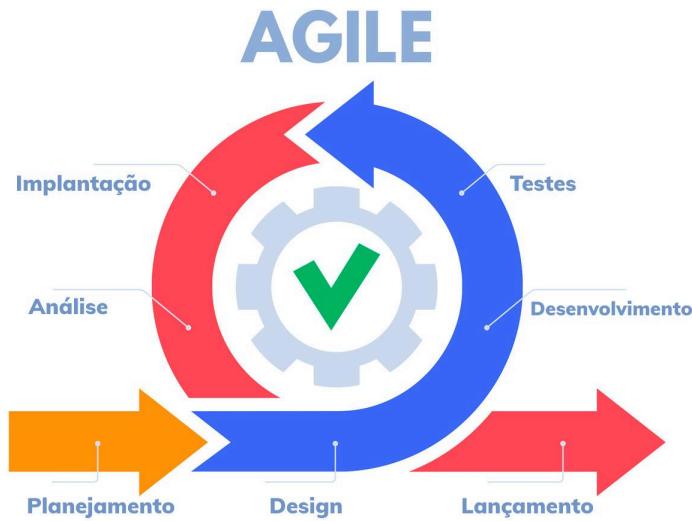
- Determinar os objetivos, alternativas e restrições
- Identificar e resolver os riscos
- Avaliar as alternativas
- Desenvolver e testar o ciclo atual
- Planejar o próximo ciclo
- Definir as técnicas para o próximo ciclo

Permite que a equipe de testes participe do desenvolvimento do software desde o início. Os testes podem ser planejados aos poucos e garante que as partes críticas do sistema serão mais bem testadas.

## Métodos Ágeis

O modelo de ciclo de vida Ágil é uma abordagem moderna e iterativa para o desenvolvimento de software, que se baseia em princípios de flexibilidade, colaboração e adaptação contínua. Diferente do modelo Cascata, os Métodos Ágeis são mais flexíveis e respondem às mudanças de forma ágil ao longo do processo de desenvolvimento.

Os Métodos Ágeis são baseados em ciclos de desenvolvimento curtos, chamados de iterações ou sprints, que geralmente têm duração de uma a quatro semanas. Cada iteração envolve o planejamento, a execução, a revisão e o ajuste das atividades de desenvolvimento do software.



Disponível em: <<https://encurtador.com.br/apy79>>. Acesso em 27 ago. 2023.

Uma das metodologias ágeis mais conhecidas é o Scrum. No Scrum, uma equipe multidisciplinar é organizada em papéis específicos, como o Product Owner, o Scrum Master e os membros do time de desenvolvimento. O trabalho é dividido em pequenas tarefas, chamadas de User Stories, que são priorizadas e adicionadas ao Backlog do Produto.

Durante cada iteração, a equipe seleciona um conjunto de User Stories do Backlog do Produto para serem trabalhadas. Essas User Stories são então desenvolvidas, testadas e entregues como incrementos de software funcional. No final de cada iteração, há uma reunião de revisão, na qual o progresso é avaliado e feedback é recebido para orientar o próximo ciclo de desenvolvimento.

Os Métodos Ágeis enfatizam a colaboração constante entre os membros da equipe e a participação ativa dos stakeholders. Além disso, eles valorizam a entrega contínua de valor ao cliente, permitindo que os requisitos sejam ajustados e refinados ao longo do tempo, conforme necessário.

Ao adotar os Métodos Ágeis, as equipes são incentivadas a serem autônomas, auto-organizadas e a buscar a melhoria contínua. A transparência e a comunicação eficaz são fundamentais para o sucesso desses métodos, pois permitem que os membros da equipe acompanhem o progresso, identifiquem obstáculos e tomem decisões informadas.

Os Métodos Ágeis são amplamente utilizados na indústria de desenvolvimento de software, especialmente em projetos complexos e com requisitos voláteis. Eles permitem uma resposta rápida a mudanças, maior eficiência, qualidade aprimorada e maior satisfação do cliente.



## RESUMO

Neste capítulo entendemos que o processo de desenvolvimento de software é complexo e envolve diversas atividades, como especificação, projeto, implementação, validação e evolução. Não existe um processo ideal, e diferentes organizações adotam abordagens diferentes para o desenvolvimento de software. Apesar disso, existem atividades fundamentais comuns a todos os processos, como a especificação das funcionalidades do software, o projeto e implementação do

sistema, a validação para garantir que o software atenda aos requisitos do cliente e a evolução para acompanhar as necessidades em constante mudança.

Além disso, existem modelos de ciclo de vida de desenvolvimento de software, como o modelo em cascata, o modelo espiral e os métodos ágeis, cada um com suas características e aplicabilidades. A escolha do modelo adequado depende das necessidades e requisitos do projeto. É importante ressaltar que o desenvolvimento de software requer planejamento, documentação e testes adequados para garantir a qualidade e eficácia do produto final.



## ATIVIDADE DE FIXAÇÃO

- 1.** Quais são as atividades fundamentais comuns a todos os processos de desenvolvimento de software?
- 2.** Qual é a finalidade da especificação de software?
- 3.** Quais são as fases envolvidas no processo de requisitos?
- 4.** O que é projeto de software e quais são as atividades envolvidas nessa fase?
- 5.** Quais são os modelos de ciclo de vida de desenvolvimento de software mencionados no texto?
- 6.** Descreva brevemente o modelo Big Bang de ciclo de vida de desenvolvimento de software.
- 7.** Quais são as principais vantagens e desvantagens do modelo Espiral de ciclo de vida de desenvolvimento de software?
- 8.** Descreva o conceito de Métodos Ágeis e explique por que eles são amplamente utilizados na indústria de desenvolvimento de software.
- 9.** Explique o processo de validação de software e por que é importante.
- 10.** Compare e contraste o modelo Cascata com os Métodos Ágeis em termos de abordagem, flexibilidade e resposta a mudanças

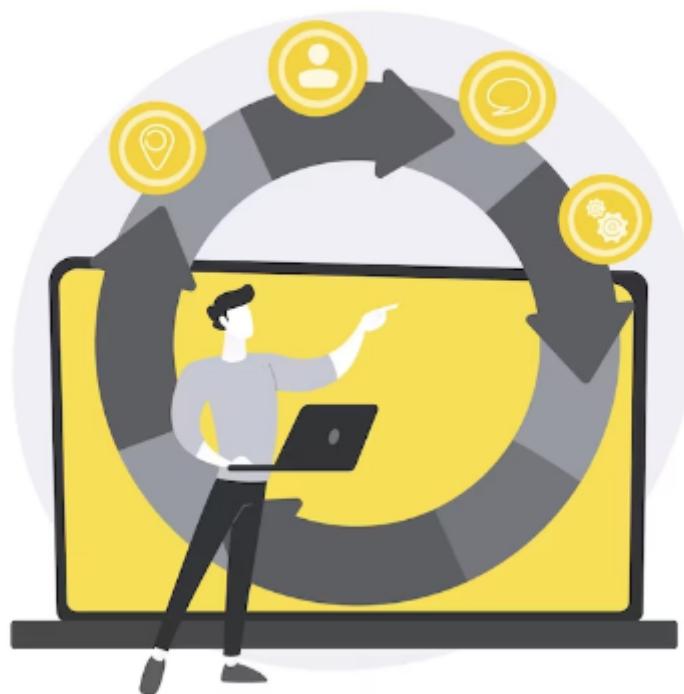
## TEMA 04

# Rational Unified Process (RUP) e TMap (Testing Management Approach)

### Habilidades:

- Compreender as necessidades dos usuários de sistemas;

### Processo Unificado (Rational Unified Process- RUP)



Disponível em: <[vectorjuice-https://tinyurl.com/ydhn4pby](https://tinyurl.com/ydhn4pby)>. Acesso em 27 ago. 2023.

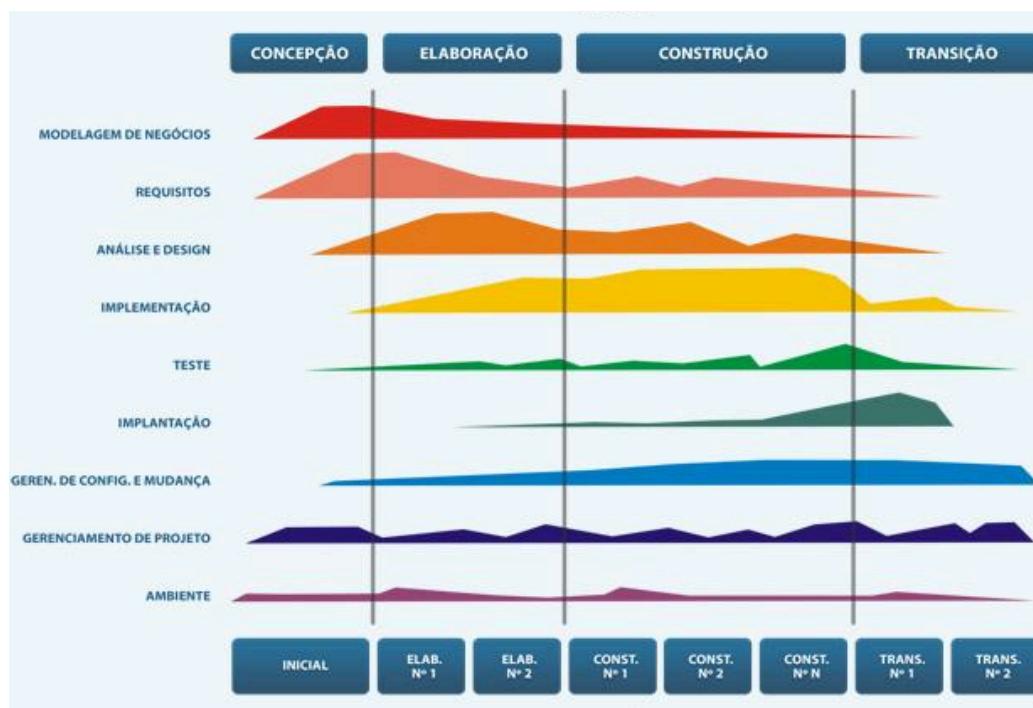
O Processo Unificado da Rational conhecido como RUP (Rational Unified Process), é um processo de engenharia de software criado para apoiar o desenvolvimento orientado a objetos, fornecendo uma forma sistemática para se obter vantagens no uso da UML. Foi criado pela Rational Software Corporation e adquirido em fevereiro de 2003 pela IBM.

O principal objetivo do RUP é atender as necessidades dos usuários garantindo uma produção de software de alta qualidade que cumpra um cronograma e um orçamento previsíveis. Assim, o RUP mostra como o sistema será construído na fase de implementação, gerando o modelo do projeto e, opcionalmente, o modelo de análise que é utilizado para garantir a robustez. O RUP define perfeitamente quem é responsável pelo que, como as coisas deverão ser feitas e quando devem ser realizadas, descrevendo todas as metas de desenvolvimento especificamente para que sejam

alcançadas.

O RUP organiza o desenvolvimento de software em quatro fases, onde são tratadas questões sobre planejamento, levantamento de requisitos, análise, implementação, teste e implantação do software. Cada fase tem um papel fundamental para que o objetivo seja cumprido, distribuídos entre vários profissionais como o Analista de sistema, Projetista, Projetista de testes, entre outros.

## Fases do RUP



Disponível em: <<http://www.sgp-ti.uerj.br/site/metodologia/>>. Acesso em 27 ago. 2023.

✓ **Fase de Concepção / Iniciação:** Esta fase do RUP abrange as tarefas de comunicação com o cliente e planejamento. É feito um plano de projeto avaliando os possíveis riscos, as estimativas de custo e prazos, estabelecendo as prioridades, levantamento dos requisitos do sistema e preliminarmente analisá-lo. Assim, haverá uma anuência das partes interessadas na definição do escopo do projeto, onde são examinados os objetivos para se decidir sobre a continuidade do desenvolvimento.

✓ **Fase de Elaboração:** Abrange a Modelagem do modelo genérico do processo. O objetivo desta fase é analisar de forma mais detalhada a análise do domínio do problema, revisando os riscos que o projeto pode sofrer e a arquitetura do projeto começa a ter sua forma básica. Indagações como “O plano do projeto é confiável?”, “Os custos são admissíveis?” são esclarecidas nesta etapa.

✓ **Fase de Construção:** Desenvolve ou adquire os componentes de Software. O principal objetivo desta fase é a construção do sistema de software, com foco no desenvolvimento de componentes e outros recursos do sistema. É na fase de Construção que a maior parte de codificação

ocorre.

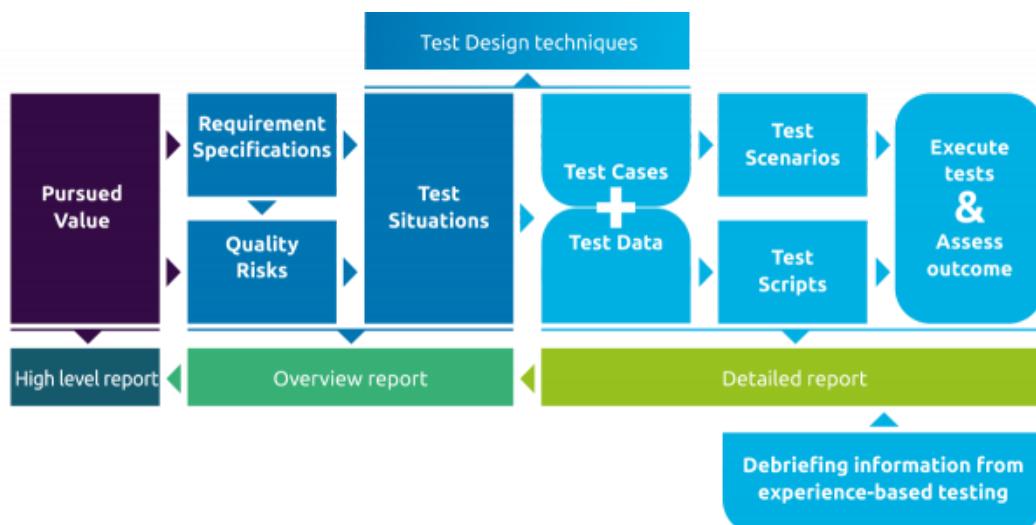
✓ **Fase de Transição:** Abrange a entrega do software ao usuário e a fase de testes. O objetivo desta fase é disponibilizar o sistema, tornando-o disponível e compreendido pelo usuário final. As atividades desta fase incluem o treinamento dos usuários finais e também a realização de testes da versão beta do sistema visando garantir que o mesmo possua o nível adequado de qualidade.

**Material Complementar:** Engenharia de Software - Rational Unified Process (RUP)- Acompanhe o vídeo para saber mais detalhes sobre RUP onde surgiu , quais implementações e utilizações do dia a dia.



Fonte do vídeo: Engenharia de Software - Rational Unified Process (RUP)

## TMap (Testing Management Approach)



Disponível em: <<https://www.tmap.net/building-blocks/test-approaches>>. Acesso em 27 ago. 2023.

## Teste de Software → Definições

Atualmente não há definições de termos em teste de software aceitas universalmente (padrão mundialmente conhecido e aceito).<https://www.tmap.net/building-blocks/test-approaches>

No entanto, se pesquisarmos, encontramos diversas definições sobre o que é teste de software:

- “É analisar um programa com a intenção de descobrir erros e defeitos.” (Myers)
- É exercitar ou simular a operação de um programa ou sistema.
- É medir a qualidade e funcionalidade de um sistema.
- É avaliar se o software está fazendo o que deveria fazer, de acordo com os seus requisitos, e não está fazendo o que não deveria fazer;
- “É qualquer atividade que a partir da avaliação de um atributo ou capacidade de um programa ou sistema seja possível determinar se ele alcança os resultados desejados.” (Bill Hetzel).

**DICA:** Testes servem para evitar problemas na produção.

*“Uma pessoa inteligente resolve um problema. Uma pessoa sábia evita-o”. (Einstein)*

Teste de software é uma das atividades executadas dentro do processo de desenvolvimento de software, ele é feito para determinar se certo produto atingiu suas especificações e funcionou corretamente no ambiente para o qual foi projetado.

O seu objetivo é revelar falhas em um produto, para que as causas dessas falhas sejam identificadas e possam ser corrigidas pela equipe de desenvolvimento antes da entrega final.

O conceito de teste de software pode ser entendido como uma visão intuitiva ou mesmo de uma maneira formal. Existem atualmente várias definições para esse conceito.

De uma forma simples, testar um software significa verificar através de uma execução controlada se o seu comportamento corre de acordo com o especificado.

O objetivo principal desta tarefa é revelar o número máximo de falhas dispondo do mínimo de esforço, ou seja, mostrar aos que desenvolvem se os resultados estão ou não de acordo com os padrões estabelecidos.

## TMMI (Test Maturity Model integration)

Já que estamos falando da importância da Atividade de Teste, não podemos esquecer do TMMI (Test Maturity Model Integration), que foi desenvolvido pela TMMI Foundation como um guia e framework para apoiar a melhoria de processos de teste.

TMMI, que é modelo de maturidade teste. Sendo modelo de maturidade teste, o TMMI vai estabelecer requisitos que são necessários de serem contemplados para que eu tenha processo de teste com qualidade.

O TMMI, assim como outros modelos de maturidade, como o CMMI, que é modelo de maturidade de processo de software, é composto por cinco níveis de maturidade, que vai do nível ao nível cinco, sendo que o nível inicial, é considerado o nível caótico.

Todos os processos de teste estão enquadrados nesse nível porque ele não requer que qualquer coisa seja contemplada. Já no nível cinco, que é o nível otimização, o processo de teste já tem bastante qualidade e o objetivo é manter essa qualidade evoluindo esse processo constantemente.

Todos os níveis, do dois ao cinco, são compostos por requisitos que devem ser contemplados, que devem ser executados no processo de teste. Esses requisitos eles são denominados áreas de processo. Então, como a gente pode ver, todos os níveis, a partir do nível dois, eles contêm áreas de processo.

Sendo assim, a área de processo, ela é então conjunto de práticas específicas relacionadas entre si, que uma vez contempladas em conjunto, elas alcançam os objetivos específicos, os quais ajudam a melhorar o processo de teste da organização relação àquela área de processo.

Já os objetivos genéricos e as práticas genéricas, são comuns a várias áreas de processo e elas descrevem características que podem ser utilizadas para institucionalizar o processo de teste dentro da organização.

## Como posso usar o modelo TMMi?

Então, o modelo pode ser usado com propósitos diferentes. A organização pode querer uma certificação num determinado nível de maturidade. Então ela vai olhar para o TMMi, verificar o que ela já cumpre do TMMi no seu processo de teste, para ela poder arrumar o seu processo de teste e contemplar aquilo que ela não está fazendo para alcançar determinado nível de maturidade.

A outra alternativa, a organização não tem processo de teste e ela pode pegar o TMMi como referencial para montar o seu processo de teste. Então ela quer organizar o processo de teste, ela vai olhar para esse modelo, que é modelo que estabelece a qualidade no processo de teste, e vai montar o seu processo de teste já tendo o TMMi como referência.

O TMMi possui 81 práticas específicas. Ele é um modelo pesado em termos de requisitos que devem ser contemplados. Às vezes a organização não possui uma infraestrutura suficiente que permita adotar todo o TMMi. Às vezes a organização também não quer adotar o modelo por completo, ela quer processo menor, processo mais enxuto.

## Para que testar?

É necessário testar por diversos motivos: por questões de negócio, de qualidade, de custos, de segurança e de confiabilidade.

Poucas empresas utilizavam softwares até a metade da década de 90. Atualmente, é muito raro alguma empresa não possuir algum software para suportar as suas atividades/necessidades.

Softwares tornam-se cada vez mais parte do nosso dia a dia, desde aplicações comerciais (ex.: bancos, indústrias em geral, comércio, etc.) até produtos de consumo (ex.: aviões, automóveis, navios).

A maioria de nós já passou por alguma situação em que algum software não funcionou conforme esperado, seja por indisponibilidade, lentidão de sites, de sistemas, etc.

Softwares que não funcionam de forma correta podem gerar muitos problemas, incluindo financeiro, tempo e reputação das empresas. Podendo, inclusive, chegar a influenciar na integridade das pessoas.

O principal motivo de porquê devemos testar é que o ser humano está sujeito a cometer erros (engano), o que pode acarretar um defeito (falha, bug), em um código, em um software ou em um documento como os de especificações, por exemplo.

Se um defeito no código for executado, o sistema falhará ao tentar fazer o que deveria (ou, em algumas vezes, o que não deveria), causando uma falha. Defeitos no software ou documentos causam falhas, mas nem todos os defeitos causam falhas.

Os defeitos ocorrem porque nós somos passíveis de falha e porque existe pressão no prazo,

códigos complexos, complexidade ou carência de infraestrutura, mudanças na tecnologia e/ou muitas interações de sistema.

Falhas também podem ocorrer por condições do ambiente tais como: radiação, magnetismo, campos eletrônicos e poluição, que podem causar falhas em software embarcado (firmware) ou influenciar a execução do software por alguma alteração no hardware.

Testes rigorosos em sistemas e documentações podem reduzir os riscos de ocorrência de problemas no ambiente operacional, e contribui para a qualidade dos sistemas de software se os defeitos forem encontrados e corrigidos antes de serem implantados em produção.

O teste de software pode também ser necessário para atender requisitos contratuais ou legais ou determinados padrões de mercado.

## Confiabilidade



Disponível em: <[rawpixel-https://tinyurl.com/3y6kp38z](https://tinyurl.com/3y6kp38z)>. Acesso em 27 ago. 2023.

Testamos os softwares para construir confiabilidade. Confiabilidade do software é a probabilidade de que o software não causará uma falha no sistema por um tempo especificado, sob condições específicas.

É necessário definir que condições o software atingirá o nível de confiabilidade prometido. Por exemplo, um sistema pode atingir a confiabilidade de não apresentar mais que uma falha por mês se usado por no máximo 10 usuários simultâneos.

- A confiabilidade do software aumenta à medida que o software é executado ao longo do tempo, sem apresentar falhas.



Fonte: Elaborado pelo autor (2023).

- Podemos afirmar também que a confiabilidade do software é medida através do tempo médio entre falhas

O resultado da execução dos testes pode representar confiança na qualidade do software caso sejam encontrados poucos ou nenhum defeito. Um teste projetado adequadamente e cuja execução não encontra defeitos reduz o nível de riscos em um sistema. Por outro lado, quando os testes encontram defeitos, a qualidade do sistema aumenta quando estes são corrigidos.

## Por que a atividade de Teste de Software é tão necessária e importante?

Bem, se fizermos essa pergunta a um estudante, ele provavelmente dirá que é importante testar para evitar que o professor que solicitou o algoritmo/software encontre problemas.

Se perguntarmos a um gerente de TI de uma grande empresa, ele certamente dirá que é uma forma de aumentar a qualidade de seus produtos e, com isso, a confiabilidade sob o ponto de vista dos clientes, mantendo-se competitivo no mercado.

De forma geral, a atividade de teste é necessária porque quem desenvolve software somos nós, humanos. E humanos podem cometer erros. E erros podem ocorrer por diversos fatores: complexidade do software desenvolvido, mudanças necessárias em software já existente e que pode levar a novas falhas, falta de uso de métodos que apoiem a validação e verificação da qualidade do software, situações imprevisíveis, questões pessoais e do próprio ambiente de trabalho e que interferem no desempenho do testador, dentre outras questões.

Considerando a importância da atividade de software sob um ponto de vista mais técnico e de negócio, ela é requerida para revelar a presença de defeitos, também pode ser usada para aumentar a confiança e satisfação do cliente com o software, e é útil para assegurar a qualidade do produto.

É essencial prever ou garantir que falhas não ocorram, ou que pelo menos sejam minimizadas ou identificadas o quanto antes.

Falhas em softwares são extremamente perigosas. Elas podem causar perdas monetárias e humanas irreversíveis, e a história está cheia de tais exemplos.



Disponível em: <<https://tinyurl.com/cd7m3fff>>. Acesso em 27 ago. 2023.

Em abril de 2015, partes das atividades do centro financeiro de Londres foram paralisadas devido à falha de hardware e software envolvendo a Bloomberg, empresa que presta serviço de tecnologia e gerenciamento de dados para o mercado financeiro. Cerca de 320,000 pessoas ao redor do mundo usavam os terminais da *Bloomberg terminals*, o que custava cerca de \$20,000 por ano. Um número significante afetado pelo problema tecnológico.



Disponível em: <<https://revistacarro.com.br/recall-da-nissan-atinge-3-5-milhoes-de-carros/>>. Acesso em 27 ago. 2023

Em abril de 2016, a Nissan anunciou um recall para mais de 3 milhões de carros, a maioria nos Estados Unidos, devido à falha de software nos sensores de airbag no banco do passageiro. Em resumo, o sistema dos carros afetados estava classificando inadequadamente quem estava sentado no banco do passageiro, se um adulto ou uma criança, ou ainda se o banco estava vazio. Como consequência, os airbags não iriam inflar. Foram identificados acidentes devido a essa falha de software.

Então, é muito importante que você, enquanto gerente de uma empresa de software, estudante da área de tecnologia, profissional, educador e/ou pesquisador da área de software, esteja ciente da importância da atividade de Teste de Software. E, mais do que isso, reconheça o valor do profissional que atua nessa área, seja no mercado de trabalho ou na academia.

## Utilização do Teste como Prática Estratégica no Mercado

Pense que empresas visam o crescimento e consequentemente o lucro. Para alcançarem isso querem seus clientes satisfeitos, seus clientes satisfeitos compram novos produtos e divulgam a sua satisfação.

Embora ainda existam empresas no mercado que prefiram não investir em qualidade como um todo, a maioria já entendeu que sem qualidade mínima de uso perde-se clientes.

Logo, começam a pensar de que forma o teste pode auxiliar no processo em direção ao sucesso dos seus projetos e produtos.

Existem muitos métodos hoje em dia que podem facilitar o entendimento do testador sem depender apenas de um treinamento e estudo sobre o que será desenvolvido.

Não citarei um específico no momento, mas aproximar o analista de teste, ou o testador, e envolvê-lo desde o início no projeto mostra uma significante melhora no entendimento de negócio, maior correção de definições incoerentes e com isso mais qualidade no produto.

Obviamente que escolhendo isso os custos do projeto podem subir além do normal, no entanto se lembrarmos que um software bem planejado e testado da melhor forma reduz o risco de falhas/defeitos, isso retorna como um investimento visto que falhas/defeitos em produção podem trazer prejuízos imensuráveis aos clientes e as empresas.

## Abrangência dos Testes de Software

Como saber o quanto de teste é suficiente? Para decidir isso devemos levar em conta os riscos, incluindo risco técnico, do negócio e do projeto, além das restrições do projeto como tempo e orçamento.

O teste deve prover informações suficientes aos interessados (stakeholders) para tomada de decisão sobre a distribuição do software ou sistema, para as próximas fases do desenvolvimento ou implantação nos clientes

Se os testes forem executados apenas ao fim do processo de desenvolvimento, certamente a abrangência será bem menor que se forem executados em andamento no processo de desenvolvimento.

Não podemos esquecer que parte do trabalho dos testes é de conferir se os requisitos estão conforme o solicitado. Por isso o ideal é que o analista de teste acompanhe desde o início do projeto,

muitas documentações são incompletas, dão ambiguidade no entendimento e podem ocultar informações.

Quando desenvolvimento e testes andam juntos a chance de se testar com capacidade e cobertura é muito maior. Com isso os riscos reduzem e o profissional de teste pode fazer seu trabalho com calma.

A cobertura de 100% não existe, mas podemos chegar perto disso. Assim como os testes não resolverão todos os problemas de um software, mas podem reduzir as chances de eles serem encontrados aos clientes.

## Teste e Qualidade

Teste é igual a qualidade? Não

Se testarmos o software, teremos qualidade? Não necessariamente... Lembrando-se do conceito de qualidade que já vimos, podemos dizer que:

O teste por si só não constrói a qualidade do software.

O teste tem a função de ajudar a medir a qualidade:

- À medida que os defeitos encontrados sejam corrigidos avaliando requisitos funcionais do software;
- Avaliando requisitos não funcionais;
- O teste pode fornecer confiança na qualidade do software.

Com a ajuda do teste é possível medir a qualidade do software em termos de defeitos encontrados, por características e requisitos funcionais ou não funcionais do software (confiabilidade, usabilidade, eficiência, manutenção e portabilidade).

DICA:

Teste de Software é uma parte de Controle da Qualidade, que faz parte da Garantia da Qualidade de Software.

O Controle da Qualidade é **orientado a detecção destes defeitos**.

A Garantia da Qualidade de Software é **orientada através da prevenção de defeitos**.

## Os Papéis em Teste de Software



Fonte: Elaborado pelo autor (2023).

**Analista de Qualidade (Quality Assurance - QA):** Profissional responsável pela realização dos testes do software, é importante enfatizar que nas metodologias ágeis a qualidade é responsabilidade de todos, porém o QA faz o papel de guardião. O QA normalmente domina técnicas de teste unitário, aceitação, integração, carga, entre outros. Muitos QA também sabem desenvolver e acabam contribuindo com o desenvolvimento do software quando possível.

**Scrum Master / Agile Master:** O Scrum Master é responsável pela cultura ágil dentro do time e por disseminar todas as práticas do framework Scrum. Ele atua como um líder do processo Scrum é um facilitador, contribuindo na resolução de conflitos e na remoção de impedimentos. O Agile Master exerce as mesmas atividades do Scrum Master, com a diferença que também domina demais metodologias de desenvolvimento ágil como Kanban ou XP.

**Product Owner (Dono do Produto):** Responsável por decidir quais os recursos e funcionalidades o produto deve ter, e qual a ordem de prioridade que devem ser desenvolvidos. Também mantém e comunica uma visão clara do que o Time Scrum está trabalhando no produto, sendo um ponto de intersecção entre a área de negócio e a área de desenvolvimento.

**Analista de Segurança:** Com o objetivo de olhar para a segurança na organização, o Analista de Segurança é fundamental para promover a Cultura do Dev SecOps e trazer o quanto antes a preocupação com segurança para o início do processo de trabalho, identificando, protegendo, detectando, respondendo e recuperando os pontos necessários sobre segurança.

**Testador, Homologador ou Tester:** é o profissional que tem como responsabilidade principal executar os testes planejados pelo Analista de Teste e, se for o caso, reportar os erros encontrados.

**Analista de Teste, Analista de Homologação ou Projetista de Teste:** tem como premissa ser mais experiente que o Testador, o ideal é que tenha sido um. É ele quem elabora os Casos de Teste que serão executados pelo Testador. Em alguns casos, põe a mão na massa e executa os testes.

**Automatizador ou Analista de Teste de Automação:** é quem transforma os Casos de Teste elaborados pelo Analista de Teste, em scripts automatizados. Além de automatizar os testes funcionais é também responsável pelos testes de Desempenho, Performance e Carga. É fundamental que conheça bem linguagens de programação.

**Arquiteto de Teste ou Engenheiro de Teste:** é responsável pela criação e manutenção de toda a infraestrutura de teste, incluindo o ambiente, a arquitetura e as ferramentas de teste.

**Líder de Teste ou Coordenador de Teste:** é o profissional responsável pela coordenação dos testes. É ele quem estima os esforços, os recursos e o tempo de teste. É também quem cria e mantém o Plano de Teste e atribui as tarefas para os demais membros da equipe.

**Gerente de Teste:** é a pessoa central que trata de todos os assuntos relacionados ao Teste de Software. É, normalmente, responsável pela definição da política de teste usada na organização, incluindo: planejamento de testes, documentação dos testes, controle e monitoramento dos testes, aquisição de ferramentas de teste, participação em inspeções e revisões do trabalho de teste.



Disponível em: <[rawpixel-https://tinyurl.com/4jmp8a4j](https://tinyurl.com/4jmp8a4j)>. Acesso em 27 ago. 2023.

Raramente um Testador após alguns meses de empresa continua sendo apenas o executor de tarefas. Às vezes não há nem um período de adaptação e ele já começa a executar as funções do Analista de Teste, logo no começo das atividades.

Além de elaborar os Casos de Teste, é muito comum ver um Analista de Teste se enveredar pelos caminhos da automação, utilizando (inicialmente) recursos de ferramentas “Record & Play” que

gravam as ações dos usuários na interface (através de mouse e teclado) e depois repetem as operações; fazendo assim o papel de um Automatizador.

Quanto ao Automatizador, por já dominar alguma linguagem de programação, esse profissional geralmente vem do desenvolvimento. E na grande maioria das vezes não iniciou carreira no Teste de Software.

O Arquiteto de Teste além de preparar toda a infraestrutura da “área técnica” do Teste, em diversos momentos executa o papel do Líder de Teste, auxiliando também na “área comportamental” como seleção de profissionais, acompanhamento de atividades, distribuição de tarefas, solução de conflitos.

O Líder de Teste, assim como Arquiteto de Teste, costuma também exercer o papel de Gerente de Teste, realizando definições gerais do planejamento dos Testes na empresa.

Além disso, observamos que em algumas empresas os profissionais do Teste não só executam funções dos diversos cargos da área, como também de outras. Por exemplo: ministram treinamentos, implantam sistemas, realizam levantamento de requisitos, visitam clientes, entre diversas outras atividades.

Neste caso, para “gerenciar” é preciso no mínimo saber como se faz. Sendo assim, é fundamental que um Analista de Teste tenha sido um Testador, que um Líder ou Arquiteto tenha sido um Analista e que um Gerente tenha exercido pelo menos alguns papéis no Teste de Software.

Algumas empresas preferem trazer profissionais de outras áreas como Help Desk, Implantação, Desenvolvimento, para os cargos da área de Teste simplesmente por entenderem que para testar, conhecer o negócio é mais do que suficiente.

Isso é muito relativo, do que adianta conhecer do negócio se o profissional não tem o feeling de Testador?! É mais difícil ensinar um profissional a testar do que ensiná-lo o Negócio.

Para exercer qualquer das funções acima, uma condição básica para o profissional é ser:

- Investigador
- Criativo
- Questionador
- Detalhista



## RESUMO

Neste capítulo entendemos que o RUP (Rational Unified Process) é um processo de engenharia de software que visa apoiar o desenvolvimento orientado a objetos, fornecendo uma abordagem sistemática para o uso da UML. Seu objetivo principal é atender às necessidades dos usuários, garantindo alta qualidade de software dentro de cronograma e orçamento previsíveis.

A atividade de teste de software é crucial para garantir a confiabilidade do produto final e reduzir os riscos de falhas no ambiente operacional, contribuindo para a qualidade dos sistemas. Os testes devem ser executados durante todo o processo de desenvolvimento para obter uma cobertura abrangente e fornecer informações relevantes para a tomada de decisões sobre o produto. Além disso, a atividade de teste não garante automaticamente a qualidade, mas ajuda a medir a qualidade do software, pois permite a identificação e correção de defeitos, garantindo que o software atenda

aos requisitos funcionais e não funcionais definidos.



## ATIVIDADE DE FIXAÇÃO

1. O que é o RUP (Rational Unified Process) e qual é o seu objetivo principal?
2. Quais são as quatro fases do RUP e o que é feito em cada uma delas?
3. Quais são os principais papéis envolvidos no teste de software?
4. Qual é a importância do teste de software no mercado atual?
5. Cite dois exemplos de falhas em software que causaram problemas financeiros ou de reputação para as empresas.
6. Explique a importância do planejamento na fase de Concepção/Iniciação do RUP
7. Descreva as atividades que são realizadas na fase de Elaboração do RUP.
8. Cite três exemplos de falhas em software que causaram problemas financeiros ou de reputação para as empresas.
9. Crie um cenário hipotético em que o TMMi seria útil para uma empresa que deseja melhorar a qualidade de seus testes.
10. Explique a diferença entre Controle da Qualidade e Garantia da Qualidade de Software e como o teste de software se encaixa em ambos os conceitos.

## TEMA 05

### Processos de revisões

#### Habilidades:

- Aplicar medidas corretivas em sistemas.;
- Compreender as melhores Práticas e quais são os usos e monitoramento no Desenvolvimento de sistemas;

#### Metodologia e Melhores Práticas De Desenvolvimento De Software



Disponível em: <[vectorjuice-https://tinyurl.com/yc85e9b3](https://tinyurl.com/yc85e9b3)>. Acesso em 27 ago. 2023.

Na era moderna, a tecnologia da informação e seus respectivos softwares, serviços e aplicativos desempenham um papel cada vez mais predominante no dia a dia da sociedade, e, para o desenvolvimento dos mesmos, utilizamos metodologia de desenvolvimento de software, que é um conjunto de métodos coordenados para se alcançar um objetivo, através de uma dinâmica interativa visando à qualidade e à produtividade dos projetos.

Para trabalhar com o desenvolvimento de software de forma mais adequada, na primavera de 2000 um grupo de líderes da comunidade do Extreme Programming (XP – metodologia que abordaremos mais à frente) se reuniram para discutir o processo de desenvolvimento com XP; no decorrer da reunião chegaram a um consenso sobre pontos importantes referentes ao desenvolvimento de software e decidiram escrever um documento que servirá como guia aos novos processos de desenvolvimento ágil de software, o famoso

**Material Complementar:** Entenda o que são os métodos ágeis em 5 minutos [Decifrando

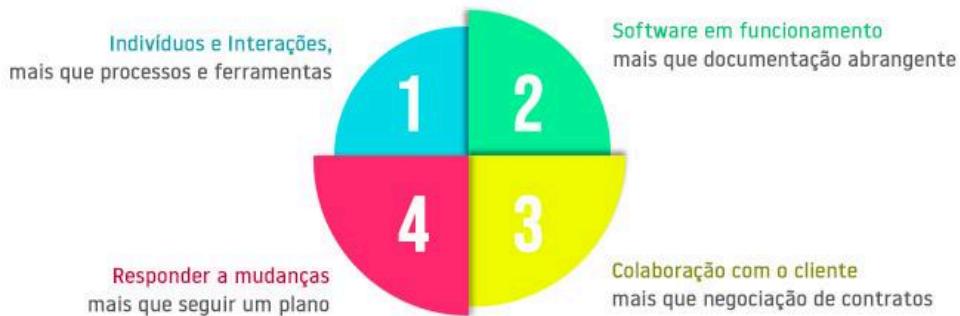
Agile 2]- Acompanhe o vídeo para saber mais detalhes sobre Métodos Ágeis e suas derivações e quais implementações e utilizações do dia a dia.



Fonte do vídeo: Entenda o que são os métodos ágeis em 5 minutos [Decifrando Agile 2]

## Manifesto Ágil

# MANIFESTO ÁGIL



Disponível em: <<https://tinyurl.com/2dx4ktxy>>. Acesso em 27 ago. 2023.

O Manifesto Ágil possui 4 valores e 12 princípios. Vejamos a seguir os valores, e note nas frases que mesmo havendo valores nos itens à direita, a valorização é maior nos itens da esquerda

## Valores

- Indivíduos e interações mais que processos e ferramentas: O desenvolvimento de software é uma atividade humana e que problemas de comunicação podem ser resolvidos com qualidade na interação entre os envolvidos. Os processos e ferramentas devem cumprir seu papel de forma simples e útil, mas sem “passar por cima” das pessoas.
- Software em funcionamento mais que documentação abrangente: O software funcionando é um indicador de sucesso é e o que o cliente espera, a documentação do mesmo deve ser algo que agregue valor e com informações necessárias.

- Colaboração com o cliente mais que negociação de contratos: A colaboração com o cliente é algo fundamental para o sucesso do desenvolvimento do software, o trabalho em equipe e a tomada de decisões em conjunto contribuem para alcançar um único objetivo.
- Responder a mudanças mais que seguir um plano: O desenvolvimento de software é algo complexo e possui alta incerteza, nessas situações devemos aprender com as situações ocorridas e adaptar o plano sempre que possível.

## Princípios

- Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adaptam a mudanças para que o cliente possa tirar vantagens competitivas
- Entregar frequentemente software funcionando, de poucas semanas a poucos meses, dando preferência à menor escala de tempo
- Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto
- Construir projetos em torno de indivíduos motivados, dando a eles o ambiente e o suporte necessário e confiando neles para fazer o trabalho
- O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face
- Software funcionando é a medida primária de progresso
- Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente • Contínua atenção à excelência técnica e bom design aumentam a agilidade
- Simplicidade: a arte de maximizar a quantidade de trabalho não realizado é essencial
- As melhores arquiteturas, requisitos e designs emergem de times auto- organizáveis
- Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo

Com o Manifesto Ágil escrito, foi criada a Agile Alliance, uma organização sem fins lucrativos que procura disseminar o conhecimento e promover discussões sobre os métodos ágeis, como o Scrum, Kanban e XP, que carregam em suas bases os valores e princípios do Manifesto Ágil. Muitas pessoas acham que as metodologias ágeis são “ágeis” porque são rápidas e entregam funcionalidades o quanto antes, e na verdade a agilidade simboliza a capacidade de se adaptar a mudanças, a velocidade nas entregas é uma consequência que acontece quando utilizamos adequadamente as metodologias que vamos abordar a seguir.

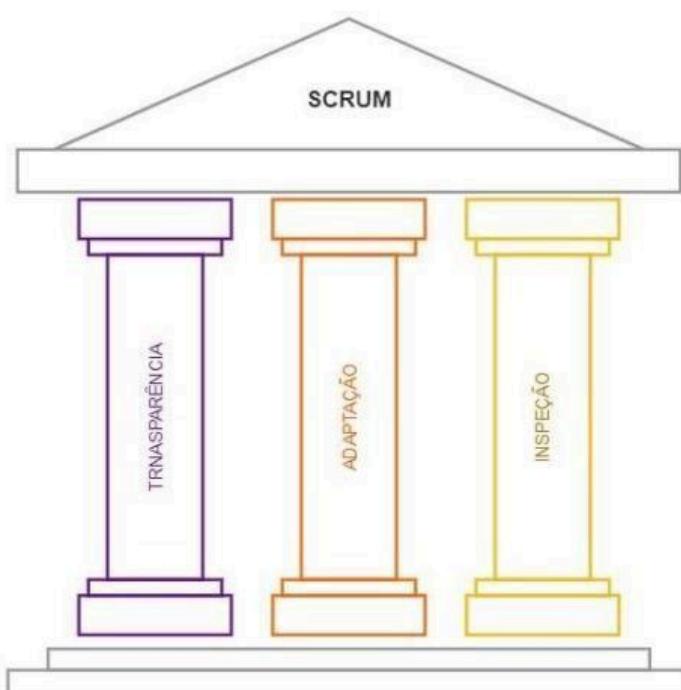
## Framework Scrum

O Scrum é um framework simples para gerenciar projetos complexos, através de uma

dinâmica, papéis e cerimônias no decorrer de todo o ciclo. O Scrum foi concebido na década de 1990, e o termo foi inspirado em uma jogada de rúgbi em que 8 jogadores se reúnem para retirar os obstáculos à frente do jogador que correrá com a bola, utilizando-se do próprio corpo como principal armadura para livrar o companheiro do time, para que assim ele consiga avançar o máximo possível e marcar mais pontos. Essa analogia descreve a importância das equipes e seu trabalho em conjunto para resolver os mais variados problemas.

O Scrum não restringe sua aplicação apenas para a área de TI, ele pode ser aplicado em projetos diversos que possuem complexidade. A abordagem incremental e iterativa é utilizada fortemente no Scrum, onde a cada determinado espaço de tempo, uma funcionalidade do produto que faz parte de um todo é entregue. Vamos detalhar agora as características desse framework, mas, antes, é necessário internalizar os pilares do Scrum!

## Pilares



Fonte: Elaborado pelo autor (2023).

**Transparência:** Todos possuem o conhecimento dos requisitos, processos e do andamento do projeto.

**Inspeção:** Durante todo o tempo é inspecionado o que está sendo feito, através de reuniões diárias ou no momento de revisão.

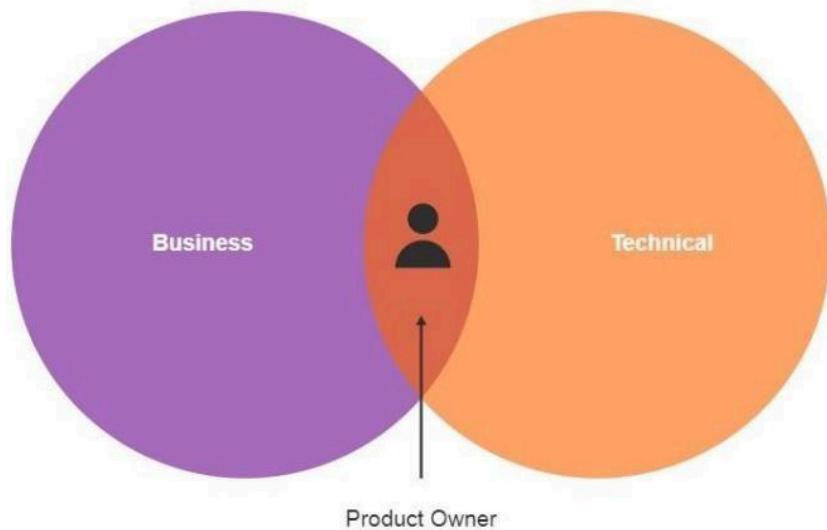
**Adaptação:** Conforme as mudanças vão acontecendo, tanto o processo quanto o produto podem sofrer adaptações, desde que preservados os valores e práticas ágeis

## Papeis Fundamentais

Para a realização do Scrum, são necessários 3 papéis fundamentais, sem esses papéis, não

temos Scrum:

**Product Owner (PO):** Como o nome já diz, é o dono do produto, o único responsável por decidir quais os recursos e funcionalidades o produto deve ter e qual a ordem de prioridade que deve ser desenvolvido. Também mantém e comunica uma visão clara do que o Time Scrum está trabalhando no produto, sendo um ponto de intersecção entre a área de negócios e a área de desenvolvimento. É muito importante que o PO saiba o que é o produto que quer desenvolver e tenha autonomia para tal, pois uma vez que ele depende de outros para tomar decisões ou tocar o seu produto, acaba virando um “PO Proxy” que não sabe o que tem que fazer, e só fica perguntando ou dependendo de terceiros para desenvolver o seu produto.



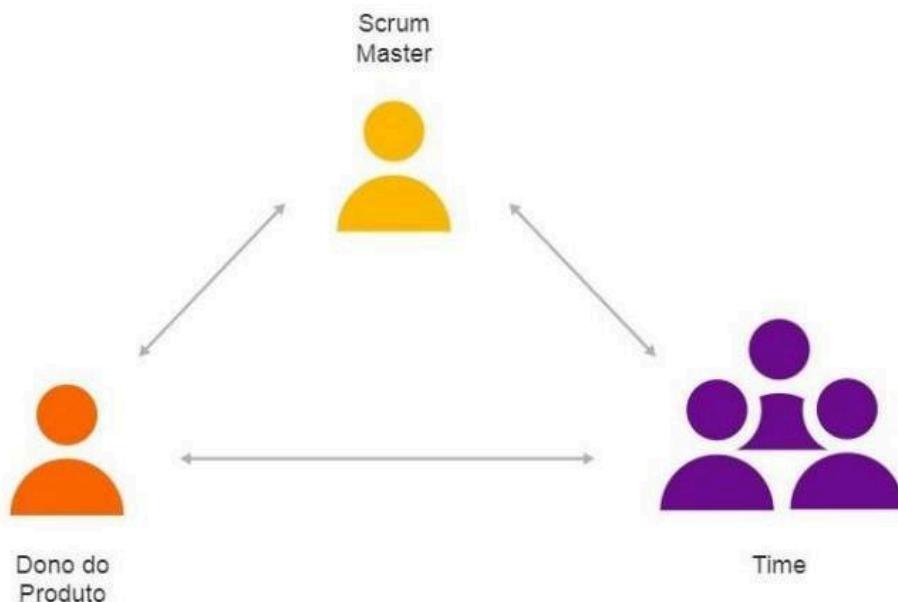
Fonte: Elaborado pelo autor (2023).

**Scrum Master (SM):** Responsável por ajudar os envolvidos a entender a cultura ágil com todos os seus valores e princípios, além de todas as práticas do framework Scrum. Ele atua como um líder do processo e ajuda os demais a desenvolver sua própria forma de trabalho com Scrum. O Scrum Master também é um facilitador, contribuindo na resolução de conflitos e remoção de impedimentos, protegendo o time contra interferências externas que possam acontecer. Existe uma linha tênue que o Scrum Master deve se atentar e que é muito comum no mercado, a confusão do Scrum Master com um chefe, que de fato ele não deve ser, e assim como o PO, ele faz parte do time. O outro ponto de atenção é referente a ser um facilitador e não um bloqueador, muitos Scrum Masters acabando indiretamente se tornando um bloqueador ao proteger demais os seus times, o Scrum Master deve procurar o desenvolvimento do time para que os mesmos sejam independentes.



Fonte: Elaborado pelo autor (2023).

**Time Scrum:** Time de desenvolvimento que possui vários membros com diferentes funções, que possuem as habilidades necessárias para o desenvolvimento do software, é o Time Scrum que decide como fazer para alcançar o objetivo do desenvolvimento. Normalmente o time Scrum possui entre 4 e 8 pessoas, em que os mesmos colaboram entre si para se tornarem um time auto-organizado e multidisciplinar. Quando são necessárias equipes maiores para o desenvolvimento de um projeto, em vez de montar uma única equipe Scrum com, por exemplo, 50 pessoas, o ideal seria montar vários times Scrum menores.



Fonte: Elaborado pelo autor (2023).

## Valores

Assim como no Manifesto Ágil, o Scrum também possui seus próprios valores, que dão personalidades aos papéis:

**Foco:** Todos estão focados em um único objetivo, em vez de cada um trabalhar em diversas partes isoladas para entregar tudo junto no final. O time deve manter o foco em pequenas partes por vez, essas partes devem estar alinhadas com a priorização do Product Owner, em que o mesmo estará focado nas necessidades mais prioritárias do seu produto. Para manter o foco entre ambas as partes, o Scrum Master facilita o meio de campo e ajuda na remoção dos impedimentos.

**Coragem:** O time precisa ter coragem para enfrentar os desafios mais diversos, independentemente da dificuldade, e sempre fazer as coisas do modo mais correto possível. No decorrer do tempo, imprevistos acontecem e a coragem envolve aceitar as mudanças, errar, corrigir o erro e aprender com o mesmo o mais rápido possível.

**Comprometimento:** Todos os papéis possuem seu comprometimento, o Time Scrum em realizar o que foi acordado, o Scrum Master em “girar a roda” do framework Scrum junto ao time, e o Product Owner em priorizar as necessidades de negócio e tocar o produto.

**Respeito:** Os membros dos times respeitam um ao outro e se sentem confortáveis para interagir, ouvir e respeitar as opiniões, reconhecendo e entendendo as diferenças entre as diversas personalidades dentro do grupo, promovendo assim um ambiente saudável para o trabalho. O Product Owner respeita as decisões técnicas do Time Scrum, e em contrapartida o Time Scrum respeita as decisões de negócio vindas pelo Product Owner.

**Abertura:** Todos do time estão abertos aos trabalhos e desafios que estão por vir, também envolve a abertura de oferecer e receber feedbacks, dialogar sobre situações e criar visibilidade dos problemas acontecidos, além da clareza do trabalho realizado e da performance do mesmo

## Princípios

Os princípios do Scrum completam os valores e são fundamentais para o bom funcionamento do framework, eles são:

- Controle dos processos empíricos: As decisões no Scrum são tomadas baseadas na observação e experimentos, através da transparência, inspeção e adaptação no decorrer dos ciclos.
- Auto-organização: Os times são auto-organizáveis e possuem a responsabilidade compartilhada, conseguem atuar ponta a ponta no desenvolvimento do produto, entregando assim um maior valor.
- Colaboração: O trabalho colaborativo defende um processo de criação de valor compartilhado, com times trabalhando em conjunto para atingirem melhores resultados.
- Priorização baseada em valor: A priorização feita pelo Product Owner considera valor, risco, incerteza e dependências, resultando em entregas que fornecem o maior valor de negócio em menor tempo possível.
- Time-boxing: O conceito do time-boxing propõe a fixação de um determinado período de tempo para cada processo e atividade no ciclo do Scrum, garantindo assim que o time

utilize da melhor maneira o seu tempo, e não desperdice tempo e energia em um trabalho no qual tenha pouco conhecimento ou não promova tanto retorno.

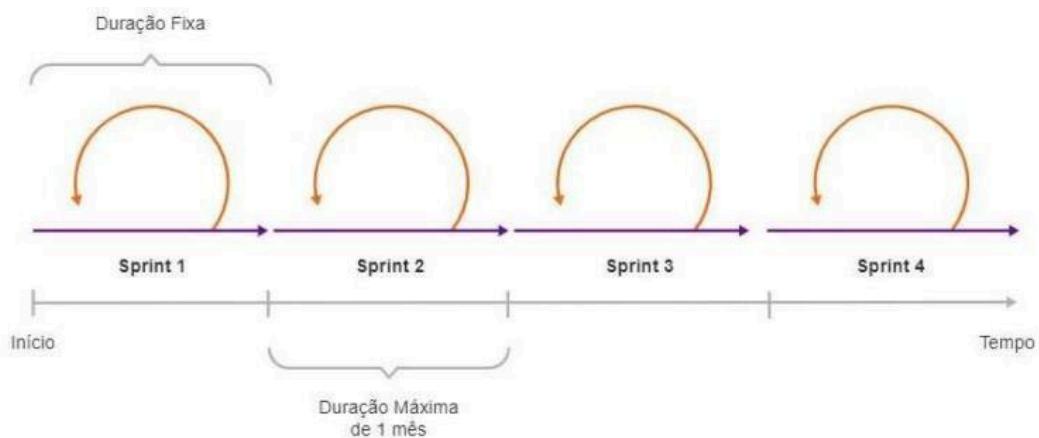
- Desenvolvimento iterativo: O processo iterativo permite que os times aprendam com o decorrer dos ciclos, promovendo refinamento a cada iteração, devido ao modelo de entrega de forma incremental

## Dinâmicas, cerimônias e artefatos

No Scrum tudo começa com a visão de um produto, em que o Product Owner descreve o que ele quer do produto e onde quer chegar, e, dada a visão macro, o Product Owner lista as funcionalidades necessárias, essa lista é o artefato Product Backlog.

Com o Backlog em mãos, o Product Owner realiza a priorização das atividades, seguindo o princípio da priorização baseada em valor. Cada funcionalidade do Backlog é definida como um artefato de User Stories (história de usuário), que possui como objetivo explicar a funcionalidade de forma simples, deixando claro quem é o usuário, o que ele quer e para qual objetivo.

Para a execução dos User Stories que estão no Backlog, no Scrum utilizamos o princípio de desenvolvimento iterativo através da Sprint, período de tempo limitado (time-boxing) de duração fixa (normalmente de 2 a 4 semanas) que cria algo de valor tangível para o cliente ou usuário do produto.



Fonte: Elaborado pelo autor (2023).

Antes de cada Sprint começar é realizada uma cerimônia chamada de Sprint Planning, para determinar qual o objetivo da Sprint e quais User Stories serão desenvolvidas naquela Sprint. Durante o planejamento, o Product Owner, Scrum Master e Time Scrum precisam entrar em um consenso referente ao que cabe dentro da Sprint baseando-se na capacidade e velocidade da equipe. Caso existam dúvidas referentes aos tamanhos das atividades, utilizamos o Planning Poker para auxiliar na decisão, na qual, dado o escopo, os integrantes realizam votações com cartas para expor sua opinião referente ao tamanho daquela User Story.



Disponível em: <[storyset-https://tinyurl.com/yr7p27we](https://tinyurl.com/yr7p27we)>. Acesso em 27 ago. 2023.

Com o escopo do Sprint definido, o Sprint Planning é finalizado e, em seguida, o Time Scrum se reúne novamente para discutir sobre as atividades e ter uma visão mais detalhada do que precisa ser feito no Sprint que será iniciado. Caso o time perceba que não poderá cumprir com o esperado no Sprint, uma negociação com o Product Owner será realizada para ajustes do escopo. É responsabilidade do Time Scrum determinar o quanto ele é capaz de se comprometer a entregar.

Perceba que houve duas fases de planejamento, a primeira fase também é chamada de Planning Estratégico que define o que será feito, e a segunda fase de Planning Tático que define como será feito.

Com o Sprint definido, o desenvolvimento das atividades é iniciado, e todos os dias é realizada uma reunião de alinhamento rápido, essa cerimônia é chamada de Daily Scrum. No Daily, três perguntas fundamentais devem ser respondidas:

- O que eu fiz ontem que contribuiu para o atingimento da meta do Sprint?
- O que eu farei hoje para contribuir para o atingimento da meta do Sprint?
- Existe alguma circunstância que impeça a mim ou ao time atingir a meta do Sprint?

O Daily deve durar no máximo 15 minutos e acontecer idealmente no mesmo horário, e todos os integrantes do Time Scrum devem comunicar suas atividades baseadas nas perguntas, o Product Owner pode participar da daily opcionalmente. É recomendado que o Daily seja realizado com todos de pé, com a intenção de fazer com que a reunião seja rápida, essa prática é chamada de Stand-Up Meeting. Esse acompanhamento diário promove a transparência e o alinhamento constante entre o time e a área de negócio, todos conseguem visualizar como está o andamento do Sprint com relação ao objetivo acordado.

Durante o período do Sprint, para saber se a atividade pode ser considerada concluída, utilizamos o conceito de Definition of Done (DoD), que é um acordo formal do time que define claramente quais são os passos mínimos para a conclusão de um item potencialmente entregável.

Perto de finalizar o Sprint, é realizada uma cerimônia para apresentar o que foi feito, o Sprint Review, com o intuito de verificar e adaptar o produto que está sendo desenvolvido. Por ser uma cerimônia mais para o final do Sprint, o ideal é que seja apresentado, durante o Sprint, o que está

sendo desenvolvido ao Product Owner de forma mais informal, para se necessário realizar alguma adaptação se possível dentro do próprio Sprint.

Para finalizar, a cerimônia de Sprint Retrospective é realizada, enquanto o objetivo do Sprint Review é verificar necessidades de adaptações e melhorias no produto, o Sprint Retrospective tem como objetivo verificar necessidades de adaptações e melhoria no processo de trabalho como um todo. É importante que na reunião de retrospectiva sejam levantados os pontos positivos e a melhora (sempre focando nas causas raiz) da Sprint que passou.



Fonte: Elaborado pelo autor (2023).

## Extreme Programming

Extreme Programming ou XP é uma metodologia de desenvolvimento de software leve e não prescritivo, criado em 1997, que procura levar ao extremo as boas práticas de engenharia de software, com foco em agilidade e qualidade, e assim como o Scrum e Kanban, se apoia em valores e práticas.

### Valores

**Comunicação:** O XP é organizado em práticas que não podem ocorrer sem a comunicação, como, por exemplo, a programação em pares, em que os desenvolvedores estão juntos programando em um único computador, trocando informações e se comunicando constantemente.

**Simplicidade:** Muitas vezes ao iniciarmos o desenvolvimento de uma demanda, acabamos por fazer algo mais complexo do que o necessário, o XP preza por fazer o mais simples possível no agora e, caso seja necessário, algo mais complexo no amanhã.

**Feedback:** O XP procura alcançar o feedback da forma mais rápida possível, através de testes automatizados, conseguimos respostas imediatas se aquilo que foi implementado ou alterado está funcionando.

**Coragem:** Para trabalhar com o XP, coragem é necessária para dar e receber feedback, fazer o que precisa ser feito, descartar o código ruim e protótipos criados que não devem virar produtos, aprender com os erros e acreditar na capacidade de reagir a mudanças.

**Respeito:** Os membros da equipe precisam respeitar uns aos outros e seus respectivos pontos de vista, saber ouvir e falar contribuirá com a comunicação e com o sucesso do projeto



Fonte: Elaborado pelo autor (2023).

## Práticas

Para aplicar o XP, existe um conjunto de boas práticas que o método sugere:

- Propriedade Coletiva (Collective Ownership): O código-fonte não tem dono, todos da equipe têm permissão para modificá-lo, contribuindo para disseminar o conhecimento de diversas partes do sistema.
- Padronização de Código (Coding Standards): O time de desenvolvimento define padrões e regras de programação que todos devem seguir, para que assim o código mantenha uma padronização independentemente da quantidade de pessoas que adiciona ou altera códigos.
- Versões Pequenas (Small Releases): O projeto é liberado em versões funcionais pequenas, auxiliando no processo de aceitação do cliente.
- Planejando o jogo (Planning Game): O desenvolvimento das interações entre cliente e desenvolvedores é realizado para priorizar as funcionalidades através de uma reunião, chamada de Jogo do Planejamento, e segue a mesma ideia do Planning do Scrum, porém acontece semanalmente.
- Testes de aceitação (Customer Tests): O cliente em conjunto com o time cria testes de aceitação referente a um requisito do sistema.

- Programação Pareada (Pair Programming): A programação é realizada em dupla e em um único computador, em que a dupla deve ser composta se possível por um profissional iniciante e outro mais experiente. Assim o código é sempre revisto por duas pessoas, além de disseminar o conhecimento entre o mais experiente e o iniciante.
- Desenvolvimento orientado a testes (Test Driven Development): É uma abordagem complexa devido ao fato de ser contrária ao padrão comum seguido nos times de desenvolvimento. Se costumeiramente se cria o código da funcionalidade para depois desenvolver os testes unitários, em TDD (e, portanto, no XP) é o contrário, primeiro cria-se os testes unitários para depois criar o código da funcionalidade que fará esses testes passarem, obrigando assim que a qualidade do código seja mantida.
- Integração contínua (Continuous Integration): Sempre integrar a funcionalidade que está sendo desenvolvida com a funcionalidade atual, para evitar o descobrimento de problemas apenas no final do ciclo.
- Design Simples (Simple Design): Manter o código simples é um dos princípios do XP, o que não quer dizer código fácil, a simplicidade do código contribui para o entendimento do mesmo e com o foco do que foi solicitado na funcionalidade.
- Refatoração (Refactoring): Processo de melhoria contínua de programação, com o objetivo de melhorar a clareza do código e permitir maior reaproveitamento do mesmo.
- Metáforas (Metaphor): Facilitar a comunicação entre o cliente e os membros do time através de metáforas, nem sempre o cliente entenderá o mundo sistêmico então devemos traduzir as necessidades que ele espera para dentro do projeto.
- Semana de 40 horas (Sustainable Pace): O XP acredita que o trabalho com qualidade deve ser feito dentro de 40 horas semanais, 8 horas por dia e sempre evitando horas extras para que não ocorra sobrecarga dos profissionais. O ambiente de trabalho deve possuir condições favoráveis e o clima do time deve ser motivado.



Os processos de revisões desempenham um papel crucial no desenvolvimento de sistemas, visando garantir a qualidade e a eficiência do software produzido. Essas revisões envolvem a análise minuciosa de códigos, documentação e outras etapas do ciclo de desenvolvimento, com o intuito de identificar erros, inconsistências e oportunidades de melhorias. Além disso, esses processos são fundamentais para a detecção precoce de falhas, o que contribui para a redução de custos e retrabalhos no longo prazo.

A aplicação de medidas corretivas é uma parte essencial dos processos de revisões. Uma vez identificadas as falhas, é importante adotar ações corretivas eficazes para solucioná-las. Isso envolve a modificação do código, ajustes na arquitetura do sistema ou até mesmo revisões nos procedimentos de desenvolvimento adotados. Dessa forma, as revisões não apenas identificam problemas, mas também promovem a evolução contínua do projeto, fortalecendo sua qualidade e confiabilidade.

Conhecer as melhores práticas no contexto das revisões é fundamental para o sucesso do desenvolvimento de sistemas. Isso implica compreender os padrões de codificação, as diretrizes de documentação e os critérios de avaliação de desempenho. Além disso, compreender quando e como realizar revisões é crucial. Estabelecer uma estratégia de revisão bem definida permite que os desenvolvedores otimizem seu tempo, concentrando-se nas áreas mais críticas e relevantes do sistema.

O uso adequado e o monitoramento contínuo dos processos de revisões são aspectos que não podem ser negligenciados. A integração de ferramentas automatizadas de análise de código, por exemplo, pode agilizar e aprofundar as revisões, identificando problemas que podem passar despercebidos a olho nu. Além disso, a definição de métricas e indicadores de desempenho auxilia no acompanhamento da eficácia das revisões ao longo do tempo, possibilitando ajustes e aprimoramentos constantes.

Em suma, os processos de revisões desempenham um papel fundamental no desenvolvimento de sistemas de qualidade. A habilidade de aplicar medidas corretivas de forma adequada, aliada ao conhecimento das melhores práticas e à capacidade de uso e monitoramento eficaz desses processos, são elementos-chave para garantir a excelência no resultado final do projeto. Ao investir na implementação sólida dessas práticas, as equipes de desenvolvimento podem otimizar seus recursos, mitigar riscos e entregar sistemas mais confiáveis e alinhados com as expectativas.



## ATIVIDADE DE FIXAÇÃO

**1.** O que é uma metodologia de desenvolvimento de software?

- a) Um conjunto de métodos coordenados para alcançar um objetivo.
- b) Um documento que serve como guia para processos ágeis.
- c) Um conjunto de ferramentas para programação.
- d) Um grupo de líderes da comunidade de XP.

**2.** Quais são os valores do Manifesto Ágil?

- a) Processos e ferramentas, documentação abrangente, negociação de contratos e seguir um plano..
- b) Indivíduos e iterações, software em funcionamento, colaboração com o cliente e responder a mudanças.
- c) Foco, simplicidade, feedback e coragem.
- d) Comunicação, planejamento tático, testes de aceitação e design simples.

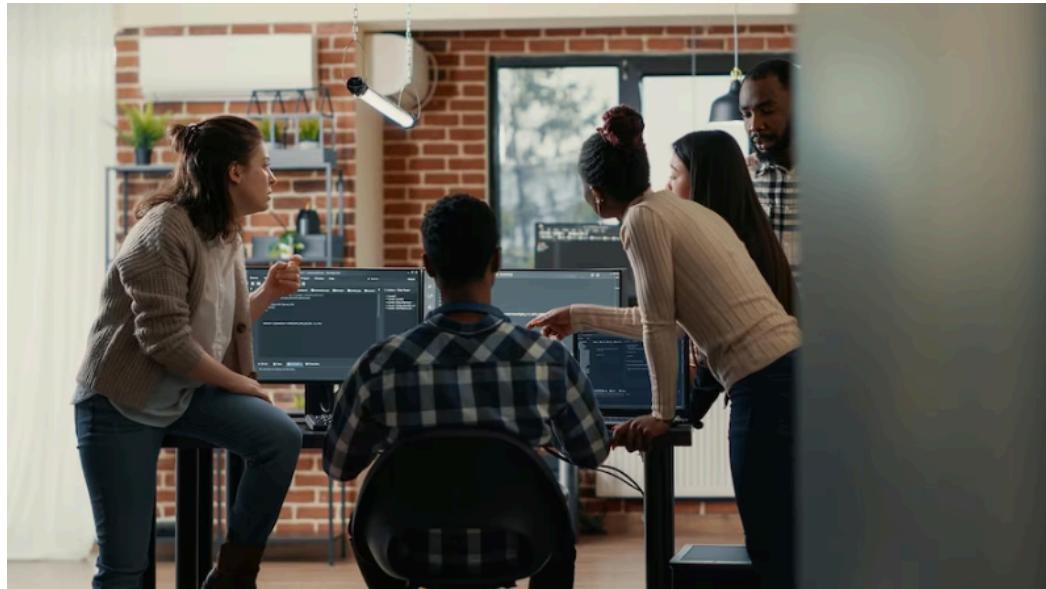
- 3.** Qual o principal objetivo do Product Owner (PO) no Scrum?
- a) Programar as funcionalidades do produto.
  - b) Criar o Product Backlog e definir o cronograma.
  - c) Priorizar as necessidades de negócio e tocar o produto.
  - d) Ser responsável por definir a ordem de prioridade das tarefas do Time Scrum.
- 4.** Qual é o princípio fundamental do Scrum relacionado ao tempo de trabalho?
- a) Foco nas funcionalidades de maior valor.
  - b) Desenvolvimento iterativo e incremental.
  - c) Controle dos processos empíricos
  - d) Semana de 40 horas, evitando horas extras.
- 5.** Cite três práticas do Extreme Programming (XP) e explique a importância de cada uma delas no desenvolvimento de software.
- 6.** Durante a Sprint, o que é a reunião Daily Scrum e qual o seu propósito? Quais perguntas são respondidas durante essa reunião?
- 7.** Explique o conceito de desenvolvimento iterativo e incremental e como ele é aplicado tanto no Scrum quanto no Extreme Programming (XP).
- 8.** Com base nos valores e princípios do Manifesto Ágil, explique como a agilidade é alcançada no desenvolvimento de software.
- 9.** Compare as metodologias Scrum e Extreme Programming (XP) em termos de valores, princípios e práticas. Identifique as semelhanças e diferenças entre elas, destacando as vantagens de cada abordagem para o desenvolvimento de software ágil.
- 10.** Explique sobre o processo de desenvolvimento iterativo no Scrum e explique como ele promove a melhoria contínua.

## TEMA 06

### Ambientes de Testes

#### Habilidades:

- Utilizar softwares de apoio ao teste de sistemas.;
- Compreender a definição e padronização da documentação de teste;



Disponível em: <DCStudio-><https://tinyurl.com/3wdunyeu>

#### Documentação de teste

A documentação de teste normalmente estará definida no processo de teste. Esses documentos deverão estar com os respectivos templates definidos também no âmbito da organização.

Caso seja necessário algum documento adicional, que já não esteja padronizado dentro da organização, o seu uso deverá ser definido neste item. Caso a documentação padrão atenda, então bastará uma pequena referência.

A norma IEEE-829 define quais são os documentos necessários para um projeto de teste de software e deve ser levada em consideração no processo de teste da empresa. Ou seja, o processo de teste deve se usar a norma como insumo.

#### Requisitos de administração de teste

Caso haja alguma atividade excepcional não prevista nos processos de teste e que tenha que ser cumprida, isso deve ser relatado detalhadamente.

## Requisitos de relatórios de teste

Os mesmos conceitos usados para a documentação de teste devem ser seguidos para os relatórios de teste, ou seja, os relatórios compõem o conjunto de documentos produzidos pelo projeto. Isto é, o processo de teste deve definir os relatórios a serem gerados.

## Glossário

Preparar um glossário dos principais termos usados no projeto de teste de software. O ideal seria que apenas um glossário fosse usado, ao invés de termos diversos glossários espalhados pelos planos de teste.

Desta forma o Plano Máster de Teste concentraria todos os glossários e teríamos um único glossário de todo o projeto.

Procedimentos de alterações do documento e histórico de alterações

Este item do plano serve para registrar as alterações feitas no plano e quem foi o responsável pela sua atualização.

Além disso, deverá ser definido em que condições o plano poderá ser alterado e quem terá a responsabilidade pela sua alteração e quais são os responsáveis pela aprovação das alterações. Caso exista um Plano de Configuração ele deverá estar contemplado no PMT, e, caso não exista, procedimentos de segurança devem ser definidos para garantir a manutenção das diversas versões do documento.

É importante lembrar que cada alteração efetuada implicará no registro do responsável pela alteração e de quem aprovou a mesma. Isso poderá implicar diversos níveis de aprovação ou de alteração.

## Plano de teste

Dependendo do tamanho do projeto de desenvolvimento poderemos ter diversos planos de teste para um mesmo projeto de teste de software. Esses planos podem ser separados por módulos, funcionalidades ou por requisitos, ou por um grupo de cada um desses elementos. Além disso, os planos podem também ser classificados por nível de teste, qual seja unitário, integração, sistema e aceitação. Quando o sistema for pequeno não há necessidade da criação de vários planos de teste, embora, mesmo nesses casos, pode ser razoável termos os planos de teste separados por nível de teste.

No entanto, o que a IEEE-829 apregoa, é que o ideal seria termos no mínimo um PMT e um PT (Plano de Teste).

O Plano de Teste deve ter os seguintes campos:

### 1. Introdução

- Identificador do Plano de Teste;
- Escopo;
- Referências

- Nível na sequência de teste
- Classe de teste e visão das condições de teste

## 2. Detalhes para este nível do plano de teste

- Itens de teste e seus identificadores;
- Matriz de rastreabilidade do teste;
- Funcionalidades a serem testadas;
- Funcionalidades que não serão testadas;
- Abordagem do teste;
- Critérios de liberação/falha dos itens;
- Requisitos de suspensão e retomada;
- Entregas do teste;

## 3. Gerência de Teste

- Tarefas do teste;
- Necessidades de ambientes;
- Responsabilidades;
- Integração entre as partes envolvidas;
- Recursos e sua alocação;
- Treinamento;
- Cronograma, estimativas e custos;
- Riscos e contingências;

## 4. Geral

- Procedimentos de garantia de qualidade;
- Métricas;
- Cobertura do teste;
- Glossário;
- Procedimentos de alteração do documento e histórico.

## Identificador do Plano de Teste

Esse campo deve especificar um identificador único para reconhecimento do Plano de Teste. Pode ser inclusive um código que serve para identificar o arquivo Plano de Teste sob um software de gerência de configuração.

As empresas normalmente possuem regras e padrões para identificar os seus artefatos de forma a ligá-los aos projetos de software em andamento. Ou seja, ao olhar o identificador é importante que o código diga que aquele arquivo é um Plano de Teste do projeto de teste do software "xx".

No caso de existir mais de um Plano de Teste, para um mesmo projeto de desenvolvimento, a regra de codificação deve contemplar também essa possibilidade. No entanto, sempre que se pensar em elaborar mais de um Plano de Teste para um mesmo projeto de desenvolvimento/teste, é aconselhável usar um PMT que seja o plano raiz, e depois quebrar em diversos planos de baixo nível classificados da forma que melhor se adapte à necessidade.

Deve também ser lembrado que o identificador deve contemplar as diversas versões do Plano de Teste dentro da gerência de configuração, local onde deverão estar armazenados todos os artefatos do projeto de teste.

## Nível na sequência de teste

Deve ser identificado em que nível da sequência de teste este plano está situado. Isto pode ser melhor representado por um diagrama ou fluxo. Por exemplo, podemos estar executando no plano de teste de aceitação, ou o plano de teste de sistema de um determinado grupo de funcionalidades.

## Classe de teste e visão das condições de teste

Os níveis de teste devem estar de acordo com o escopo definido para o projeto que este plano de teste reflete. Por exemplo, se estamos no nível de aceitação (plano de teste de aceitação), entendemos que o foco deverá ser atender aos requisitos. No caso das classes podemos considerar algumas técnicas de teste que deverão ser usadas, tais como, valores limites, teste positivo e negativo, teste exploratório, etc.

## Detalhes para este nível do plano de teste Itens de teste e seus identificadores

Dentro do escopo do projeto de teste, devem-se listar os itens a serem testados ou usados nos testes: procedimentos, classes, módulos, lista de requisitos, bibliotecas, componentes, subsistemas, programas, etc., especificando a forma como esses itens serão disponibilizados para o teste. Nesta lista deve ser separado o que será testado e o que não será testado. Entende-se como revisão ou inspeção técnica, trabalhos incluídos na lista das atividades pertinentes ao processo de teste de software.

Procedimentos específicos deverão ser executados se os itens de teste estiverem sob o controle de um software de gerência de configuração. Neste caso as respectivas versões dos artefatos devem ser identificadas.

Devem-se incluir referências aos documentos onde os itens estão descritos (requisitos, modelos, manuais, etc.)

O padrão define como Itens de Teste todo artefato que será passado para a equipe de teste e que deverá ser usado no seu trabalho de teste.

Outras referências podem ser a autorização do projeto, o plano de garantia de qualidade e o plano de gerência de configuração.

Exemplo: Lista de requisitos; Plano do Projeto; Casos de Uso; etc. A lista de requisitos serve para o caso de inspeção ou revisão e também para a preparação dos casos de teste. O Plano do Projeto pode ser revisto dentro da visão dos testadores e serve também como documento base para a elaboração do Plano de Teste. No caso dos Casos de Uso, esses são essenciais para a elaboração dos casos de teste. Lembre-se que não estamos abrindo o conteúdo de cada um desses artefatos, mas

apenas listando por nome ou identificação.

Qualquer problema com os itens de teste deve ser relatado no relatório de anomalias.

### Matriz de rastreabilidade de teste

Na maioria dos casos os projetos de desenvolvimento já contemplaram uma matriz de rastreabilidade com os seus artefatos. Neste caso seria necessário incluir nessa matriz os artefatos de teste tais como procedimentos de teste, casos de teste e scripts de teste. Caso o projeto de desenvolvimento não tenha uma matriz será necessário criar uma específica para o projeto de teste começando pelos requisitos do projeto.

Exemplo parcial de uma matriz de rastreabilidade:

Requisitos	Casos de uso	Programas	Casos de teste
Requisito 1	Caso de uso 1	Programa 1	Caso de teste 1
			Caso de teste 2

Fonte: Elaborado pelo autor (2023).

### Funcionalidades a serem testadas (O quê?)

O IEEE, no seu manual, lista como exemplos de funcionalidades: Conversão do banco de dados, Geração periódica de relatórios, Segurança, Recuperação, Performance. Por exemplo, caso o requisito diga que o software deve manter um tempo de resposta para o acesso a cada uma das suas páginas, isso pode ser um indicador de que algum tipo de teste específico, no caso teste de desempenho, pode vir a ser necessário.

De qualquer forma se aconselha levantar as funcionalidades a serem testadas sempre do ponto de vista do usuário, ou seja, usando a sua visão. Para facilitar pode-se definir que os itens de teste são a visão técnica composta por toda a documentação produzida pelo projeto e funcionalidades a serem testadas a visão do usuário, expressa pelos requisitos do software.

Neste item nós listamos as funcionalidades e requisitos do software, mas não especificamos ainda que tipos de teste serão executados.

Podemos considerar o Plano Master de Teste como o guarda-chuva que contempla todo o projeto de desenvolvimento. Neste caso, cada plano de teste deve especificar quais funcionalidades

estarão cobertas pelo seu escopo e quais não estarão cobertas.

## Abordagem do teste

A abordagem do teste deve levar em consideração o nível do Plano de Teste, qual seja, unitário, integração, sistema ou aceitação. Lembre-se que o plano de teste de sistema poderá também estar quebrado em diversos planos de teste classificados por exemplo, por módulos, funcionalidades ou requisitos.

A abordagem deverá ser suficientemente detalhada para permitir identificar as principais tarefas e o tempo estimado para executar cada uma delas.

A norma sugere o uso de um documento chamado matriz de requisitos de teste na qual deverão estar listados as funcionalidades a serem testadas, as funcionalidades que não serão testadas combinados com a abordagem de teste. Cada funcionalidade ou requisito deverá ter uma identificação de tipo e origem ou fonte. Por tipo entendem-se requisitos de software, código, projeto, teste, artefatos do projeto em geral, etc. Por fonte entende-se o local onde foi capturado o requisito. Se o requisito for extraído de um texto em um documento deverá ser identificado o documento, página, parágrafo, etc.

Para cada elemento da matriz de requisitos de teste deve ser colocado de forma genérica o método de teste que será usado para testá-lo.

Por métodos de teste entendemos o seguinte:

- Caixa preta - tipo de teste baseado nas funcionalidades.
  - Caixa branca - tipo de teste onde a estrutura interna dos programas é avaliada, por exemplo, verificar se todas as linhas de código foram executadas.
  - Análise - situações onde algum arquivo ou outro tipo de saída deverá ser analisado para verificar se os testes foram executados corretamente. A análise poderá ser feita através da geração de relatórios, arquivos, bancos de dados, etc.
  - Inspeção - tipo de teste estático, onde a documentação e o código são avaliados.
- A matriz de teste deverá ser combinada com a matriz de rastreabilidade ou se for possível formar uma única matriz.

## Critérios de liberação / falha dos itens

Definir os critérios para liberação dos artefatos testados e também se algum nível de falha pode ser aceito.

Um dos critérios de liberação pode ser a execução de todos os casos de teste, sem que nenhum defeito ou anomalia<sup>71</sup> de alta severidade esteja ainda pendente de acerto. Outro critério seria pelo nível de cobertura do código, ou seja, que percentual do código do programa deve ser coberto pelos testes.

Alguns aspectos importantes também precisam ser considerados, pois uma incidência de inúmeros defeitos de alta severidade, acima do padrão normal da empresa, medido por indicadores históricos, pode ser um sinal de que os testes precisam ser feitos com mais cuidado do que o habitual (veja critério de suspensão). Nesses casos pode haver inclusive uma suspensão dos testes.

Outra visão da cobertura dos testes pode ser definida pela Regra de Pareto. Segundo essa

regra, cerca de 20% do software é composto de 80% das regras de negócio mais importantes. Desta forma, na falta de recursos, testar esses 20% do software, que cobrem as regras de negócio mais importantes, é uma garantia que alguns riscos serão minimizados.

Algumas outras questões precisam também ser consideradas:

- Definir as expectativas quanto à completude do teste, tais como, grau de cobertura para alguns tipos de teste;
- Considerar, caso existam, as restrições do teste, tais como, restrições orçamentárias (lembre-se que é por funcionalidade) - veja a abordagem do teste;
- Definir os critérios de interrupção dos testes, ou seja, quando e como os testes serão dados como terminados (se o critério for intervalo entre falhas, isso deve ser especificado).

Exemplo: O teste pode ser dado como terminado quando todos os casos de teste tiverem sido executados e nenhum defeito de severidade grave estiver pendente de correção. Caso não seja possível atingir esse nível, deve ser definida, junto com os usuários e desenvolvedores, uma prioridade de teste por funcionalidade, escolhendo-se, primeiro, aquelas mais importantes para o negócio. Nesse caso devem ser considerados os níveis de integridade.

## Requisitos de suspensão e retomada

Nesse item devem ser especificados os critérios usados para suspender todas ou parte das atividades de teste associadas ao Plano de Teste. Especificar que atividades de teste devem ser repetidas no caso de retomada dos testes.

Um critério importante é o número de defeitos ou anomalias encontrados no teste. Se os defeitos ou anomalias encontradas atingirem um nível considerado muito alto (conforme indicadores históricos) pode ser definido que o software tem sérios problemas e que está sendo um desperdício de recursos testá-lo.

Por outro lado, deve também ser estabelecido o momento de suspender os testes. A existência de nenhum defeito aberto ou de nenhum defeito com alto grau de severidade pode ser um critério de suspensão dos testes. Lembre-se que, muitas vezes, existem limitações de prazo ou de orçamento que impedem os testes de continuarem indefinidamente (ver requisitos de liberação).

Exemplo: Os testes poderão ser interrompidos quando um determinado caso de uso (representado por um conjunto de casos de teste) apresenta uma incidência de anomalias acima no normal permitido, justificando, então, uma revisão dos códigos da aplicação, que deverá ser feita pelos desenvolvedores antes de liberar novamente para os testes. Devem ser consultados os indicadores de número de defeitos esperados por caso de uso, conforme o seu grau de complexidade.

## Entregas do teste

Identificar os documentos a serem entregues pelo projeto de teste. Os seguintes documentos deveriam ser incluídos na lista:

- Plano Máster de Teste;
- Plano de Teste (nos seus diversos níveis);

- Projeto de teste;
- Casos de teste;
- Procedimentos de teste;
- Relatórios (listar todos);
- Dados de teste usados nesse projeto;
- Programa e componentes;
- Ferramentas de teste;

O PMI estabelece que o plano de um projeto deve contemplar uma gerência de comunicação ou plano de comunicação. Neste plano, dentre outras coisas, devem ser listados todos os artefatos produzidos pelo projeto e para quem serão os mesmos encaminhados no decorrer do projeto, assim como o nome dos responsáveis pela sua execução e elaboração. Normalmente, a EAP – Estrutura Analítica do Projeto lista as atividades do projeto com os produtos associados (exemplo: atividade - Criar plano de teste; produto: Plano de Teste). Pode ser que este documento possa ser adequado para servir de suporte para o plano de comunicação e para o cronograma do projeto. Deve ser tomado um cuidado grande para não termos as mesmas informações em mais de um artefato, pois isso é bastante perigoso na evolução do projeto.

Exemplo: Deverá ser entregue, pela equipe de teste, uma lista de artefatos (ver relação anterior) indicando quem irá receber cada um e o prazo estabelecido para a entrega.

As datas para as reuniões de progresso do projeto, e quem deverá estar presente, como já vimos, constam do plano de comunicação do projeto e estão descritas em outro item.

A lista de artefatos a serem entregues deverá ser escolhida de acordo com o tipo do projeto de teste e os compromissos firmados com as partes envolvidas (stakeholders).

## Gerência de teste

Atividades e tarefas planejadas e a progressão do teste de todas as tarefas necessárias para preparar e executar os testes, identificando, se possível, a inter-relação entre elas e as habilidades necessárias para a sua execução. Essa lista pode ser considerada a WBS (Work Breakdown Structure) ou a EAP (Estrutura Analítica do Projeto).

Algumas atividades executadas podem ser as seguintes:

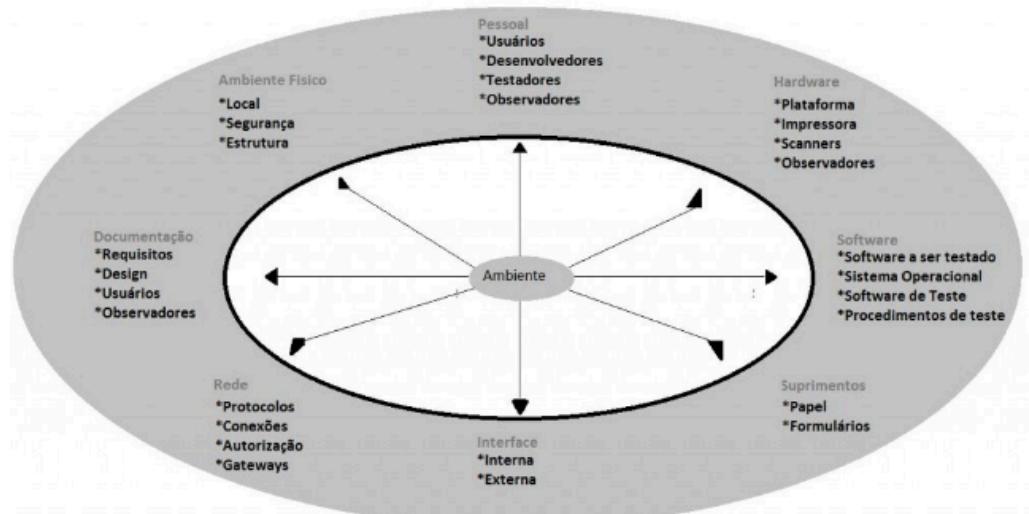
- Participação na revisão da documentação de desenvolvimento;
- Elaboração da lista de requisitos de teste usando os requisitos do sistema;
- Elaboração do Planejamento dos Testes (Plano de Testes e Estratégia de Testes);
- Preparação do ambiente de testes;
- Elaboração dos Casos de Teste;
- Preparação da massa de teste;
- Execução dos testes (programas e scripts);
- Registro dos defeitos encontrados;
- Documentação dos resultados do teste;
- Revisão dos resultados do teste;
- Manutenção da documentação de testes (bibliotecas de casos de testes, etc.).

Exemplo: Usando a lista anterior, por exemplo, devem ser identificados quais os produtos gerados por cada uma das atividades e se possível a quem caberá a sua execução.

Na parte final deste livro pode ser encontrada uma lista de atividades sugeridas para um projeto de teste.

## Necessidades de ambientes e infraestrutura

Especificar o ambiente necessário para a execução dos testes. Muitos autores consideram como ambiente de teste os seguintes elementos: software, hardware, comunicação, ambiente de trabalho, facilidades, pessoal, etc. No entanto deve ser tomado cuidado com outros itens desse documento que já contemplam alguns dos elementos do ambiente de teste. O gráfico segue um modelo clássico, mas não necessariamente está alinhado com o modelo clássico do projeto de teste de software.



Disponível em: <<https://www.tiespecialistas.com.br/ambiente-de-teste/>>. Acesso em 27 ago. 2023.

Seja qual for a abordagem adotada, o ambiente necessário deve estar descrito em detalhes, assim como todas as providências necessárias para que o mesmo seja implantado no momento certo, ou seja, quando for o momento de ser usado.

## Responsabilidades (Quem?)

Identificar os grupos responsáveis pela gerência, projeto, preparação, execução e revisão, além de outras atividades, no projeto de teste de software.

Deve ser também explicitado quem deverá fornecer os itens de teste e quem irá preparar o ambiente de teste.

Esses grupos deverão ser compostos por desenvolvedores, testadores, analistas de suporte, usuários, analistas de qualidade, etc.

Os principais envolvidos ("stakeholders") no projeto com as suas funções devem estar listados e identificados. Por exemplo, Fulano é responsável pela Gerência de Requisitos.

Exemplo: Uma abordagem simples é identificar os seguintes stakeholders: gerente do projeto de teste, gerente do projeto de desenvolvimento, usuários envolvidos caracterizando a sua responsabilidade, responsável pela montagem do ambiente de teste, responsável pelo acompanhamento dos riscos do projeto, responsável pela condução dos treinamentos necessários (veja o exemplo da aquisição de uma nova ferramenta de teste de carga), onde muitas vezes será necessário treinar a equipe de teste, além de outros possíveis envolvidos como o gerente de requisitos, o gerente de configuração, etc.

## Interfaces entre as partes envolvidas

Descrever como será feita a comunicação entre as partes envolvidas no projeto, principalmente, se houverem recursos externos. Dentro do conceito de gerência de comunicação podemos considerar que essa interface esteja incluída nesse plano.

## Necessidades de equipe e de treinamento

Para o projeto devem ser listadas as necessidades por tipo de qualificação que serão demandadas.

Devem também ser definidas as necessidades de treinamento para as equipes do projeto.

As empresas normalmente mantêm um arquivo com uma relação dos seus empregados e colaboradores onde estão incluídas informações sobre a sua formação técnica. Além disso, deve haver uma forma para o líder do projeto identificar as disponibilidades (com datas previstas) de cada profissional. A partir dessa informação deverá ser definida a equipe de projeto e as suas necessidades de treinamento. Por exemplo, aquele projeto envolve o uso de uma ferramenta nova, que nunca foi usada pela equipe de teste. Isso possivelmente vai precisar de um plano de treinamento.

## Cronograma, estimativas e custos

O cronograma deve ser o mais detalhado possível, incluindo inclusive os pontos de controle para a monitoração do projeto. Para cada tarefa deverá ser estimado o tempo necessário para a sua execução.

Como a norma não fala em medições de tamanho, para preparar o cronograma será necessária a utilização de alguma técnica de estimativa, seja através de Pontos de Teste ou Pontos de Caso de Teste, dentre outras. O importante é que haja um racional e um histórico para dar suporte a esse serviço de estimativa. Existem outros critérios de estimativas como a metodologia Wideband-Delphi ou a complexidade de caso de uso, que junto com um histórico de medições passadas, podem ser usados para fazer as estimativas do projeto de teste.

Todos os pontos de controle do projeto de teste devem ser claramente identificados, assim como a sua relação com o projeto de desenvolvimento.

As atividades e tarefas definidas em outro item do PT devem estar listadas com os seus respectivos prazos de conclusão nesse item.

Exemplo: Um bom ponto de partida pode ser pegar as atividades e tarefas do teste (ver anteriormente) e definir prazos para cada uma delas. No entanto, aconselha-se usar algum critério

técnico para a elaboração do cronograma. A primeira sugestão seria dimensionar o projeto de teste, medindo o seu tamanho, por exemplo, em pontos de teste. Depois estimar o tempo necessário para a sua execução.

Uma base de indicadores históricos poderá ajudar para que as medições sejam cada vez mais precisas.

Existem outras técnicas, como a Delphi, onde o tempo para a execução é estabelecido através de sugestões emitidas por técnicos experientes. O tempo total deve ser distribuído depois pelas tarefas ou atividades do projeto.

Por outro lado, a distribuição do tempo total estimado por etapas do ciclo de vida do projeto, vai depender também de indicadores históricos. Por exemplo, poderíamos ter uma informação histórica que nos indique que o planejamento consome em média 40% do tempo do projeto. Para poder chegarmos a essa informação, precisaríamos ter coletado dados do esforço despendido em projetos de teste já concluídos.

## Riscos e contingências

Segundo o PMI, qualquer projeto deve identificar os seus riscos e preparar planos de mitigação e planos de contingência. Um projeto de teste deve seguir as mesmas regras. As empresas normalmente possuem listas pré-definidas dos riscos mais comuns, pois a maior parte delas já faz parte dos mesmos tipos de projetos da mesma empresa, ou seja, os riscos tendem a se repetir na sua maior parte. O projeto de teste de software deverá ter uma lista própria e esses riscos precisam ser monitorados através das reuniões agendadas nos pontos de controle do cronograma.

Uma lista de riscos deve ser preparada e incluirá os riscos usuais além daqueles específicos do projeto de teste do software que esteja sendo testado. Os planos de mitigação e de contingência para cada um dos riscos devem ser elaborados. Evidentemente que os riscos devem ser monitorados durante todo o projeto para que seja evitada a sua ocorrência.

Plano de mitigação: Servem para evitar que os riscos ocorram.

Plano de contingência: Se o risco ocorreu e já virou um problema, precisamos ter um plano que diminua ou contorne o impacto da ocorrência do risco.

Observação: Lembre-se que as probabilidades de ocorrência dos riscos mudam durante o andamento do projeto.

Exemplo: Numa primeira abordagem podemos concluir que no projeto de Controle de Viagens temos um risco que é o seguinte: Aquisição de ferramenta para automação do teste de carga. Caso essa ferramenta não esteja disponível no prazo, assim como a equipe treinada no seu uso, teremos um problema que poderá atrasar a conclusão dos testes. Outros riscos possíveis poderiam ser: Não disponibilidade da equipe de teste (não existem profissionais disponíveis no momento); Falta de qualificação da equipe (a maioria é ainda estagiário), etc. Para cada risco deve ser estabelecido um plano de mitigação (para que não ocorra) e um plano de contingência (para o caso de vir a ocorrer). Existem livros específicos sobre análise de riscos no mercado que podem fornecer mais detalhes sobre esse tipo de planejamento.

Esta seção do plano de teste pode estar subordinada à uma seção correspondente no plano

máster de teste. Neste caso deverá ser explorado um nível de detalhe maior ou incluir apenas os riscos específicos deste nível do projeto. Ou seja, se os riscos forem semelhantes em todos os planos de teste, basta listá-los no PMT.

## Geral

### Procedimentos de garantia de qualidade

Deve ser especificado quais os meios que serão utilizados para garantir a qualidade dos processos e produtos de teste. Normalmente existe um Plano de Controle de Qualidade que define esses critérios e também uma Equipe de Controle de Qualidade responsável por esse controle.

## Métricas

Devem ser especificados quais os indicadores que serão coletados no decorrer do projeto de teste de software. Esses indicadores devem estar definidos no Plano de Controle de Qualidade, caso o mesmo exista na organização. Os indicadores deverão também alimentar um histórico que permita fazer estimativas para os projetos de teste de software.

Exemplo: Algumas empresas ou organizações usam classificações como complexidade de casos de uso ou complexidade de requisitos para identificar esses elementos. Posteriormente são coletadas informações que liguem os requisitos ao número de casos de teste e ao tempo de teste do requisito ou caso de uso. Temos desta forma um elemento que permitirá estimativas futuras quando quisermos saber quanto tempo precisamos para executar um determinado teste.

Cabe lembrar que as métricas devem considerar o plano de teste especificamente que está sendo preparado. Caso seja, por exemplo, um Plano de Teste Unitário, as métricas dirão respeito apenas a esse plano.

## Cobertura dos testes

A cobertura dos testes vai depender do nível do plano de teste que está sendo executado. Por exemplo, quando tivermos um plano de teste unitário a cobertura poderá ser o número de linhas de código cobertas pelo teste. No caso de um plano de teste de sistema, a cobertura poderá ser o número de requisitos testados. Outros itens de cobertura podem ser definidos e usados, porém o objetivo sempre será definir o grau de profundidade ou extensão até onde o teste deverá ser executado.

### Procedimentos de alterações do documento e histórico de alterações

Este item do plano serve para registrar as alterações feitas no plano e quem foi o responsável pela sua atualização.

Além disso, deverá ser definido em que condições o plano poderá ser alterado e quem terá a

responsabilidade pela sua alteração e quais são os responsáveis pela aprovação das alterações. Caso exista um Plano de Configuração, este deve estar incluído no Plano de Teste, porém, caso não exista, procedimentos de segurança devem ser definidos para garantir a manutenção das diversas versões do PT.

É importante lembrar que cada alteração efetuada implicará no registro do responsável pela alteração e de quem aprovou a mesma. Isso poderá implicar diversos níveis de aprovação ou de alteração.

O Plano de Teste (PT) poderia ter um item que seria o plano de configuração. Por outro lado, o PT, ele próprio, será também um dos artefatos controlados pelo plano de gerência de configuração.

## Glossário

Preparar um glossário dos principais termos usados no projeto de teste de software, desde que este não esteja contemplado no PMT ou em outros documentos de maior nível.

## Observações sobre algumas mudanças no Plano de Teste na nova versão da norma.

A norma IEEE 829-1998 não falava especificamente em orçamento, o que já foi corrigido nesta sua nova versão.

A versão anterior também não definia claramente o uso de métricas e da coleta de indicadores, sejam de qualidade, ou para a formação de um histórico para servir de base para estimativas de projetos. Essa falha foi corrigida nesta nova versão.

## Desenho ou design (projeto) de teste

O leitor deve tomar um cuidado com o termo Projeto de Teste. O termo em inglês é Test Design que resolvemos traduzir como Projeto de Teste na versão da norma de 1998. No entanto deve haver uma precaução adicional pois temos também uma terminologia que caracteriza os projetos de teste, neste caso, aquele projeto que segue paralelo ao projeto de desenvolvimento.

Agora temos um documento chamado Projeto de Teste e um não tem nenhuma relação com o outro, a não ser o fato deste ser um documento usado em um projeto. Sugerimos que seja usado o termo Desenho de Teste para evitar a confusão com a terminologia do projeto de teste.

O propósito do documento Projeto ou Desenho de Teste é refinar a abordagem do teste definida no Plano de Teste para identificar os requisitos e as features (funcionalidades) que serão testadas e quais os tipos de teste deverão ser usados. Em muitas empresas esse documento é redundante, visto que poderá ser coberto pelos Procedimentos de Teste e pelos Casos de Teste. No entanto, em grandes empresas, onde os softwares desenvolvidos são complexos e extensos, se faz necessário criar uma passagem intermediária para agrupar alguns procedimentos. Usar um Plano de Teste Máster quebrado em vários Planos de Teste específicos é um recurso para melhor organizar os artefatos do projeto de teste. Além disso, para uma melhor organização, pode ser que o Plano de Teste tenha que ser dividido em outros documentos chamados Projeto ou Desenho de Teste.

Trata-se de uma forma de melhorar a distribuição dos artefatos usando documentos menores e não tão pesados como poderia ficar se usássemos um único documento.

Lembre-se que em projetos pequenos pode não ser necessário usar esse documento, que estará incorporado dentro do Plano de Teste. O objetivo principal do Projeto ou Desenho de Teste é facilitar o entendimento e acompanhamento do Plano de Testes, através da sua abertura em diversos Projetos/Desenhos de Teste.

O documento Desenho ou Projeto de Teste deve conter os seguintes campos:

**1) Introdução**

- Identificador;
- Escopo;
- Referências;

**2) Detalhes deste nível do Desenho (Projeto) de Teste**

- Features (ou funcionalidades) a serem testadas;
- Abordagem refinada;
- Casos de teste com a sua respectiva identificação;
- Critérios de passagem e falha por característica ou funcionalidade;
- Entregáveis;

**3) Geral**

- Glossário;
- Procedimentos de alterações do documento e histórico de alterações.

## Introdução

**Identificador:** Código único que identifique o projeto de teste e que mantenha relação com o Plano de Teste.

**Escopo:** Neste caso o escopo deveria ser uma parte do escopo do Plano de Teste a que este documento está se referindo. Poderia, por exemplo, se referir a um grupo de funcionalidades, que deverão ser listadas depois em outro campo.

**Referências:** No Plano de Teste a que este documento se refere encontramos as seguintes referências:

Referências internas:

- Autorização do projeto;
- Plano do projeto de desenvolvimento;
- Plano de garantia de qualidade;
- Plano de gerência de configuração;
- Lista de requisitos;
- Etc.

Referências externas:

- Leis
- Regulamentos governamentais

- Padrões
- Políticas

Devemos no caso do Projeto ou Desenho de Teste selecionar quais as referências que são específicas deste documento.

## Detalhes deste nível do documento Desenho ou Projeto de Teste

### Features ou funcionalidades a serem testadas

Devem ser listadas todas as funcionalidades que serão cobertas por este Projeto ou Desenho de Teste e a sua relação com os requisitos. Por exemplo, cadastrar usuário pode ser uma funcionalidade e também um requisito. Algumas empresas usam a terminologia Cenário de Teste para definir uma funcionalidade ou grupo de funcionalidades do software que serão testadas por um grupo de casos de teste. Cada cenário seria composto por um conjunto de casos de teste. Poderíamos ter outro desenho de teste composto por outros cenários de teste que mantivessem um nível de integração, caso isso ocorra. Desta forma teríamos vários projetos de teste e um único Plano de Teste.

### Abordagem refinada

A abordagem usada no Plano de Teste deve ser agora refinada. Devem ser detalhadas quais as técnicas de teste que serão adotadas. Por exemplo, se a opção for para fazer teste caixa branca, deve ser especificado como esse teste deverá ser conduzido, inclusive se serão necessárias ferramentas. Métodos para avaliação de resultados devem ser descritos, caso, por exemplo, quando algum tipo de software venha a ser usado. Dentro da abordagem refinada devem ser especificados os atributos dos casos de teste deste projeto. Ao aprofundar a abordagem dada no Plano de Teste, deve ser incluída a especificação das técnicas de teste que serão usadas, os métodos necessários para avaliar os resultados dos testes (ex. visual, programa para comparar resultados, etc.), atributos ou restrições comuns a todos os casos de teste (ex. algum software precisa estar instalado antes da execução dos casos de teste).

### Casos de teste com a sua respectiva identificação

Fazer uma breve descrição dos casos de teste desse projeto ou desenho com a sua respectiva identificação. Deve ser mantida uma associação entre os casos de teste e as funcionalidades listadas. Esse cuidado vai facilitar a montagem da matriz de rastreabilidade do projeto. Para isso, os testes precisam ser conferidos através de um programa que bata as informações ou por controle visual acessando os dois cadastros.

### Entregáveis

O Plano de Teste tem uma lista de entregas do projeto de teste que são as seguintes:

- Plano de Teste (nos seus diversos níveis);
- Projeto de teste;

- Casos de teste;
- Procedimentos de teste;
- Relatórios (listar todos);
- Dados de teste usados nesse projeto;
- Programa e componentes;
- Ferramentas de teste;
- Outros artefatos gerados pelo projeto de teste.

No caso específico deste documento, Projeto ou Desenho de Teste, devem ser escolhidos os artefatos que serão entregues nesta etapa.

## Geral

### Glossário

Preparar um glossário dos principais termos usados neste documento projeto ou desenho de teste de software, caso já não estejam no PT ou no PMT.

### Procedimentos de alterações do documento e histórico de alterações.

Este item do documento serve para registrar as alterações feitas e quem foi o responsável pela sua atualização. É importante lembrar que cada alteração efetuada implicará no registro do responsável pela alteração e de quem aprovou a mesma. Isso poderá implicar diversos níveis de aprovação ou de alteração.

**Material Complementar:** Como criar ambientes de testes dinâmicos usando Kubernetes + GitHub Actions Acompanhe o vídeo para criar um ambiente de teste utilizando a Plataforma da Microsoft e a GitHub Actions.



Fonte do vídeo: [Como criar ambientes de testes dinamicos usando Kubernetes + GitHub Actions](#)



O capítulo conclui que o processo de teste de software requer uma documentação adequada, definida no Plano de Teste. Esses documentos devem seguir templates padronizados pela organização. Caso seja necessário algum documento adicional, ele deve ser definido neste item.

O Plano de Teste deve detalhar a abordagem de teste, identificar os itens de teste e suas respectivas matrizes de rastreabilidade, bem como definir as funcionalidades a serem testadas e os critérios de liberação/falha dos itens testados. Também são destacadas as necessidades de equipe, treinamento e ambiente de teste, bem como a importância de garantir a qualidade dos processos e produtos de teste através de métricas e indicadores. Por fim, é sugerido que o Plano de Teste possa ser dividido em vários Projetos ou Desenhos de Teste, especialmente em projetos maiores e mais complexos.



## ATIVIDADE DE FIXAÇÃO

- 1.** Quais são os documentos necessários para um projeto de teste de software de acordo com a norma IEEE-829?
- 2.** O que é um glossário no contexto de teste de software, e qual é o benefício de ter apenas um glossário para todo o projeto?
- 3.** Explique a importância de registrar as alterações feitas no Plano de Teste e quem é o responsável por sua atualização.
- 4.** Quais são os principais campos que devem estar presentes no Plano de Teste?
- 5.** O que é uma matriz de rastreabilidade de teste e por que ela é importante para o processo de teste?
- 6.** Cite pelo menos três exemplos de funcionalidades que podem ser listadas para serem testadas em um projeto de software.
- 7.** Descreva a abordagem do teste de sistema e explique como ela difere da abordagem de teste de unidade.
- 8.** O que são requisitos de suspensão e retomada no contexto de um Plano de Teste e por que eles são importantes?
- 9.** Comente sobre a importância de definir as responsabilidades dos envolvidos no projeto de teste de software e como isso contribui para o sucesso do projeto.
- 10.** Explique a importância das interfaces entre as partes envolvidas no projeto de teste de software e como elas facilitam a comunicação e colaboração entre os diferentes grupos.

## TEMA 07

# Virtualização

### Habilidades:

- Promover a inclusão digital em ambientes virtuais e sistemas. ;
- Promover acessibilidade e naveabilidade



Disponível em: <[rawpixel-https://tinyurl.com/354bwmc5](https://tinyurl.com/354bwmc5)>. Acesso em 27 ago. 2023.

O sistema operacional é como um grande “orquestrador” de cada um destes componentes, decidindo como utilizá-los para atender da melhor maneira possível as demandas oriundas dos processos ativos (ou programas em execução).

No entanto, se o sistema operacional está limitado aos recursos físicos do hardware, como explicar os modernos ambientes na nuvem, da qual arrastamos uma barra horizontal, e em um passe de mágica (e por uma centena de dólares), temos mais processador, memória e espaço de armazenamento?

**Material Complementar:** Virtualização - Dica 01 - O que é virtualização? Acompanhe o vídeo para entender sobre o Conceito de Virtualização e entender os conceitos e usos práticos do dia a dia.

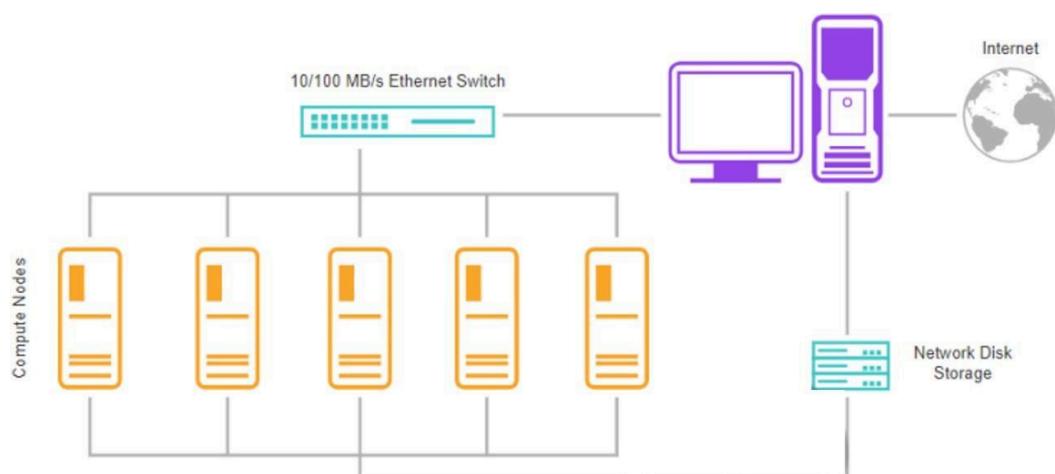


Fonte do vídeo: [Virtualização - Dica 01 - O que é virtualização?](#)

## Como funciona?

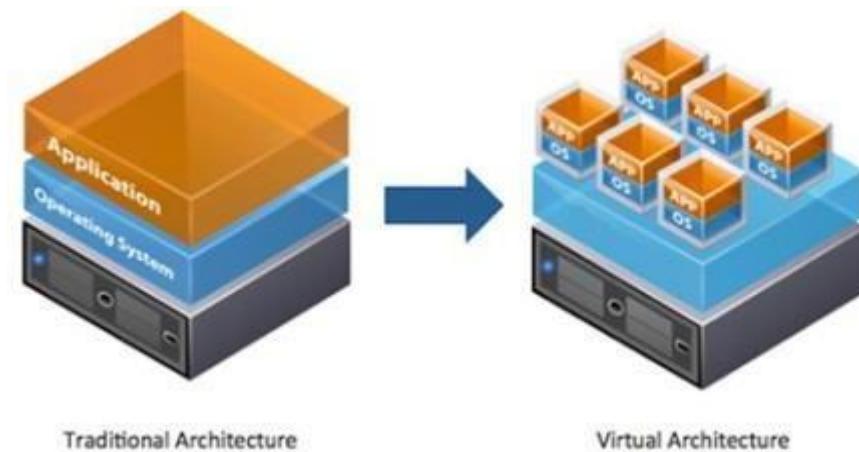
A virtualização surgiu como uma forma de separar melhor o hardware do software. Conforme mencionado, no modelo tradicional os softwares são instalados em um sistema operacional que, por sua vez, está instalado sobre uma infraestrutura física (o hardware). No entanto, sempre que os softwares demandam por mais infraestrutura física para suportar mais usuários ou serviços, o hardware precisava ser substituído por outro mais poderoso (processadores mais rápidos, mais memória e por aí vai), o que é chamado de upgrade vertical.

Os clusters de computadores (que são computadores ligados entre si dividindo as tarefas e “juntando forças” para resolver problemas computacionais mais complexos) inauguraram a era upgrades horizontais: assim, sempre que fosse necessário atender mais usuários ou trabalhar com um volume maior de dados, bastava adicionar novos computadores a esta verdadeira “força tarefa”. Ainda neste modelo, cada um dos computadores colocados neste cluster mantinha sua própria estrutura de sistema operacional e, portanto, seus próprios processos.



Fonte: Elaborado pelo autor (2023).

A virtualização possibilitou a abstração do hardware, pois uma camada de software mimetizando uma infraestrutura física é inserida entre o hardware (real) o sistema operacional, possibilitando uma separação mais eficiente destas duas camadas. A virtualização permite, por exemplo, a instalação de vários sistemas operacionais em um mesmo equipamento.



Disponível em: <[https://www.teleco.com.br/tutoriais/tutorialdatacenter1/pagina\\_4.asp](https://www.teleco.com.br/tutoriais/tutorialdatacenter1/pagina_4.asp)>. Acesso em 27 ago. 2023.

Não demorou muito para surgir um inverso: Um único sistema operacional (chamado neste contexto de máquina virtual) instalada sob uma única camada de virtualização que compreende diversos computadores abaixo dela (o cluster!). Nossa máquina virtual “enxerga” um único hardware (que se comporta como um único supercomputador com vários núcleos e muita, muita memória para armazenamento), permitindo uma flexibilidade muito grande no aumento destes recursos para a máquina virtual. Sempre que esta precisar atender mais usuários, poderá receber rapidamente mais processamento e memória e, por sua vez, sempre que a infraestrutura física for insuficiente, novos computadores serão adicionados ao cluster em um upgrade horizontal.

O Cloud Computing é o compartilhamento desta infraestrutura entre diversas necessidades e clientes, em um modelo de negócio e cobrança diferenciado.

## Benefícios

Esta tecnologia permite simular aplicações, servidores, armazenamento e redes, sendo capaz de reduzir custos, aumentar eficiência, agilidade, flexibilidade e escalabilidade dentro de uma infraestrutura corporativa.

Com a virtualização é possível consolidar servidores, cargas de trabalho e ambientes, aumentar a utilização de recursos e acelerar a implantação de desktop e aplicativos.

As principais vantagens são:

- Redução de custos:

A virtualização pode reduzir investimentos em hardware, energia e espaço físico, isso porque permite aproveitar recursos físicos que ficam ociosos.

Lembre-se que o processador, memória e outros recursos não são 100% exigidos o tempo todo, até porque alguns processos exigem mais processamento (chamados de CPU bound) e outros exigem mais das memórias (chamados de I/O bound), como os bancos de dados. Se em um exemplo hipotético a maioria dos processos de um sistema operacional exige, no máximo, 40% do processador, temos, então, 60% do processador “sobrando”, certo?

A virtualização permite que um segundo sistema operacional ocupe os 60% restantes do processador sem qualquer prejuízo para os processos do primeiro. Sensacional, não é mesmo?

- Redução do tamanho do parque de equipamentos (datacenter):

Com o melhor aproveitamento dos recursos atuais, a necessidade de aquisição de novos equipamentos diminui, reduzindo gastos com instalação, refrigeração, espaço físico, manutenção, consumo de energia e assim por diante.

- Gerenciamento centralizado:

Torna-se mais fácil monitorar serviços/processos em execução, já que o gerenciamento destes é feito de forma centralizada (no entanto, isso depende da estratégia de virtualização adotada).

- Manutenção de sistemas legados:

Um grande desafio nas grandes empresas é, sem dúvida alguma, a manutenção de sistemas legados. Não é raro encontrar um equipamento cujo hardware possui mais de vinte anos de idade, rodando um sistema operacional absolutamente obsoleto com o único objeto de manter “no ar” uma aplicação ou serviço que “só roda naquela configuração específica”.

Pois bem, a virtualização permite simular aquele hardware que já deveria ter sido aposentado há muito tempo, e instalar nesta infraestrutura emulada aquele Windows 3.11 (pergunte a seus pais) para fazer uma hora extra que até a Microsoft ficaria espantada. E tudo isso em componentes físicos que vão mergulhar no sono eterno a qualquer momento.

- Ambientes de testes:

A virtualização permite até mesmo emular hardwares de arquiteturas diferentes. Como testar uma aplicação feita para funcionar em smartphone Android ou um iPhone? Claro que você pode comprar os equipamentos, mas nem sempre é uma opção: um desenvolvedor de aplicativos preocupado com seus usuários teria que possuir vinte smartphones diferentes (chutando baixo). Por meio da virtualização, é possível simular uma arquitetura ARM (Advanced RISC Machine), típica de smartphones, em uma arquitetura x86-64 bits (dos microcomputadores).

- Confiabilidade e Segurança:

Já que cada máquina virtual (VM) funciona de maneira independente, se um problema surgir em uma delas (como uma vulnerabilidade de segurança, por exemplo) este não afetará as demais;

- Migrações e ampliações mais simples:

Trocar um fornecedor da infraestrutura Cloud Computing é mais simples, pois a VM pode ser exportada e importada em outro ambiente com facilidade (a alternativa seria montar o sistema todo de novo, “do zero”). Além disso, ampliar o hardware demandando mais processamento, memória ou armazenamento se torna mais flexível, graças ao compartilhamento de hardware e upgrades horizontais.

## Infraestrutura

Pode-se dizer que uma Máquina Virtual (ou Virtual Machine, VM), nada mais é que um computador emulador, que possui processador, memória, armazenamento, rede, interfaces (como a USB) e até periféricos emulados, com a capacidade de executar as mesmas funções que um computador físico.

A virtualização é uma tecnologia de software já consagrada que muda fundamentalmente a maneira da computação em tirar componentes de camadas e consolidar recursos em pools. Podendo executar várias máquinas virtuais em uma única máquina física assim como vários Sistemas

Operacionais (Windows, Linux, Solaris, entre outros) além de compartilhar os recursos desse computador único (CPU, memória, dispositivo de rede, armazenamento, este software que permite criar e executar máquinas virtuais é chamado de Hypervisor.

Com a Virtualização podemos fornecer uma versão virtual de muitas tecnologias essenciais em computação, como principais podemos citar Hardware, Armazenamento e Redes:

- **Hardware:** esse é o um dos principais itens dentro da tecnologia de virtualização, um sistema operacional pode ser instalado sobre outro tipo de sistema, com seus recursos de hardware sendo representados via software.
- **Armazenamento:** também conhecido pelo nome de Software Defined Storage - SDS (Armazenamento Definido por Software). É uma camada de software criada sobre discos físicos, na qual os dispositivos acessam esses discos de modo a tornar o acesso mais flexível, gerenciável e personalizável.
- **Rede:** Chamado de Software Defined Networking – SDN (Rede Definida por Software). É possível criar um tipo de infraestrutura lógica (software) de redes sobre uma determinada rede física, permitindo a configuração e o detalhamento de acordo com as necessidades do ambiente.
- Ao usar essas técnicas de virtualização, todos os dispositivos físicos podem ser representados em forma de softwares: servidores e estações de trabalho se tornam Máquinas Virtuais (VMs/Virtual Machines), a rede e o storage são virtualizados, transformando-se em SDN e SDS respectivamente. Com isso, construímos o que conhecemos por e SDDC – Software Defined Data Center (Data Center Definido por Software).

Vamos convencionar alguns termos a partir de agora:

- **Hospedeiro (host):** trata-se da infraestrutura real, local da qual as VMs ficaram hospedadas; ele pode ter seu próprio sistema operacional (instalado diretamente no hardware, de forma convencional) ou não.
- **Hypervisor:** é uma camada de software localizada entre a camada de hardware e o sistema operacional.
- **Convidado ou hóspede (guest):** é a máquina (VM) instalada sob o hardware emulado.

## Tipos de Virtualização

Existem vários tipos de virtualização. Vamos classificá-las a seguir.

### Virtualização de aplicativos

A virtualização de aplicativos fornece um aplicativo hospedado em uma única máquina para um grande número de usuários. O aplicativo pode estar situado na nuvem em máquinas virtuais de alta qualidade, mas, como um grande número de usuários o acessa, seus custos são compartilhados

por esses usuários.

Isso torna o aplicativo mais barato para entregar ao usuário final. O usuário final não precisa ter hardware de alta qualidade para executar o aplicativo; uma máquina barata, como uma estação de trabalho de baixo custo ou um terminal thin client, será suficiente.

E se os dados usados pelo aplicativo virtual são armazenados na nuvem, o usuário não está ligado a nenhum dispositivo ou local para usar a aplicação ou acessar seus dados. Normalmente, nesses casos, o aplicativo virtual é consumido por meio de um aplicativo móvel ou de um navegador da Internet pelo usuário final.

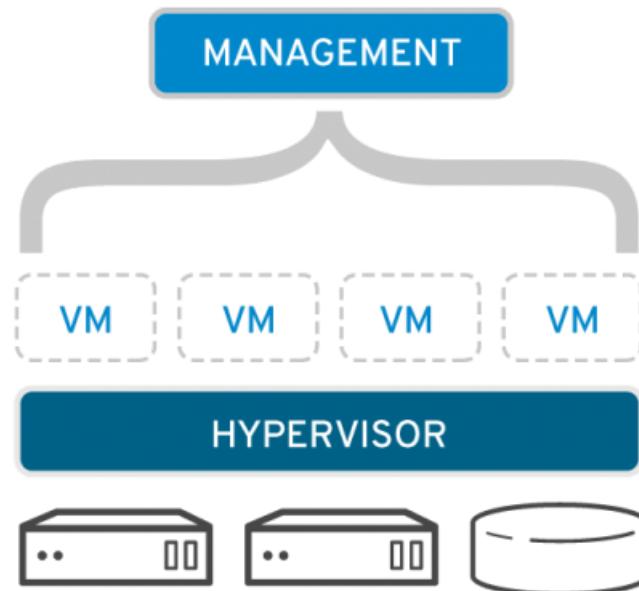


Fonte: Elaborado pelo autor (2023).

Para citar um caso de uso deste tipo de virtualização, lembramos que no início de 2019, a Google anunciou seu projeto Stadia, uma plataforma de jogos por streaming. A ideia da empresa é ter um hardware de baixo custo a ser “plugado” na TV do assinante (como uma espécie de Google Chromecast) apenas para fazer a interface com o joystick e rodar a aplicação do serviço; os jogos, que por sua vez requerem um alto poder de processamento, rodarão nos servidores da empresa.

### Sistema virtualizado por máquina virtual (H-based)

H-based é o tipo mais comum. Uma máquina virtual (VM, do inglês Virtual Machine) requer um Sistema Operacional (SO) completo e exclusivo, além de kernel próprio, binários, aplicativos e bibliotecas, o que exige dimensionar espaço grande no servidor e custo de manutenção. Diversas VMs com SOs diferentes podem ser executadas em uma mesma infraestrutura física

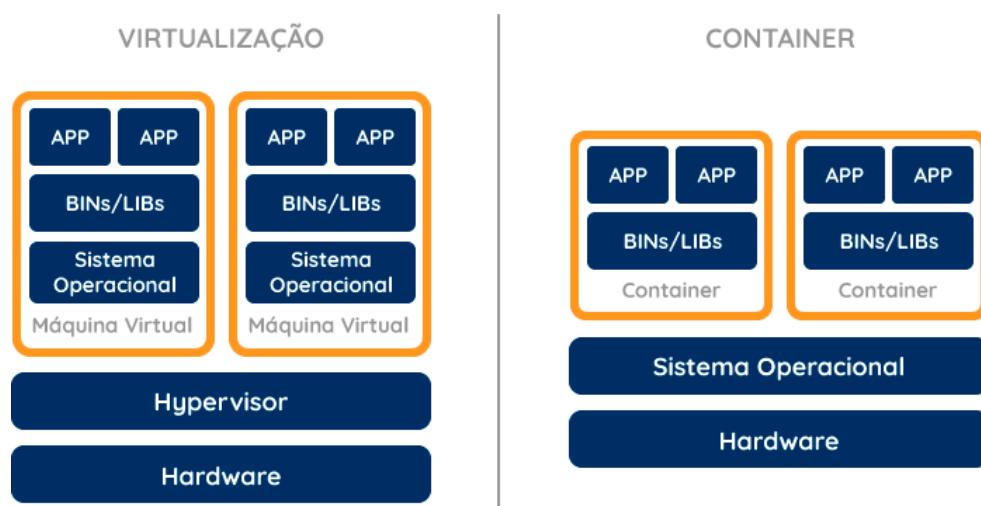


Fonte: Elaborado pelo autor (2023).

Virtualização por VM necessita uma camada intermediária, chamada hypervisor, para gerenciar a comunicação de cada VM com o SO hospedeiro (exemplo: VMWare, Hyper-V e o VirtualBox). O hypervisor é um elemento fundamental para virtualizar o servidor, pois é responsável por criar e executar VMs, com o intuito de possibilitar que um determinado software seja executado sobre um servidor físico para emular um sistema de hardware.

### Sistema virtualizado por container (OS-based)

Implementação de virtualização de containers estão fazendo sucesso na Cloud Computing devido à facilidade de uso, além de possibilitar melhor gerenciamento de serviços e melhor gestão dos recursos computacionais da nuvem.



Fonte: Elaborado pelo autor (2023).

Virtualizar por container possibilita, por exemplo, portar sua aplicação diretamente do seu

notebook para o servidor de produção ou para uma instância virtual em uma nuvem pública.

A depender da literatura, os containers podem ser chamados de sandboxes (ou “caixas de areia”).

Em suma, a virtualização é o coração da computação em nuvem (Cloud Computing) responsável por esta grande revolução. Afinal, é graças a essa e outras tecnologias que startups no mundo inteiro ganham escalabilidade na velocidade que tanto precisam.

Considerando a importância da infraestrutura empresarial baseada em cloud, é nesse tipo de estrutura que a maioria dos ambientes a se proteger estarão no futuro, tornando este aprendizado obrigatório para um profissional de segurança da informação, cujas máquinas virtuais são uma de suas mais importantes ferramentas.



## RESUMO

A virtualização é um tema essencial no cenário tecnológico atual, que envolve a criação de ambientes virtuais e sistemas para uma série de aplicações. Uma das principais habilidades associadas a essa temática é a promoção da inclusão digital. Ao adotar a virtualização, as barreiras geográficas e físicas são minimizadas, permitindo que indivíduos de diversas localidades tenham acesso a recursos e informações, fomentando assim a inclusão digital e ampliando oportunidades para pessoas de todas as origens.

Além disso, a virtualização também se destaca por sua capacidade de promover acessibilidade e naveabilidade. Através da criação de ambientes virtuais bem projetados, é possível oferecer interfaces amigáveis que atendam às necessidades de diferentes usuários, incluindo aqueles com deficiências visuais, auditivas ou motoras. Essa abordagem inclusiva torna os sistemas mais acessíveis a um público diversificado, contribuindo para a construção de uma sociedade digital mais equitativa.

No contexto empresarial, a virtualização desempenha um papel crucial na otimização de recursos e na eficiência operacional. Ao criar ambientes virtuais para servidores e infraestrutura de TI, as organizações podem consolidar recursos físicos, reduzir custos de manutenção e energia, e ao mesmo tempo aumentar a flexibilidade e escalabilidade de suas operações. Isso demonstra como a virtualização se liga à capacidade de promover ambientes mais sustentáveis e economicamente viáveis.

Contudo, é importante considerar os desafios que a virtualização pode apresentar. A segurança cibernética, por exemplo, se torna ainda mais crítica, uma vez que os ambientes virtuais podem estar sujeitos a ataques e invasões. A implementação adequada de medidas de proteção e a conscientização sobre práticas seguras são fundamentais para garantir a integridade e a confidencialidade dos dados em ambientes virtuais.

Em síntese, a virtualização é um tema multifacetado que engloba a promoção da inclusão digital, a acessibilidade, a otimização de recursos e a segurança. Ao explorar essas habilidades, indivíduos e organizações podem aproveitar os benefícios dessa abordagem, construindo ambientes virtuais mais inclusivos, eficientes e protegidos.



## ATIVIDADE DE FIXAÇÃO

- 1.** O que é virtualização? O que é virtualização?
- 2.** Quais são os principais benefícios da virtualização?
- 3.** Qual a função do sistema operacional em um ambiente virtualizado?
- 4.** Explique o conceito de "upgrade vertical" e "upgrade horizontal" no contexto da virtualização.
- 5.** Cite duas vantagens da virtualização relacionadas à redução de custos.
- 6.** Qual é o objetivo da virtualização de aplicativos?
- 7.** Diferencie a virtualização de aplicativos da virtualização por máquina virtual (H-based) em relação aos requisitos de infraestrutura
- 8.** Quais são os principais desafios enfrentados pelas empresas na manutenção de sistemas legados? Como a virtualização pode ajudar nesse cenário?
- 9.** Descreva como a virtualização possibilita a consolidação de servidores, cargas de trabalho e ambientes em uma infraestrutura corporativa, destacando suas implicações na eficiência e escalabilidade.
- 10.** Em um ambiente virtualizado, como as máquinas virtuais são capazes de compartilhar recursos físicos, como processador e memória, sem afetar o desempenho das aplicações?

## TEMA 08

# Teste Funcional

### Habilidades:

- Compreender como os testes funcionais funcionam no teste de sistemas.

Então, um dos conceitos mais importantes da atividade de teste de softwares são as técnicas e seus critérios. De forma geral, vamos iniciar considerando o cenário típico da atividade de teste. Então, considerando um programa qualquer, que eu estou chamando de P, esse programa tem domínio de entrada, que está representado aqui por essas bolinhas por exemplo.



Fonte: Elaborado pelo autor (2023).

O que é domínio de entrada? O conjunto de todos os valores que podem ser usados para executar o programa. A partir desse domínio de entrada, a gente pode selecionar alguns dados de teste. E o quê que é o dado de teste? Elemento do domínio de entrada do programa, que no caso a gente está chamando de P.

Então, cada dado de teste deste pode ser utilizado para compor caso de teste. E o quê que é caso de teste? Par contendo dados de teste como entrada e a saída esperada, o resultado esperado. Se eu tenho vários casos de teste, a gente chama isso de conjunto de casos de teste. E é esse conjunto de casos de teste que vai ser submetido ao programa para execução, a fim de verificar se existem defeitos no mesmo, está certo? E aí no final, o que a gente tem após executar o programa com esses casos de teste?

A gente tem que verificar se para cada caso de teste a saída obtida era igual à saída esperada que foi definida anteriormente aqui no caso de teste. Então este é um cenário típico de atividade de teste.

Agora a gente precisa pensar algumas questões, porque tendo programa e tendo domínio de entrada para esse programa, o ideal seria testar esse programa com todos os elementos do domínio de entrada. Então, supondo que o meu domínio de entrada é essa piscina de bolinhas, eu deveria testar todas as bolinhas da piscina de bolinhas.



Disponível em: <https://tinyurl.com/5fz8yucu>. Acesso em 27 ago. 2023.

O que é inviável para grandes domínios ou domínios infinitos. Isso é praticamente um teste exaustivo que é infactível, como foi colocado aqui, para grandes ou infinitos domínios. E o quê que a gente pode fazer nesse caso? O ideal seria selecionar subdomínios, a partir desse domínio de entrada, para garantir, mas garantindo ainda assim, mesmo dividindo, garantindo alta probabilidade revelar defeitos no software.



Fonte: Elaborado pelo autor (2023).

E, a partir desses subdomínios, a gente poderia selecionar, como a gente tinha verificado aqui antes, dados de teste que representam cada subdomínio desse para depois compor caso de teste e aí o percurso é o mesmo. Agora, como identificar esses subdomínios? Porque no exemplo anterior a regra que a gente usou para criar os subdomínios, olha só foi a cor.

Então a partir da cor a gente criou os subdomínios aqui das bolinhas. Agora como fazer isso com o software? Então, são definidas regras chamadas técnicas e essas técnicas possuem critérios que são utilizados para quê? Para identificar quando dados de teste devem estar no mesmo subdomínio ou subdomínios diferentes.

Então, basicamente é isso que essas técnicas fazem, a partir do domínio de entrada que a gente usa, as técnicas para separar domínios e a partir daí identificar os casos de teste relevantes. Então, aqui a gente tem um resumo das principais técnicas de teste, da atividade de teste.

A primeira delas é o teste funcional, depois a gente tem o teste estrutural, teste baseado em defeitos que inclusive são os testes tratados neste curso. O que diferencia uma técnica da outra? O tipo de informação utilizada para derivar os requisitos de teste.

O teste funcional usa a especificação do software; o teste estrutural usa a implementação, o

código-fonte para derivar requisitos de teste; e a técnica baseada em defeitos utiliza defeitos típicos do processo de implementação de software, uma técnica que trabalha um pouquinho diferente. E aí cada uma tem alguns passos principais que a gente precisa identificar.

De forma geral, o teste funcional a gente precisa identificar as funções e, a partir dessas funções, criar casos de teste capazes de verificar se essas funções estão corretas.

Já no teste estrutural a maioria dos critérios trabalha com a representação do programa, que a gente chama de grafo de programa e a técnica baseada em defeitos trabalha criando uma implementação alternativa e força o testador a projetar casos de teste que revelam os defeitos que foram introduzidos nessas implementações alternativas. E aí aqui a gente tem exemplos, alguns exemplos de critérios para cada uma dessas técnicas.

Os principais do teste funcional são o particionamento das classes de equivalência, análise do valor limite, combinatória, dentre outros. Os critérios da técnica teste estrutural são critérios baseados em complexidade, fluxo de controle, fluxo de dados e a técnica baseada em defeitos, o principal é realmente o teste de mutação e aí a gente tem aqui a análise de mutantes e a mutação de interface.

Também é importante considerar quais técnicas podem ser aplicadas e quais fases da atividade de teste. Então, por exemplo, o teste funcional de forma geral pode ser aplicado em todas as fases de teste: unidade, progressão, sistema. Já o estrutural de mutação é mais focado na parte de unidade e na parte de integração.

Aqui a gente tem um resumo sobre as técnicas de teste da atividade de teste. Então, de forma geral essas técnicas existem para ajudar a estabelecer subdomínios, considerando que a ideia de domínios de entrada muito grandes ou infinitos, onde o teste exaustivo é inviável. A diferença entre elas é o tipo de informação utilizada. Então algumas usam a especificação, por exemplo, o teste funcional, enquanto outras usam a implementação, o código si, no caso na estrutural e de defeitos.

As técnicas principais, o teste funcional, o teste estrutural e o de defeitos. O funcional, o particionamento classes de equivalência, análise do valor limite, combinatória, grafo causa-efeito, dentre outros.

O estrutural baseado na complexidade, no fluxo de controle ou no fluxo de dados e o defeito a gente tem como critério principal o teste de mutação. É importante que você verifique as fases de teste, quais técnicas podem ser aplicadas em cada fase: unidade, integração, sistema.

E também é importante identificar pesquisas, porque essas técnicas estão sendo evoluídas, têm sido criados novos critérios, novas técnicas, então é importante ficar atento na questão das pesquisas recentes nessa área também. Então, para terminar eu deixo para você algumas dicas de próximos passos na sua aprendizagem.

## Técnicas de Teste Funcional

Imagina o seguinte: você comprou lote, terreno e quer construir uma casa, a casa dos seus sonhos considerando a sua demanda, a sua necessidade.



Disponível em: <[freepik-https://tinyurl.com/5enp95mb](https://tinyurl.com/5enp95mb)>. Acesso em 27 ago. 2023.

## O quê que você faz?

Você contrata uma equipe com pedreiros, com engenheiro, que vai construir essa casa para você seguindo as suas vontades e necessidades.

Ao longo do desenvolvimento, ao longo da construção da casa, o quê que é normal?

Você faz visitas periódicas para verificar questões externas, por exemplo, será que a tinta que eu escolhi lá na loja ficou legal na parede da minha casa? Será que tem alguma rachadura? Será que tem alguma coisa que não está ficando conforme eu esperava?

Então, a gente pode compreender que o desenvolvimento de software é bem parecido com a ideia de construção de casas. E essa analogia pode ser utilizada por exemplo para compreendermos a técnica de teste chamada teste funcional. Por quê? Porque o teste funcional pode ser compreendido como sendo aquele proprietário que realiza visitas de vistoria externa na

sua casa. Então, considerando que eu tenho a especificação do requisito de software, especificação de requisitos de software.

## O que é uma especificação?

Demandas, necessidade, uma obrigatoriedade, desejo. Eu posso utilizar o teste funcional para projetar casos de teste, submeter esses casos no programa e verificar se a saída obtida está de

acordo com a saída que era esperada.

Então, o teste funcional de forma geral tende a responder perguntas desse tipo aqui:

- O usuário consegue fazer tal coisa no sistema?
- Tal funcionalidade funciona adequadamente, funciona da forma que deveria, da forma que foi identificado antes?
- Como resultado, o quê que a gente tem?
- São reveladas inconformidades com os objetivos especificados anteriormente. Que tipo de inconformidade?

Funções incorretas ou ausentes, erros na interface, erros na estrutura de dados, erros de acesso a softwares externos, bem como a inicialização e o encerramento, o término do programa.

Como os critérios da técnica de teste funcional baseiam-se na especificação do produto testado e não no código-fonte si, ela requer boa especificação dos requisitos para que bons casos de teste sejam projetados.

Pode ser aplicada todas as fases de testes, por exemplo, unidade, integração, sistemas, dentre outros; permite equipes independentes, por exemplo, eu posso ter equipe de desenvolvimento, equipe de teste, a equipe de teste pode ser separada: a equipe que projeta os testes e a equipe que executa ou automatiza o teste; pode ser realizado de forma manual, de forma automática ou a mistura de ambas; pode ser aplicada produtos criados diversos paradigmas e linguagens de programação, dentre outras questões.

Então, aqui a gente tem uma visão geral da aplicação do teste funcional. De forma geral, a gente precisa identificar as funções que o software deveria realizar; projetar os casos de teste que sejam capazes de checar se essas funções estão sendo realizadas pelo software; executar esses casos de teste e comparar se os resultados obtidos estavam de acordo com os resultados que eram esperados.

Então, com a técnica de teste funcional ela pode, pode ser aplicada para verificar todo o domínio de entrada, o que é teste exaustivo, principalmente quando o domínio de entrada é infinito ou muito grande, critérios são aplicados para quê? A fim de identificar bons casos de teste, casos de testes mais adequados e que possuam maior cobertura do software si.

Então, exemplos de critérios do teste funcional são: particionamento casos de equivalência, análise do valor limite, teste funcional sistemático, gráfico de causa e efeito, dentre outros critérios.

De forma geral, ela requer a especificação do produto para derivar casos de teste, o foco não é no código si. Considerando essa questão, ela reflete a ótica do usuário, do stakeholder que vai utilizar o programa. Pode ser aplicada qualquer programa, qualquer paradigma de programação; pode ser aplicada todas as fases de teste: unidade, de integração, sistema; para projetar os casos de testes, são aplicados critérios também baseados na especificação do produto.

Alguns exemplos de critérios, a gente tem gráfico de causa e efeito, teste funcional sistemático, análise do valor limite, particionamento classes de equivalência e cada critério desses por tipos diferentes de defeitos, utilizando valores específicos do domínio de entrada.

## Particionamento em Classes de Equivalência

Considerando a técnica de Teste Funcional, um de seus critérios é o particionamento classe de equivalência. Então, de forma geral, todos os critérios da técnica de Teste funcional, são baseados na especificação do produto testado e não na implementação em si. Inclusive considerando

particionamento classe de equivalência.

Então, supondo domínio muito grande, o teste exaustivo é bastante complicado, considerando domínios de entradas infinitos ou muito grandes, nesse contexto. Então o quê que o critério "Particionamento Classes de Equivalência" faz?

Ele divide o domínio de entrada das classes de equivalência, que a gente pode considerar como sendo subconjuntos- então aqui a gente tem exemplificado- são subconjuntos do domínio.

E aí, de acordo com a especificação do programa, esses subconjuntos são definidos e podem ser tratados da mesma maneira. Ou seja, considerando que eu tenha esses subconjuntos, ou seja, essas classes de equivalência, qualquer elemento que eu pegar deveria representar aquele subconjunto do qual ele faz parte.

Esse elemento, ele é na verdade dado de teste que vai compor caso de teste que será submetido ao sistema que está sendo testado. Assim podemos ter uma visão geral da aplicação do critério do "Particionamento Classes de Equivalência".

Basicamente, a partir da especificação do "software", a gente precisa identificar as classes de equivalência, os subconjuntos.

Para ajudar na identificação é importante verificar a especificação do "software" que está sendo testado, e tentar identificar coisas do tipo, por exemplo: intervalo, conjunto e palavras similares.

E se elementos de uma mesma classe estiverem sendo tratados de forma diferente, se você perceber isso, então é importante dividir classes menores ainda, definir classes válidas e classes inválidas.

Considerando isso, o próximo passo é o quê?

Gerar os casos de teste, selecionando elementos de cada classe. Com as classes identificadas, escolhe-se de forma arbitrária, nesse primeiro momento, cada elemento, elementos de cada classe de equivalência, de cada subconjunto, e os casos de testes são então definidos, considerando classes válidas e inválidas.

As diretrizes que precisam ser seguidas para definir as classes, por isso precisa ser identificada. Se a condição de entrada específica intervalo, por exemplo: "Digite número entre e dez!", define-se uma classe válida, ou seja, número entre e dez, e duas classes inválidas, por exemplo, o onze, ou doze, números acima ou números anteriores.

Se a condição especifica uma quantidade, por exemplo: "Digite dois números!", apenas, então, define-se uma classe válida, ou seja, apenas dois números digitados. E duas classes inválidas, por exemplo, um número digitado, ou nenhum número digitado, ou três números digitados, quatro números digitados. Então a ideia é bem simples, né?!

E se a condição especifica conjuntos determinados de valores, está aqui exemplo, relacionado com a tabela do imposto de renda, são classes, são conjuntos específicos. Então, define-se uma classe válida para cada conjunto desses e uma classe inválida geral com outro valor qualquer. Agora, se a condição de entrada específica: "Deve ser assim!", "Deve ser dessa forma!", ou seja, uma coisa bastante objetiva, então define-se uma classe válida, exemplo, "Identificador deve iniciar com uma letra!". Sendo assim, uma classe válida é o quê?

Identificador iniciando com uma letra, uma classe inválida, identificador que não inicia com uma letra, que inicia com número, por exemplo. Então, aqui tem um exemplo, para a gente verificar a aplicação do critério.

É a especificação do programa "Identifier", ou Identificador, é a seguinte:

Variáveis de entrada	Classes de equivalência válidas	Classes de equivalência inválidas
Comprimento ( $t$ )	$1 \leq t \leq 6$ (1)	$t < 1$ (2) e $t > 6$ (3)
Iniciar com uma letra (i)	Sim, inicia com letra (4)	Não inicia com letra (5)
Contém letras ou dígitos (c)	Sim, só contém letras ou dígitos (6)	Contém caracteres diferentes de letras e dígitos (7)

Fonte: Elaborado pelo autor (2023).

Então, considerando as diretrizes que foram apresentadas anteriormente, o quê que a gente pode identificar aqui?

Podemos identificar algumas variáveis de entrada, por exemplo, o comprimento, que precisa ser no mínimo e no máximo seis caracteres, então está aqui, ( $T$ ) está simbolizando a variável comprimento, tem que ser maior ou igual a e menor igual a seis, então essa seria a minha primeira classe de equivalência válida.

Agora considerando a diretriz para quantidade, eu preciso definir o quê? Duas classes inválidas.

Assim temos o número, comprimento, menor do que zero, por exemplo, e comprimento maior do que seis, por exemplo, sete.

Outra coisa que dá para a gente identificar é que tem que iniciar com uma letra, então eu vou chamar isso de i, variável (I). Então, uma classe válida seria: "Sim, inicia com uma letra", e uma classe inválida seria: "Não, não inicia com uma letra", exemplo que não inicia com uma letra.

E por fim, o que mais que a gente tem aqui? "Deve conter apenas letras ou dígitos", então essa variável de entrada poderia ser chamada de (C), por exemplo. Uma classe válida seria "Sim", valor que contém só letras ou dígitos e "Não", contém outros caracteres que não sejam letras ou dígitos.

No segundo passo, considerando as Classes de Equivalências, a gente pode definir os "Casos de Testes".

- $T_0 = \{(a5, Válido), ("", Inválido), (665432197, Inválido), (B*ss1)\}$
- |               |     |        |     |
|---------------|-----|--------|-----|
| (1), (4), (6) | (2) | (3, 5) | (7) |
|---------------|-----|--------|-----|

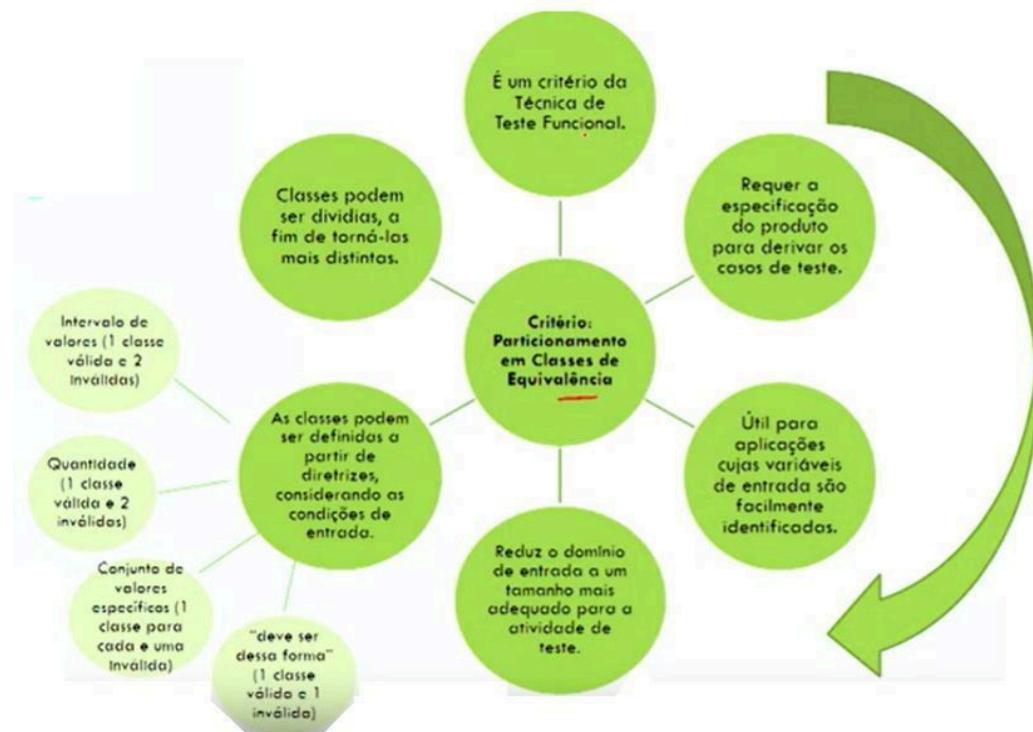
Fonte: Elaborado pelo autor (2023).

Então de forma geral, resumindo o critério de "Particionamento Classes de Equivalências", é critério da técnica de teste funcional, requer a especificação do produto para derivar os casos de testes, ou seja, não requer a implementação sim.

É útil para aplicações cujo as variáveis de entrada são facilmente identificadas, considerando a especificação. Observa que esse critério reduz o domínio de entrada a tamanho mais adequado para a atividade teste.

As classes podem ser definidas a partir de diretrizes, considerando as condições de entrada. A gente identificou algumas, então, por exemplo: "Se a condição de entrada for intervalo de valores, a gente precisa especificar uma classe válida e duas inválidas"; "Se a condição de entrada for uma quantidade, a gente precisa especificar uma classe válida e duas inválidas"; "Se a condição de entrada for conjunto de valores específicos, a gente determina uma classe válida para cada conjunto e uma

"inválida"; e "Se a condição de entrada especificar algo exato, por exemplo: Deve ser assim!, Deve ser dessa forma!, então especificamos uma classe válida e uma classe inválida".



Fonte: Elaborado pelo autor (2023).

## Análise do Valor Limite

Todos os critérios da técnica de teste funcional são baseados na especificação do produto testado e não na implementação sí, considerando a análise do valor limite.

Além disso, o critério análise do valor limite complementa os resultados do particionamento das classes de equivalência.

Tendo domínio de entrada infinito ou muito grande, onde o teste exaustivo é complicado ser realizado, a gente entra com critério de particionamento classes de equivalência e basicamente ele divide o domínio de entrada subconjuntos do domínio, para ficar mais fácil de testar.

E considerando esses conjuntos a gente pode selecionar de forma arbitrária elementos, indicado o subconjunto, para compor casos de teste. Só que o critério de análise de valor limite usa as classes de equivalência e vez dos dados de testes serem escolhidos aleatoriamente, assim como o critério para funcionamento caso de equivalência, eles devem ser selecionados

considerando o valor limitante dessas classes, tanto abaixo quanto acima, é o que a gente vai verificar aqui.

Recomendações gerais com aplicação do critério análise do valor limite:

Se a condição de entrada...	Exemplo	... devem ser definidos dados de teste
(1) especifica um intervalo de valores	Um valor no intervalo entre -1 e +1. 	para os limites desse intervalo e dados de teste imediatamente subsequentes, que explorem classes inválidas vizinhas: <ul style="list-style-type: none"> <li>-1</li> <li>+1</li> <li>-1.001</li> <li>+1.001</li> </ul>

Fonte: Elaborado pelo autor (2023).

Se a condição de entrada especifica intervalo de valores, por exemplo: digite valor no intervalo entre -1 e +1. Então, deve ser definido dados de teste para os limites desse intervalo, então, por exemplo, +1 e -1, e também dados de teste, quer dizer, valores relacionados aos subsequentes, que explorem as classes inválidas vizinhas.

Neste caso seria mais 1.001, a -1.100. Outra questão, se a condição de entrada especifica uma quantidade de valores, por exemplo: digite valor no tamanho de até 255 caracteres. Então, devem ser definidos casos de teste com nenhum valor de entrada, com somente valor de entrada, com 255 valores, e com 256, ou seja, a gente tem aqui limite inferior, limite superior e antes e depois, basicamente essa é a ideia.

Podemos usar as recomendações se entrada ou saída for conjunto ordenado, a gente precisa se preocupar com o primeiro e com o último elemento da lista ordenada. Esse é o exemplo acima citado do programa Identifier, que é identificador. Então, o programa deve determinar se o identificador é válido ou não.

E o que é identificador válido?

Deve começar com uma letra, e conter apenas letras ou dígitos, por exemplo. Aqui seria exemplo de identificador válido.

Além disso deve ter no mínimo caractere e no máximo seis caracteres de comprimento.

Considerando essa especificação, a gente precisa primeiro identificar as classes equivalência considerando o critério particionamento de classe de equivalência. De forma geral a variável de entrada é o comprimento do identificador, ele deve começar com uma letra, e deve conter letras ou dígitos.

Entrada			Saída
t	i	c	
""(nenhum valor) (2) (5)			inválido
a (1) (4) (6)	a (1) (4) (6)	a (1) (4) (6)	válido
a12345 (1) (4) (6)	a12345 (1) (4) (6)	a12345 (1) (4) (6)	válido
a123456 (3)			Inválido
2 (5)			Inválido

Fonte: Elaborado pelo autor (2023).

Aqui estamos chamando de t, de i, e de c. Considerando as variáveis de entrada, as classes de

equivalência válidas e inválidas, ou seja, ele tem que ter o cumprimento entre e seis, tem que iniciar com uma letra, e tem que conter só letras ou dígitos.

A classe de equivalência inválidas, considerando o critério de particionamento classes de equivalência, seria tamanho zero ou menor do que tamanho maior do que 6, por exemplo, sete, não iniciar com letra, e conter caracteres diferentes de letras, por exemplo, asterisco, de entre outros.

Agora podemos aplicar o critério análise do valor limite, considerando os limites e as recomendações para cada critério.



## RESUMO

A atividade de teste de software envolve a utilização de técnicas para estabelecer subdomínios de entrada e identificar casos de teste relevantes. Uma das técnicas utilizadas é o teste funcional, que se baseia na especificação do software e permite verificar se as funções estão corretas. Outra técnica é o particionamento em classes de equivalência, que divide o domínio de entrada em subconjuntos para selecionar casos de teste representativos. Além disso, a análise do valor limite complementa o particionamento das classes de equivalência, testando os limites dos intervalos de valores.

Essas técnicas são fundamentais para lidar com a inviabilidade do teste exaustivo em domínios de entrada muito grandes ou infinitos. Elas podem ser aplicadas em diferentes fases do teste, como unidade, integração e sistema, e permitem projetar casos de teste adequados e com alta cobertura do software. É importante ficar atento às pesquisas recentes na área, pois novos critérios e técnicas estão sendo desenvolvidos para aprimorar o processo de teste de software.



## ATIVIDADE DE FIXAÇÃO

- 1.** Defina o que é o domínio de entrada de um programa e explique sua importância no teste de sistemas.
- 2.** O que é um dado de teste e como ele é utilizado na composição de um caso de teste?
- 3.** Explique o que é um conjunto de casos de teste e por que é essencial submetê-lo ao programa durante a atividade de teste.
- 4.** Por que realizar um teste exaustivo é inviável para grandes ou infinitos domínios de entrada? Apresente uma alternativa para garantir alta probabilidade de revelar defeitos no software.
- 5.** Descreva o que são técnicas de teste e seus critérios, e como são usados para identificar dados de teste nos mesmos subdomínios ou em subdomínios diferentes.

**6.** Dê exemplos de critérios para o teste funcional, o teste estrutural e o teste baseado em defeitos.

**7.** Em quais fases da atividade de teste é possível aplicar o teste funcional, o teste estrutural e o teste baseado em defeitos?

**8.** Como o critério de análise do valor limite complementa os resultados do particionamento em classes de equivalência? Explique como identificar os valores limitantes e apresente exemplos de aplicação dessa técnica.

**9.** Explique a diferença entre o teste funcional, o teste estrutural e o teste baseado em defeitos, em relação ao tipo de informação utilizada para derivar os requisitos de teste.

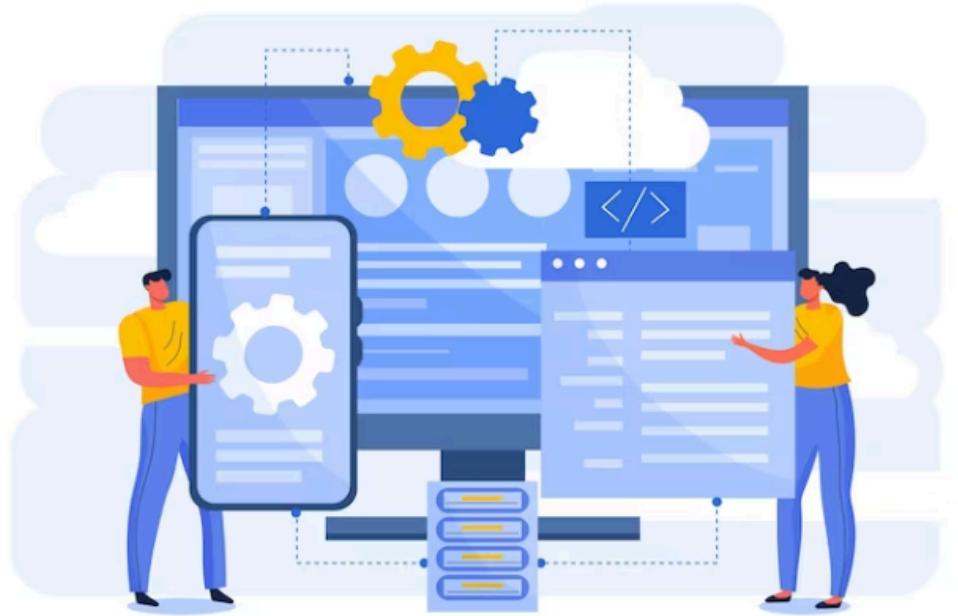
**10.** Descreva a técnica de teste estrutural conhecida como "Teste de Mutação". Explique como ela trabalha com implementações alternativas e como pode ajudar a revelar defeitos típicos do processo de implementação de software.

## TEMA 09

### Teste Estrutural

#### Habilidades:

- Compreender como os testes estruturais funcionam no teste de sistemas.



Disponível em: <[freepik-https://tinyurl.com/4arw43ar](https://tinyurl.com/4arw43ar)>. Acesso em 27 ago. 2023.

Testes estruturais são amplamente difundidos na prática industrial e são encontrados em muitos padrões de desenvolvimento de software. A execução de todas as instruções (instrução e cobertura), todas as filiais (cobertura de filiais) ou todas as condições com os valores lógicos True e False (cobertura de condição) são objetivos de teste comuns. Os métodos de teste estrutural são geralmente aplicados a testes unitários. Não há teste de estrutura obrigatória, critérios para testes de integração ou testes de sistema.

O objetivo de aplicar testes evolutivos a testes estruturais é a geração automática de uma quantidade de dados de teste, que conduz à melhor cobertura possível dos respectivos critérios de teste. No caso de teste de declaração, o objetivo do teste é executar cada instrução do programa pelo menos uma vez. No caso de teste de ramificação, as ramificações vazias também devem ser executadas. Os objetivos do teste são baseados na suposição de que um teste, que não inclui cada instrução e todas as ramificações (do sistema em teste sendo executado) pelo menos uma vez, não apresenta uma verificação completa do objeto de teste. Portanto, o objetivo geral do projeto do caso de teste é definir um conjunto de dados que garantem que cada instrução ou cada ramo é executado.

Considerando que todo o trabalho anterior se concentrou em critérios de teste estrutural

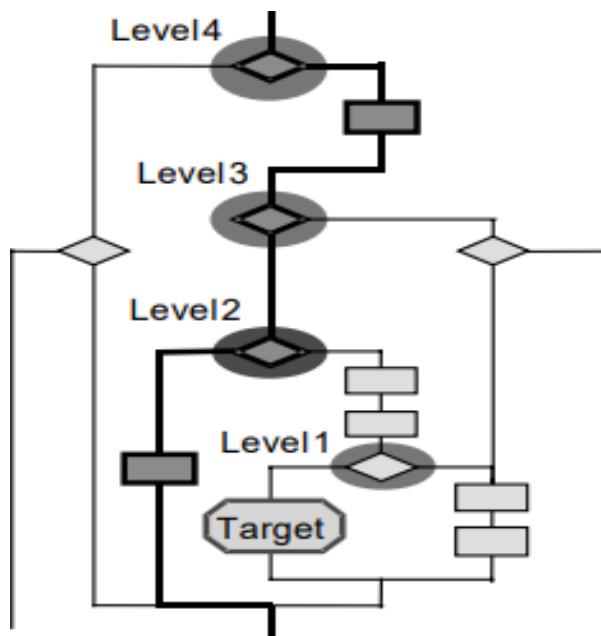
(declaração, ramificação, condição e teste de caminho), nosso ambiente de teste foi desenvolvido para suportar todos os testes orientados a fluxo de controle e fluxo de dados comuns.

Para pesquisar o conjunto de dados de teste, o teste é dividido em objetivos parciais. Cada objetivo parcial representa uma estrutura de programa que requer execução para alcançar cobertura total, por exemplo uma determinada declaração ou ramo. Para cada objetivo parcial, uma função objetivo individual é formulada e uma otimização separada é realizada para procurar um dado de teste executando o objetivo parcial.

A fim de direcionar a busca para estruturas de programa não coberto, a função objetivo calcula uma distância para cada indivíduo que indica o quanto longe está de executar a estrutura de programa desejada. Indivíduos mais próximos para a execução da estrutura de programa desejada serão selecionados como pais e combinados para produzir indivíduos descendentes.

As funções objetivo dos objetivos parciais consistem em duas componentes – o nível de aproximação e o cálculo da distância local.

## Cálculo do nível de aproximação



Fonte: Elaborado pelo autor (2023).

O nível de aproximação fornece um valor para um indivíduo que dá o número de nós de ramificação entre estruturas de programa cobertas pelo indivíduo e a estrutura do programa. Para este cálculo, são considerados apenas os nós de ramificação que contêm uma saída borda que resulta em uma perda da estrutura de programa desejada.

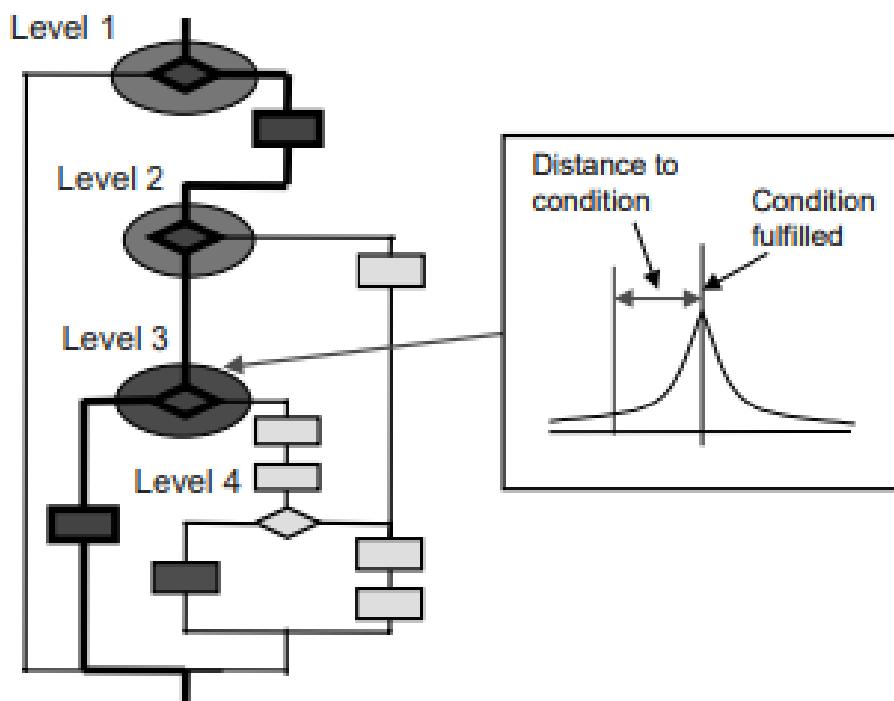
Um exemplo é dado na figura acima. O gráfico do programa contém quatro nós de ramificação que podem resultar na perda do nó de destino (representando a instrução ou ramificação desejada).

Cada nó é atribuído ao nível de aproximação correspondente (nível 4 ao nível 1). Um indivíduo que se afasta do nó de destino no primeiro nó de ramificação atinge um menor nível de

aproximação (nível 4) do que um indivíduo que atinge o nível 3 etc. A figura mostra a execução de um indivíduo que perde o nó de destino na ramificação nível com o nível de aproximação 2. O indivíduo passa os nós de ramificação nos níveis 4 e 3 conforme desejado, mas perde o nó de destino no nível 2.

## Cálculo de distância local

O cálculo da distância local é realizado para distinguir diferentes indivíduos executando o mesmo programa caminho. Para isso, uma distância para a execução do outro caminho do programa é calculada para o indivíduo por meio de condições de ramificação no nó de ramificação em que o nó de destino é perdido. A Figura abaixo ilustra esse cálculo.

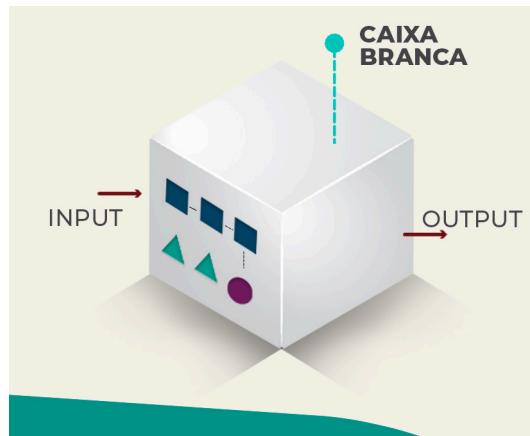


Fonte: Elaborado pelo autor (2023).

Por exemplo, se uma condição de ramificação  $x==y$  precisa ser avaliado como True para alcançar o nó de destino, então o objetivo função pode ser definida como  $|x-y|$ . Se um indivíduo obtiver a distância local 0, é encontrado um dado de teste que cumpre a condição de ramificação:  $x$  e  $y$  têm o mesmo valor.

Se um nó de ramificação contiver várias condições, a distância local é uma combinação das distâncias locais de cada condição. Para um nó do tipo  $a \vee b$  a distância local de um individual é obtida a partir do valor mínimo para cada predicado único  $a$  e  $b$ . No caso de  $a \wedge b$  a distância local de um indivíduo é o resultado da soma das distâncias determinado para cada predicado único.

## Teste de Caixa Branca



Disponível em: <[freepik-https://tinyurl.com/4arw43ar](https://tinyurl.com/4arw43ar)>. Acesso em 27 ago. 2023.

O teste de caixa branca é altamente eficaz na detecção e resolução de problemas, porque os bugs podem ser encontrados antes de eles causarem problemas. O teste de caixa branca é o processo de dando a entrada para o sistema e verificando como o sistema processa os processos que entram para gerar a saída necessária. Caixa branca teste também é chamado de análise de caixa branca, teste de caixa clara ou análise de caixa clara. O teste de caixa branca é aplicável em integração, níveis de unidade e sistema do teste de software processo. O teste de caixa branca é considerado uma segurança, método de teste que pode ser usado para validar se o código de implementação segue o design pretendido, para validar funcionalidade de segurança implementada e para descobrir vulnerabilidades exploráveis.

Alguns tipos diferentes de técnicas de teste de caixa branca são:

- Teste de caminho básico
- Teste de Loop
- Teste de estrutura de controle

Vantagens do teste de caixa branca:

- Todos os caminhos independentes em um módulo serão exercidos em ao menos uma vez.
- Todas as decisões lógicas serão exercidas.
- Todos os loops em seus limites serão executados.
- Estruturas de dados internas serão exercitadas para manter sua validade.
- Erros em códigos ocultos são revelados.
- Aproxime o particionamento feito por execução equivalência.
- O desenvolvedor cuidadosamente explica a implementação.

Desvantagens do teste de caixa branca:

- Perde os casos omitidos no código
- Como o conhecimento do código e da estrutura interna é um pré-requisito, um testador qualificado é necessário para realizar este tipo de teste, o que aumenta o custo.
- E é quase impossível examinar cada pedaço de código para descobrir erros ocultos, que podem criar problemas, resultando na falha do aplicativo.

O teste de caixa branca é altamente eficaz em detectar e resolver problemas, porque os bugs (bug ou falha é uma manifestação de um erro em um Software), muitas vezes pode ser encontrado antes eles causam problemas. Podemos definir brevemente isso como testar software com o conhecimento da estrutura interna e codificação dentro do programa. O teste de caixa branca também é chamado de análise de caixa branca, teste de caixa clara ou análise de caixa clara.

É uma estratégia de software depuração (é o processo de localizar e corrigir bugs no código do programa de computador ou a engenharia de um dispositivo de hardware, veja em qual o testador tem excelente conhecimento de como os componentes do programa interagem.

Este método pode ser usado para aplicativos de serviços da Web e raramente é prático para depuração em grandes sistemas e redes. Além disso, em teste de caixa branca é considerado como um teste de segurança (o processo para determinar que uma informação sistema protege os dados e mantém a funcionalidade como pretendido), método que pode ser usado para validar se a implementação do código segue projeto pretendido, para validar implementado funcionalidade de segurança e para descobrir vulnerabilidades.



## RESUMO

Os testes estruturais desempenham um papel fundamental no processo de teste de sistemas, visando garantir a integridade e confiabilidade do software. Essa abordagem concentra-se na avaliação direta da estrutura interna do código-fonte de um sistema, revelando possíveis falhas e vulnerabilidades. Ao compreender como os testes estruturais funcionam, os profissionais de teste podem identificar lacunas no código, como trechos não executados ou caminhos lógicos não testados.

Os testes estruturais operam por meio da análise do fluxo de controle e dados dentro de um programa. Isso envolve a criação de casos de teste que abrangem diferentes caminhos de execução, permitindo uma avaliação abrangente do código. Ao examinar as estruturas internas do software, os testadores podem revelar erros de programação, como loops infinitos, condições de corrida e erros de lógica.

Um dos métodos mais comuns de teste estrutural é o teste de cobertura, que mede a extensão em que o código foi executado durante os testes. A cobertura de código é uma métrica valiosa para avaliar a abrangência dos testes e identificar áreas não testadas. Além disso, os testes estruturais também podem envolver técnicas como o teste de caminho, onde todos os caminhos possíveis de execução são testados individualmente.

Em resumo, as habilidades necessárias para compreender os testes estruturais são cruciais para garantir a qualidade do software. Ao explorar as nuances do funcionamento desses testes, os profissionais de teste podem identificar e mitigar problemas de código antes que eles se transformem em problemas maiores. Compreender os princípios por trás dos testes estruturais capacita os testadores a desenvolver casos de teste mais eficazes e a contribuir para a construção de sistemas robustos e confiáveis.



## ATIVIDADE DE FIXAÇÃO

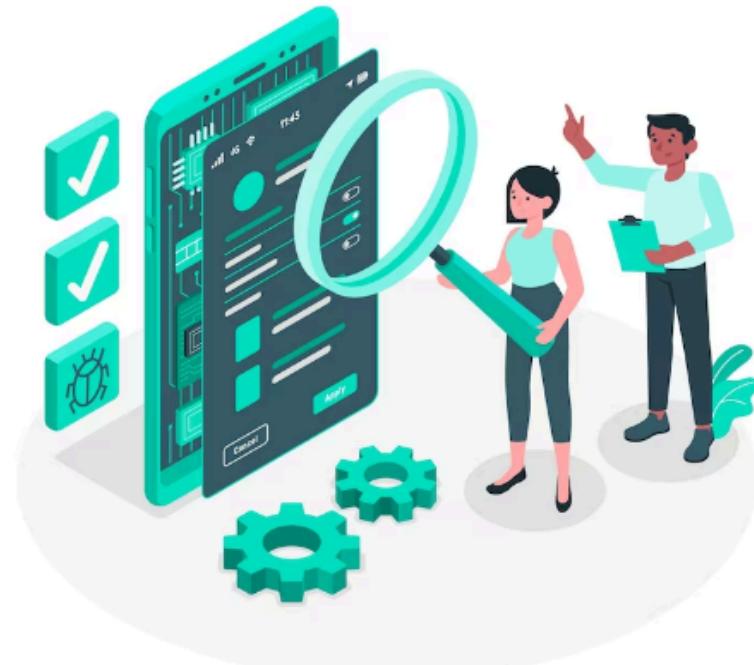
1. Quais são os objetivos de teste comuns dos testes estruturais?.
2. que é cobertura de condição em testes estruturais?
3. Qual é o objetivo do teste de declaração?
4. Por que os testes evolutivos são aplicados aos testes estruturais?
5. O que é um objetivo parcial em testes estruturais?
6. Qual é o objetivo do teste de caixa branca?
7. Como as funções objetivo dos objetivos parciais são compostas em testes estruturais?
8. Quais são as vantagens do teste de caixa branca? Dê exemplos para cada uma delas.
9. Suponha que você esteja realizando um teste de caixa branca em um módulo de software complexo. Descreva como você abordaria o teste de caminho básico e o teste de loop para garantir a cobertura abrangente.
10. Imagine que você é um testador qualificado encarregado de realizar testes estruturais em um sistema de grande escala. Como você lidaria com os casos omitidos no código e os erros ocultos? Desenvolva uma estratégia para lidar com essas situações desafiadoras.

## TEMA 10

### Diagnóstico

#### Habilidades:

- Compreender a importância do processo de teste de software para encontrar e corrigir bugs durante o desenvolvimento de sistemas.



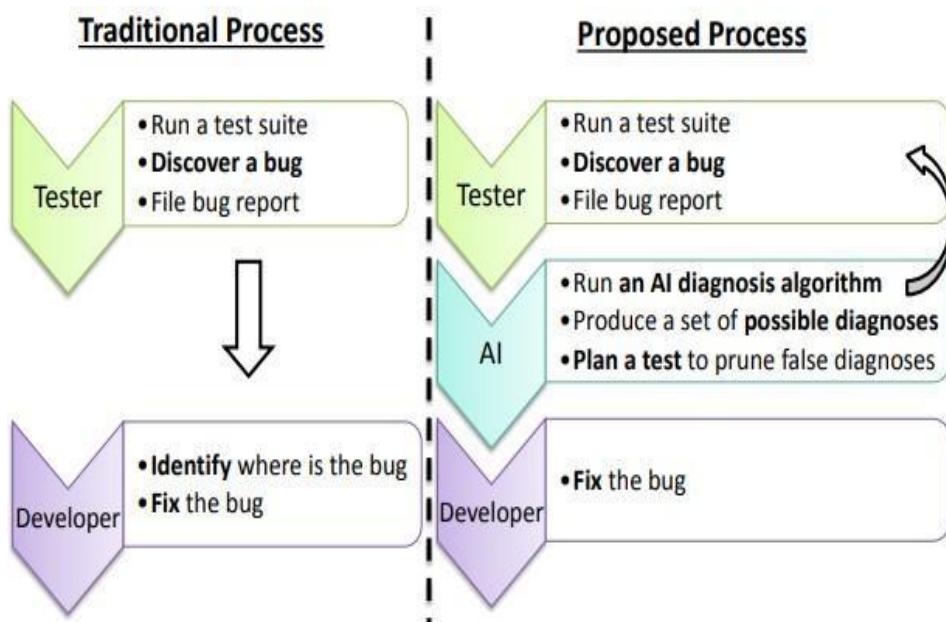
Disponível em: <[storyset-https://tinyurl.com/bdzyvzrs](https://tinyurl.com/bdzyvzrs)>. Acesso em 27 ago. 2023.

O teste é uma parte muito importante quando se está criando um software (Myers et al. 2004). Na fase de teste de software, a ideia é descobrir problemas, também chamados de bugs, e consertá-los. Para o programador, consertar esses bugs geralmente envolve duas coisas. Primeiro, é preciso descobrir a causa principal do problema e, depois, arrumar as partes do software que estão com defeito, como as funções ou classes. Encontrar a razão de um bug no software é difícil e geralmente requer tentar várias coisas diferentes até encontrar a solução certa. Isso significa que o programador vai fazer testes e investigar várias possibilidades até achar o que está errado. É difícil porque muitas vezes não é fácil repetir ou copiar o bug que foi encontrado por outra pessoa.

Uma forma ideal de lidar com isso seria se a pessoa que está testando, quando encontrar um bug, fizesse testes extras para ajudar o programador a descobrir a parte do software que está causando o bug. No entanto, planejar esses testes extras não é simples, especialmente se a pessoa

que está testando não conhece bem o código do software. Na maioria das vezes, o teste é feito por especialistas em garantir a qualidade (chamados QA), que não estão familiarizados com o código do software que estão testando. Essa separação entre quem escreve o código e quem testa é considerada uma boa prática, pois ajuda a melhorar o teste.

Quando falamos de testador, nos referimos à pessoa que está fazendo os testes, e quando falamos de desenvolvedor, nos referimos à pessoa que está criando o software. O objetivo do processo tradicional de teste de software (que chamamos simplesmente de teste) é verificar se o sistema funciona bem. Podemos pensar nos testes como uma forma de comunicação entre o testador e o desenvolvedor, mostrada na parte abaixo da Figura abaixo.



Fonte: Elaborado pelo autor (2023).

O pessoal que testa um programa faz uma série de testes para verificar se ele funciona corretamente. Essa série de testes é chamada de conjunto de testes. O testador executa todos esses testes no conjunto até que todos sejam feitos e deem certo, ou até que um deles dê errado. Se algum teste der errado, significa que tem um problema, um bug. O testador reporta o teste que falhou para quem criou o programa, chamado de desenvolvedor. Isso normalmente é feito por meio de um "relatório de bug". Enquanto isso, o testador continua fazendo outros testes. O desenvolvedor é responsável por consertar os bugs que o testador encontrou.

Para fazer isso, muitas vezes se usam ferramentas que ajudam a rastrear e resolver esses problemas (por exemplo, HP Quality Center, Bugzilla e IBM Rational ClearQuest). Para consertar um bug, o desenvolvedor precisa descobrir qual parte do programa está com defeito e, então, fazer a correção. Esse processo, de encontrar e consertar a parte defeituosa, é chamado de "depuração".

Entender como os testes funcionam e o que eles mostram é um pouco complicado. A maioria dos estudos sobre testes se concentra em ver quão preciso eles são em detectar problemas. No entanto, a precisão dos testes pode variar muito entre os estudos, porque eles podem usar critérios diferentes para dizer se algo está certo ou errado.

Também é importante perceber que ter um teste super preciso nem sempre significa que o resultado final para as pessoas vai melhorar. Por isso, é importante fazer análises detalhadas para entender como os testes podem ajudar realmente.

Nos anos 1990, um grupo de pesquisadores começou a pensar em formas de revisar e analisar os testes. Eles criaram um lugar chamado Colaboração Cochrane para reunir todas essas informações. Demorou um tempo para a colaboração decidir como fazer isso. Finalmente, depois de muitos anos, eles concordaram em incluir revisões de testes diagnósticos na biblioteca Cochrane.

Para fazer isso, eles formaram um grupo de pessoas que trabalhavam em métodos e planejaram como as revisões seriam feitas. Eles se encontraram e discutiram maneiras de analisar os testes e chegaram a um acordo sobre como fazer isso.

Com o tempo, eles desenvolveram um software e começaram a registrar as revisões de testes na biblioteca. Foi um processo complicado, mas em 2008 eles conseguiram publicar a primeira revisão de teste diagnóstico na biblioteca Cochrane.

Durante todo esse processo, eles também ajudaram outras pessoas a aprender sobre como fazer essas revisões e como entender os resultados dos testes.

## Metodologia inicial

Nos anos finais da década de 1980 e começo dos anos 1990, começaram a ser feitas análises que juntavam informações de vários estudos sobre testes diagnósticos. Isso foi feito de forma parecida com o que já se fazia para análises de intervenções médicas: coletando e escolhendo estudos, avaliando a qualidade deles, resumindo os resultados em um só lugar (meta-análise) e tirando conclusões a partir disso.

No entanto, fazer uma análise de precisão de testes diagnósticos era mais complicado. Isso porque as medidas de precisão dos testes geralmente vêm em pares: sensibilidade e especificidade; valores preditivos positivos e negativos; e razão de chances positiva e negativa. E essas medidas de precisão mudam dependendo do ponto de corte usado para dizer se o teste é positivo ou negativo.

Um método que ajudou a lidar com isso foi desenvolvido por Moses, Littenberg e colegas. Esse método levou em conta essas variações nos pontos de corte e foi mais fácil de usar. Ele se tornou muito popular para análises posteriores.

Fazer essas análises não envolve apenas a parte estatística. Até mesmo formular a pergunta certa para a análise pode ser complicado, já que a precisão de um teste pode ser diferente em situações diferentes. O jeito como o estudo é feito também pode afetar a precisão estimada, e não existe um "melhor" jeito de fazer isso, como existe no caso dos estudos que comparam intervenções médicas.

Além disso, não existe uma linguagem padrão para descrever os diferentes tipos de estudos usados para avaliar a precisão dos testes. Isso torna mais difícil encontrar os estudos relevantes em bancos de dados e escolher quais usar na análise.

Quando olhamos para as conclusões dessas análises, é importante entender que um teste muito preciso por si só não garante uma melhora nos resultados para o paciente. O que realmente importa é como o médico lida com os resultados do teste e toma decisões com base neles.

Essas questões, tanto as estatísticas quanto as relacionadas com a forma como os estudos são feitos e interpretados, continuam sendo relevantes e importantes mesmo após quase 20 anos desde

esses avanços terem sido feitos

## Formulação de perguntas e interpretação dos resultados

Foi percebido que o valor dos testes diagnósticos depende muito de como eles são usados em diferentes situações na prática médica. Isso também afeta como entendemos e aplicamos as descobertas dos estudos: essas descobertas valem para todos os casos ou as situações diferentes podem afetar os resultados do teste?

Por exemplo, um questionário usado para identificar se pessoas idosas estão desenvolvendo demência pode ser útil em geral. No entanto, se esse questionário for usado em um lugar onde os pacientes já têm vários sintomas em comum, ele pode não conseguir distinguir entre alguém com problemas cognitivos gerais e alguém com demência.

Mesmo que o questionário funcione bem para diferenciar entre essas condições, seu valor pode depender de outras coisas, como se saber se alguém tem demência, ao invés de problemas cognitivos, vai afetar a forma como essa pessoa é tratada e como ela vive.

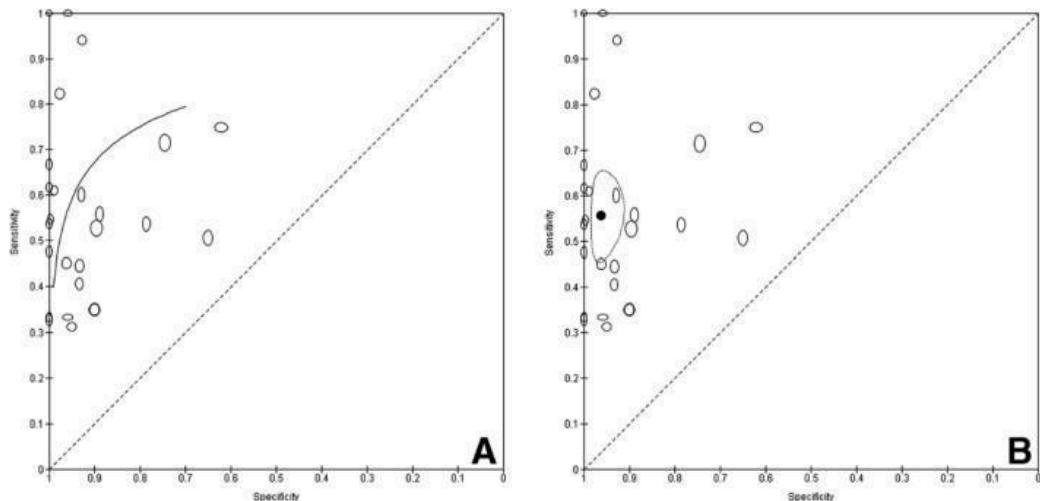
Quando fazemos revisões de estudos sobre testes diagnósticos, também precisamos entender que esses testes não são usados sozinhos, mas sim em comparação com outros. Por isso, essas revisões do Cochrane se focam em entender o valor relativo dos testes, ou seja, o quanto eles são melhores ou piores em comparação com outras opções.

Encontrar os estudos certos para fazer essas revisões é mais difícil do que parece. Diferentes tipos de estudos são feitos para avaliar a precisão dos testes, e não existe uma palavra específica que ajude a encontrar todos eles. Os pesquisadores precisam testar diferentes combinações de palavras para encontrar os estudos que são relevantes.

Depois de achar os estudos, é importante avaliar a qualidade deles. Uma ferramenta chamada "QUADAS" foi criada para isso. Ela tem 14 itens que ajudam a ver se os estudos foram feitos bem e se as conclusões deles são confiáveis. Com o tempo, essa ferramenta foi melhorada para ser ainda mais precisa.

Para fazer análises estatísticas desses estudos, foram desenvolvidos métodos específicos. Alguns modelos são mais simples e outros mais complexos. Esses modelos ajudam a juntar as informações dos diferentes estudos para termos uma ideia geral da precisão do teste.

Em resumo, entender os testes diagnósticos é um processo complicado. Precisamos considerar como eles são usados na prática médica, encontrar os estudos certos, avaliar a qualidade deles e usar métodos estatísticos adequados para tirar conclusões confiáveis sobre a precisão dos testes.



Fonte: Elaborado pelo autor (2023).

## RESUMO

O processo de testar um software é crucial para encontrar e consertar erros durante o desenvolvimento de sistemas. Quando temos que corrigir esses erros, muitas vezes precisamos descobrir o motivo principal e arrumar as partes com defeito do programa. Descobrir a razão de um erro pode ser complicado, então é importante que as pessoas que testam e as que desenvolvem o software trabalhem juntas. É recomendado que quem cria o código seja diferente de quem testa, o que torna os testes mais eficazes.

No processo de teste, os testadores executam um conjunto de testes para verificar se o sistema funciona bem. Se algum teste não der certo, isso significa que tem um erro, e aí os desenvolvedores precisam consertá-lo. Fazer análises de precisão de testes diagnósticos foi um desafio, mas conseguimos melhorar a qualidade dos estudos e usar métodos mais complexos para analisar a sensibilidade e especificidade dos testes.

Procurar estudos sobre a precisão dos testes diagnósticos é complicado, e para fazer isso é importante entender onde esses testes podem ser aplicados. Avaliar a qualidade dos estudos é feito usando uma ferramenta chamada QUADAS, que é um padrão para medir a qualidade dos estudos. Nas análises estatísticas que combinam os resultados de vários estudos, usamos métodos rigorosos para calcular estimativas resumidas de como esses testes diagnósticos funcionam em geral.

## ATIVIDADE DE FIXAÇÃO

1. Qual é o propósito do processo de teste de software?
2. Por que diagnosticar a causa raiz de um bug de software pode ser desafiador?

3. O que é uma sequência de testes executada pelo testador chamada?
4. Quais são as duas tarefas que geralmente envolvem a correção de bugs no desenvolvimento de software?
5. O que indica um teste com falha durante o processo de teste de software?
6. Por que a separação entre quem escreve o código e quem o teste é considerada uma prática recomendada?
7. Explique o processo de colaboração entre testadores e desenvolvedores durante o teste de software.
8. Como o desenvolvedor corrige os bugs encontrados pelo testador?
9. Por que a busca por estudos de acurácia diagnóstica é mais difícil do que a busca por ensaios de intervenção?
10. Quais são as implicações de um teste altamente preciso para o bem-estar do paciente durante o processo de teste de diagnóstico?



## REFERÊNCIAS

ADVE, Sarita and BOEHM, Hans-J. **Memory models: a case for rethinking parallel languages and hardware.** In: Communications of the ACM. 2010, vol. 53, no 8, p. 90-101.

ALFONSO, Pedro L. **Revisión de modelos para evaluar la calidad de productos web. Experimentación en portales bancarios del NEA,** M.S. tesis, Dep: facultad de Informática, Universidad Nacional de La Plata, Argentina, 2012.

AL-QUTAISH, Rafa E. **Quality models in software engineering literature: an analytical and comparative study.** In: Journal of American Science. 2010. vol. 6, no.3, p.166-175

ALVAREZ CHIRIBOGA, Daniel Alejandro, y ORTEGA NAVARRETE, Fernando Enrique. **Plan de implantación de nivel de madurez CMMI 3 para una empresa de desarrollo de software ecuatoriana.** Facultad De Ingeniería, Escuela De Sistemas, Escuela Politécnica Nacional, Ecuador, 2009, p.156.

ANGO HERRERA, Luis Fernando. **“Evaluación de Sistemas”**, Tesis, Ibarra, Pontifica Universidad Católica del Ecuador, 2014, 19p.

B. L. Roció Gabriela. **Bootstrap, Estándares y modelos de calidad aplicados al software,** Conferencia, Tepic, Nayarit, 2015.

BAPTISTA, P. T. **“MODELOS DE CALIDAD MC CALL, ISO/IEC 9126, ISO 25000”.** In Gestión Operativa de la Calidad en el Software, Ingeniería de Sistemas de la Universidad Autónoma Gabriel René Moreno, 2012.

BOEHM, B. W. **Software Risk Management: principles and practices**, IEEE Software, Volume 8. No1. p 32-40, 1991.

CATOTA TOCA, Ximena Alexandra. **“Análisis, diseño e implementación de un sistema informático para gestionar los trabajos de Help Desk del área de tecnología de la Cooperativa Codesarrollo basado en Itil y Silverlight.”** Tesis de Ingeniería de Sistemas. Quito, 2015.

CHAPMAN, C. e WARD, S. **Project Risk Management: Processes, Techniques and Insights.** John Wiley & Sons. p 30-41, 1997.

CHARETTE, R. **Application strategies for risk analysis.** New York: MultiScience Press. p 17-21, 1990.

FAIRLEY, R. **Risk Management For Software’s Projects.** IEEE Software. p 54-66. HALL, E. M. 1998. Managing Risk. 2a Ed. USA: Addison Wesley. p 88-103, 1994.

EESO/IEC 12207, **Information Technology – Software Life-Cycle Processes**, International Standard Organization, Geneve, 1995.

KOSCIANK,, Andre; SOARES, Michel S. **Qualidade de Software: Aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. 2ª Edição. São Paulo: Novatec Editora, 2007.

KRUCHTEN, P. **Introdução ao Rup: Rational Unified Process**. 2a Ed. Ciência Moderna. São Paulo. p 25-36, 1994.

MARKKULA, M. **Knowledge Management in Software Engineering Projects**. In: Proc. of the 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, 1999.

MIAN, P. G., Natali, A. C. C., and Falbo, R. A. **Ambientes de Desenvolvimento de Software e o Projeto ADS**. Revista Engenharia Ciência Tecnologia, Vitória, ES, v. 04, n. 04, p. 3-10, 2001.

MOREIRA, M. **A questão da produção e da avaliação do software educacional**. In: Seminário o Computador e a Realidade Educacional Brasileira, 2. Belo Horizonte, UFMG/Centro Piloto de Informática na Educação, mai 1987.

PAULK, M. C & Weber, C. V & Curtis, B. & Chrassis, M. B. **The Capability Maturity Model: Guidelines for Improving the Software Process**. Carnegie Mellon University, Software Engineering Institute, Addison- Wesley Longman Inc, 1997.

SELLEN, A e NICOL, A. **Building user-centered on-line help**. In “**The art for human computer interface design**” edited by Brenda Laurel. Addison-Wesley Publishing Company. Massachusetts, 1990.

SOFTWARE ENGINEERING INSTITUTE, 2001. **CMMI - Capability Maturity Model Integration version 1.1** Pittsburgh, PA. Software Engineering Institute, Carnegie Mellon University. USA.

STAHL, M. **Software educacional: características dos tipos básicos**. In: Anais do Simpósio Brasileiro de Informática na Educação 1: 34-46. Rio de Janeiro, nov 1990.

WEBER, K. C, **Qualidade e Produtividade em Software**. 3 ed. São Paulo: Makron Books do Brasil Ltda, 1999.

