
# Certified Tester

## Foundation Level Syllabus

CTFL

2018br

3.1

---


Brazilian Software Testing Qualifications Board

Tradução realizada pelo WG-Traduções do BSTQB do syllabus do ISTQB:

***Certified Tester Founbdation Level Syllabus* – 2018 3.1**

# Direitos autorais

# Histórico da Revisão

| Version | Date | Remarks |
|---|---|---|
| ISTQB 2018 3.1 | 11/11/2019 | Minors update |
| ISTQB 2018 | 27/04/2018 | Candidate general release version |
| ISTQB 2018 | 12/02/2018 | Candidate beta version |
| ISTQB 2018 | 19/01/2018 | Cross-review internal version 3.0. |
| ISTQB 2018 | 15/01/2018 | Pre-cross-review internal version 2.9, incorporating Core Team edits. |
| ISTQB 2018 | 9/12/2017 | Alpha review 2.5 – Technical edit of 2.0, no new content added |
| ISTQB 2018 | 22/11/2017 | Alpha review 2.0 – Certified Tester Foundation Level Syllabus Major Update 2018 – see Appendix C – Release Notes for details |
| ISTQB 2018 | 12/06/2017 | Alpha review release - Certified Tester Foundation Level Syllabus Major Update 2018 – see Appendix C – Release Notes |
| ISTQB 2011 | 1/04/2011 | Certified Tester Foundation Level Syllabus Maintenance Release – see Release Notes |
| ISTQB 2010 | 30/03/2010 | Certified Tester Foundation Level Syllabus Maintenance Release – see Release Notes |
| ISTQB 2007 | 01/05/2007 | Certified Tester Foundation Level Syllabus Maintenance Release |
| ISTQB 2005 | 01/06/2005 | Certified Tester Foundation Level Syllabus |
| ASQF V2.2 | 06/2003 | ASQF Syllabus Foundation Level Version 2.2 “Lehrplan Grundlagen des Software-testens“ |
| ISEB V2.0 | 25/02/1999 | ISEB Software Testing Foundation Syllabus V2.0 |

# Agradecimentos

Este documento foi formalmente divulgado pela Assembléia Geral do *ISTQB®* (11 de novembro de 2019).

Esse documento foi produzido pelo time do International Software Testing Qualifications Board: Klaus Olsen (chair), Meile Posthuma and Stephanie Ulrich. Essa equipe agradece aos Conselhos Membros pelos comentários de revisãorelacionados ao Foundation Syllabus 2018

A versão 2018 foi produzida por uma equipe do International Software Testing Qualifications Board: Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Debra Friedenberg, Judy McKay, Meile Posthuma, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, and Eshraka Zakaria.

O time agradece Rex Black e Dorothy Graham por sua edição técnica e a equipe de revisão e revisão cruzada e as diretorias de membros por suas sugestões e contribuições.

As seguintes pessoas participaram da revisão, comentando e votando deste syllabus:

Tom Adams, Tobias Ahlgren, Xu Aiguo, Chris Van Bael, Katalin Balla, Graham Bath, Gualtiero Bazzana, Arne Becher, Veronica Belcher, Lars Hilmar Bjørstrup, Ralf Bongard, Armin Born, Robert Bornelind, Mette Bruhn-Pedersen, Geza Bujdoso, Earl Burba, Filipe Carlos, Young Jae Choi, Greg Collina, Alessandro Collino, Cui Zhe, Taz Daughtrey, Matthias Daigl, Wim Decoutere, Frans Dijkman, Klaudia Dussa-Zieger, Yonit Elbaz, Ofer Feldman, Mark Fewster, Florian Fieber, David Frei, Debra Friedenberg, Conrad Fujimoto, Pooja Gautam, Thorsten Geiselhart, Chen Geng, Christian Alexander Graf, Dorothy Graham, Michel Grandjean, Richard Green, Attila Gyuri, Jon Hagar, Kobi Halperin, Matthias Hamburg, Zsolt Hargitai, Satoshi Hasegawa, Berit Hatten, Wang Hongwei, Tamás Horváth, Leanne Howard, Chinthaka Indikadahena, J. Jayapradeep, Kari Kakkonen, Gábor Kapros, Beata Karpinska, Karl Kemminger, Kwanho Kim, Seonjoon Kim, Cecilia Kjellman, Johan Klintin, Corne Kruger, Gerard Kruijff, Peter Kunit, Hyeyong Kwon, Bruno Legeard, Thomas Letzkus, Alon Linetzki, Balder Lingegård, Tilo Linz, Hongbiao Liu, Claire Lohr, Ine Lutterman, Marek Majernik, Rik Marselis, Romanos Matthaios, Judy McKay, Fergus McLachlan, Dénes Medzihradszky, Stefan Merkel, Armin Metzger, Don Mills, Gary Mogyorodi, Ninna Morin, Ingvar Nordström, Adam Novak, Avi Ofer, Magnus C Ohlsson, Joel Oliviera, Monika Stocklein Olsen, Kenji Onishi, Francisca Cano Ortiz, Gitte Ottosen, Tuula Pääkkönen, Ana Paiva, Tal Pe'er, Helmut Pichler, Michaël Pilaeten, Horst Pohlmann, Andrew Pollner, Meile Posthuma, Vitalijs Puiso, Salvatore Reale, Stuart Reid, Ralf Reissing, Shark Ren, Miroslav Renda, Randy Rice, Adam Roman, Jan Sabak, Hans Schaefer, Ina Schieferdecker, Franz Schiller, Jianxiong Shen, Klaus Skafte, Mike Smith, Cristina Sobrero, Marco Sogliani, Murian Song, Emilio Soresi, Helder Sousa, Michael Sowers, Michael Stahl, Lucjan Stapp, Li Suyuan, Toby Thompson, Steve Toms, Sagi Traybel, Sabine Uhde, Stephanie Ulrich, Philippos Vakalakis, Erik van Veenendaal, Marianne Vesterdal, Ernst von During, Salinda Wickramasinghe, Marie Walsh, Søren Wassard, Hans Weiberg, Paul Weymouth, Hyungjin Yoon, John Young, Surong Yuan, Ester Zabar, e Karolina Zmitrowicz.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2018): Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Dani Almog, Debra

Friedenberg, Rashed Karim, Johan Klintin, Vipul Kocher, Corne Kruger, Sunny Kwon, Judy McKay, Thomas Muller, Igal Levi, Ebbe Munk, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Hans Schaefer, Radoslaw Smilgin, Mike Smith, Steve Toms, Stephanie Ulrich, Marie Walsh, Eshraka Zakaria, and Stevan Zivanovic. A equipe principal agradece à equipe de revisão e a todos os membros da Diretoria por suas sugestões.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2011): Thomas Muller (chair), Debra Friedenberg. A equipe principal agracede a equipe de revisão (Dan Almog, Armin Beer, Rex Black, Julie Gardiner, Judy McKay, Tuula Pääkkönen, Eric Riou du Cosquier Hans Schaefer, Stephanie Ulrich, Erik van Veenendaal), e todos os Conselhos Nacionais por suas sugestões.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2010): Thomas Muller (chair), Rahul Verma, Martin Klonk and Armin Beer. The core team thanks the review team (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Judy McKay, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veenendaal), e todos os Conselhos Nacionais por suas sugestões.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2007): Thomas Muller (chair), Dorothy Graham, Debra Friedenberg, and Erik van Veenendaal. The core team thanks the review team (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, e Wonil Kwon) e todos os Conselhos Nacionais por suas sugestões.

International Software Testing Qualifications Board Working Group Foundation Level (Edition 2005): Thomas Muller (chair), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson e Erik van Veenendaal e todos os Conselhos Nacionais por suas sugestões.

# Índice

# 0 Introdução

## 0.1 Objetivo deste syllabus

Este syllabus forma a base para a *ISTQB® CTFL Foundation Level*. O *ISTQB®* fornece este syllabus da seguinte forma:

1. Aos Conselhos Nacionais, para traduzirem ao seu idioma local e credenciar os provedores de treinamento. Os Conselhos Nacionais podem adaptar o programa às suas necessidades linguísticas específicas e modificar as referências para se adaptarem às suas publicações locais.
2. Aos Conselhos de Exame, para derivar questões de exame em sua língua local com base nos objetivos de aprendizagem para cada módulo.
3. Aos Provedores de Treinamento, para que produzam o material didático e determinem métodos apropriados de ensino.
4. Aos Candidatos à Certificação, como uma fonte para se prepararem para o exame.
5. À Comunidade Internacional de Software e à Engenharia de Sistemas, para avançar a profissão de software e testes de sistema, e servir como base para livros e artigos.

O *ISTQB®* permite que outras entidades utilizem este *syllabus* para outros fins, desde que procurem e obtenham permissão prévia por escrito do *ISTQB®*.

## 0.2 A Certificação fundamental em teste de software

A certificação *CTFL Foundation Level* é destinada a qualquer pessoa envolvida em testes de software. Isso inclui pessoas em funções como testadores, analistas de teste, engenheiros de teste, consultores de teste, gerentes de teste, testadores de aceite de usuários e desenvolvedores de software. Este nível de certificação também é apropriado para quem quer um conhecimento básico de teste de software, como proprietários de produto, gerentes de projeto, gerentes de qualidade, gerentes de desenvolvimento de software, analistas de negócios, diretores de TI e consultores de gerenciamento. Os detentores do certificado no nível fundamental poderão passar para qualificações de teste de software de nível superior.

O *ISTQB® Foundation Level Overview 2018* [ISTQB_FL_OVER] é um documento separado que inclui as seguintes informações:

- Resultados de negócios para o syllabus;
- Matriz mostrando rastreabilidade entre resultados de negócios e objetivos de aprendizado;
- Resumo deste syllabus.

## 0.3 Objetivos de aprendizagem e níveis cognitivos

Os objetivos de aprendizagem apoiam os resultados de negócios e são usados para criar os exames do *CTFL Foundation Level*.

Em geral, todo o conteúdo deste *syllabus* é classificável no nível K1, com exceção da Introdução e Apêndices. Ou seja, o candidato pode ser solicitado a reconhecer, lembrar ou recordar uma palavra-chave ou conceito mencionado em qualquer um dos seis capítulos. Os níveis de conhecimento dos objetivos de aprendizagem são mostrados no início de cada capítulo e classificados da seguinte forma:

- K1: lembrar
- K2: entender
- K3: aplicar

Mais detalhes e exemplos de objetivos de aprendizado são fornecidos no Apêndice B.

As definições de todos os termos listados como palavras-chave logo abaixo dos títulos dos capítulos devem ser lembradas (K1), mesmo que não sejam explicitamente mencionadas nos objetivos de aprendizagem.

## 0.4 O Exame de certificação

O exame de certificação será baseado neste syllabus. As respostas às perguntas do exame podem exigir o uso de material com base em mais de um capítulo deste syllabus. Todas as seções são examináveis, com exceção da Introdução e Apêndices. Padrões, livros e outros syllabus do *ISTQB*® são incluídos como referências, mas seu conteúdo não é passível de análise, além do que está resumido neste próprio documento a partir de tais padrões, livros e outros syllabus do *ISTQB*®.

O exame possui 40 questões de múltipla escolha na língua portuguesa. Para passar no exame, pelo menos 65% das perguntas (ou seja, 26 perguntas) devem ser respondidas corretamente.

Os exames podem ser feitos como parte de um curso de treinamento credenciado ou realizados de forma independente (p. ex., em um centro de treinamento ou em um exame público). A conclusão de um curso de treinamento credenciado não é um pré-requisito para o exame.

## 0.5 Credenciamento

Um Conselho Nacional do *ISTQB*®, como o BSTQB, pode credenciar provedores de treinamento cujo material de curso segue este syllabus. Os provedores de treinamento devem obter diretrizes de credenciamento do Conselho Nacional ou órgão que realiza o credenciamento. Um curso acreditado é reconhecido em conformidade com este syllabus, e é permitido ter um exame do *ISTQB*® como parte do curso.

## 0.6 Nível de Detalhe

O nível de detalhe neste syllabus permite cursos e exames internacionalmente consistentes. Para atingir este objetivo, o syllabus possui:

- Objetivos gerais com instruções descrevendo a intenção do nível fundamental;

- Uma lista de termos que os alunos devem recordar;
- Os objetivos de aprendizagem para cada área do conhecimento, descrevendo o resultado do nível cognitivo a ser alcançado;
- Uma descrição dos principais conceitos, incluindo referências as fontes como literatura aceita ou à padrões.

O conteúdo programático não é uma descrição de toda a área de conhecimento do teste de software. Ele reflete o nível de detalhe a ser coberto nos cursos de treinamento do nível fundamental. Ele se concentra em conceitos e técnicas de teste que podem ser aplicados em todos os projetos de software, incluindo projetos ágeis. Este syllabus não contém nenhum objetivo de aprendizado específico relacionado a qualquer ciclo de vida ou método de desenvolvimento de software específico, mas discute como esses conceitos se aplicam em projetos ágeis, outros tipos de ciclo de vida iterativo e incremental e em ciclos de vida sequenciais.

## 0.7 Como este syllabus está organizado

Existem seis capítulos com conteúdo examinável. No título de cada um está especificado o tempo mínimo de dedicação ao estudo. Para cursos de formação credenciados, o syllabus requer um mínimo de 16h45 de treinamento, distribuídas por capítulo desta forma:

| Capítulo | Título | Tempo (min) |
|---|---|---|
| 1 | Fundamentos do teste | 175 |
| 2 | Teste durante o ciclo de vida de desenvolvimento de software | 100 |
| 3 | Testes estáticos | 135 |
| 4 | Técnicas de teste | 330 |
| 5 | Gerenciamento de teste | 225 |
| 6 | Ferramentas de apoio ao teste | 40 |

# 1 Fundamentos de teste [175 min]

### *Conceitos-chave*

cobertura, depuração de código, defeito, erro, falha, qualidade, garantia de qualidade, causa-raiz, análise do teste, base de teste, caso de teste, conclusão do teste, condição de teste, controle de teste, dados de teste, modelagem de teste, execução do teste, cronograma de execução do teste, implementação, monitoramento de teste, objeto de teste, objetivo de teste, oráculo de teste, processo de testeplanejamento de teste, procedimento de teste, suíte de teste, teste, testware, rastreabilidade, validação, verificação.

### *Objetivos de aprendizagem*

#### *1.1 O que é o teste?*

FL-1.1.1 (K1) Identificar os objetivos típicos de teste

FL-1.1.2 (K2) Diferenciar o teste da depuração

#### *1.2 Por que o teste é necessário?*

FL-1.2.1 (K2) Dar exemplos de porque o teste é necessário

FL-1.2.2 (K2) Descrever a relação entre testes e garantia de qualidade e dar exemplos de como os testes contribuem para melhorar a qualidade

FL-1.2.3 (K2) Distinguir erro, defeito e falha

FL-1.2.4 (K2) Distinguir causa-raiz de um defeito e seus efeitos

#### *1.3 Os sete princípios de testes*

FL-1.3.1 (K2) Explicar os sete princípios de teste

#### *1.4 Processo de teste*

FL-1.4.1 (K2) Explicar o impacto do contexto no processo de teste

FL-1.4.2 (K2) Descrever as atividades de teste e respectivas tarefas dentro do processo de teste

FL-1.4.3 (K2) Diferenciar os produtos de trabalho que suportam o processo de teste

FL-1.4.4 (K2) Explicar o valor de manter a rastreabilidade entre a base de teste e os produtos de trabalho de teste

#### *1.5 A psicologia do teste*

FL-1.5.1 (K1) Identificar os fatores psicológicos que influenciam o sucesso do teste

FL-1.5.2 (K2) Explicar a diferença entre a mentalidade necessária para as atividades de teste e a mentalidade necessária para atividades de desenvolvimento

## 1.1 O que é teste?

Os sistemas de software são parte integrante da vida, desde aplicações comerciais (p. ex., serviços bancários) até produtos de consumo (p. ex., carros). A maioria das pessoas já teve uma experiência com software que não funcionou como o esperado. O software que não funciona corretamente pode levar a muitos problemas, incluindo a perda de dinheiro, de tempo ou da reputação comercial, e até mesmo ferimentos ou morte. O teste de software é uma maneira de avaliar a qualidade do software e reduzir o risco de falha do software em operação.

Um equívoco comum do teste é que ele consiste apenas em executar testes, ou seja, executar o software e verificar os resultados. Conforme descrito no capítulo 1.4, o teste de software é um processo que inclui muitas atividades diferentes, e a execução do teste (incluindo a verificação dos resultados) é apenas uma dessas atividades. O processo de teste também inclui atividades como planejamento de teste, análise, modelagem e implementação dos testes, relatórios de progresso e resultados de testes e avaliação da qualidade de um objeto de teste.

Alguns testes envolvem a execução de um componente ou sistema que está sendo testado, sendo esse teste chamado de teste dinâmico. Outros testes não envolvem a execução do componente ou sistema que está sendo testado, sendo chamados de teste estático. Assim, o teste também inclui a revisão de produtos de trabalho, como requisitos, histórias de usuários e código-fonte.

Outro equívoco comum do teste é que ele se concentra inteiramente na verificação de requisitos, histórias de usuários ou outras especificações. Embora o teste envolva verificar se o sistema atende aos requisitos especificados, ele também contempla a validação, que está verificando se o sistema atenderá às necessidades do usuário e *stakeholders* em seu(s) ambiente(s) operacional(is).

As atividades de teste são organizadas e executadas de forma diferente em diferentes ciclos de vida (ver capítulo 2.1).

### 1.1.1 Objetivos típicos do teste

Para qualquer projeto, os objetivos do teste podem incluir informações para:

- Evitar defeitos, avaliar os produtos de trabalho, como requisitos, histórias de usuários, modelagem e código;
- Verificar se todos os requisitos especificados foram cumpridos;
- Verificar se o objeto de teste está completo e validar se funciona como os usuários e *stakeholders* esperam;
- Criar confiança no nível de qualidade do objeto de teste;
- Encontrar defeitos e falhas reduz o nível de risco de qualidade inadequada do software;
- Fornecer informações suficientes aos *stakeholders* para que tomem decisões especialmente em relação ao nível de qualidade do objeto de teste;
- Cumprir os requisitos ou normas contratuais, legais ou regulamentares, ou verificar a conformidade do objeto de teste com esses requisitos ou normas.

Os objetivos do teste podem variar, dependendo do contexto do componente ou sistema que está sendo testado, do nível de teste e do modelo de ciclo de vida de desenvolvimento de software. Essas diferenças podem incluir, por exemplo:

- Durante o teste do componente, um objetivo pode ser encontrar tantas falhas quanto possível, de modo que os defeitos subjacentes sejam identificados e corrigidos antecipadamente. Outro objetivo pode ser aumentar a cobertura de código dos testes de componentes;
- Durante o teste de aceite, um objetivo pode ser confirmar que o sistema funciona como esperado e satisfaz os requisitos. Outro objetivo deste teste pode ser fornecer informações aos *stakeholders* sobre o risco de liberar o sistema em um determinado momento.

### 1.1.2 Teste e depuração de código

Teste e depuração de código são diferentes. A execução dos testes pode mostrar falhas causadas por defeitos no software. A depuração de código é a atividade de desenvolvimento que localiza, analisa e corrige esses defeitos. Os testes de confirmação subsequentes verificam se as correções resolveram os defeitos. Em alguns casos, os testadores são responsáveis pelo teste inicial e pelo teste de confirmação final, enquanto os desenvolvedores fazem a depuração e o teste de componente associado. No entanto, no desenvolvimento ágil e em alguns outros ciclos de vida, os testadores podem estar envolvidos na depuração e no teste de componentes.

O padrão [ISO29119-1] tem mais informações sobre conceitos de teste de software.

## 1.2 Por que o teste é necessário?

Testes rigorosos de componentes e sistemas e sua documentação associada podem ajudar a reduzir o risco de falhas durante a operação. Quando defeitos são detectados e posteriormente corrigidos, há contribuição para a qualidade dos componentes ou sistemas. Além disso, o teste de software também é necessário para atender aos requisitos contratuais ou legais ou aos padrões específicos do setor.

### 1.2.1 Contribuições do teste para o sucesso

Ao longo da história da computação, é bastante comum que softwares e sistemas sejam entregues à operação e, devido à presença de defeitos, subsequentemente causar falhas ou não atender às necessidades dos *stakeholders*. No entanto, com o uso de técnicas de teste apropriadas pode-se reduzir a frequência de tais entregas problemáticas, quando essas técnicas são aplicadas com o nível apropriado de experiência em testes, e nos pontos certos do ciclo de vida de desenvolvimento de software. Exemplos incluem:

- Ter testadores envolvidos nas revisões de requisitos ou no refinamento da história do usuário poderia detectar defeitos nesses produtos de trabalho. A identificação e remoção de defeitos de requisitos reduz o risco de desenvolvimento de funcionalidade incorreta ou não testável;
- Ter testadores trabalhando em conjunto com os projetistas do sistema enquanto o sistema está sendo projetado pode aumentar o entendimento de cada parte sobre o projeto e como

testá-lo. Esse maior entendimento pode reduzir o risco de defeitos fundamentais de projeto e permitir que os testes sejam identificados em um estágio inicial;
- Ter testadores trabalhando em estreita colaboração com os desenvolvedores enquanto o código está em desenvolvimento pode aumentar o entendimento de cada parte sobre o código e como testá-lo. Esse aumento de compreensão pode reduzir o risco de defeitos no código e nos testes;
- Ter testadores para verificar e validar o software antes de liberar pode detectar falhas que poderiam ter sido perdidas e suportar o processo de remoção dos defeitos que causaram as falhas (ou seja, depuração de código). Isso aumenta a probabilidade de que o software atenda às necessidades dos *stakeholders* e satisfaça os requisitos.

Além desses exemplos, a aplicação dos objetivos do teste definidos (ver capítulo 1.1.1) contribui para o sucesso geral do desenvolvimento e manutenção de software.

### 1.2.2 Garantia da qualidade e teste

Enquanto as pessoas costumam usar a frase "Garantia de Qualidade" (ou apenas QA) para se referir a testes, as definições não são as mesmas, mas estão relacionadas. Um conceito maior, gerenciamento de qualidade, os une. A gestão da qualidade inclui todas as atividades que direcionam e controlam uma organização em relação à qualidade. Entre outras atividades, a gestão da qualidade inclui a garantia de qualidade e o controle de qualidade. A garantia de qualidade é tipicamente focada na adesão a processos adequados, a fim de fornecer confiança de que os níveis apropriados de qualidade serão alcançados. Quando os processos são realizados adequadamente, os produtos de trabalho criados por esses processos são geralmente de maior qualidade, o que contribui para a prevenção de defeitos. Além disso, o uso de análise de causa raiz para detectar e remover as causas de defeitos, juntamente com a aplicação adequada das conclusões de reuniões retrospectivas para melhorar os processos, é importante para garantir a qualidade efetiva.

O controle de qualidade envolve várias atividades, incluindo atividades de teste, que apoiam a obtenção de níveis adequados de qualidade. As atividades de teste são parte do processo geral de desenvolvimento ou manutenção de software. Como a garantia de qualidade está relacionada com a execução adequada de todo o processo, a garantia de qualidade apóia o teste. Conforme descrito nas seções 1.1.1 e 1.2.1, os testes contribuem para a obtenção da qualidade de várias maneiras.

### 1.2.3 Erros, defeitos e falhas

Uma pessoa pode cometer um erro (engano), que pode levar à introdução de um defeito (falha ou *bug*) no código do software ou em algum outro produto de trabalho relacionado. Um erro que leva à introdução de um defeito em um produto de trabalho pode acionar outro erro que leva à introdução de um defeito em um outro produto de trabalho relacionado. Por exemplo, um erro de elicitação de requisitos pode levar a um defeito de requisitos, o que resulta em um erro de programação que leva a um defeito no código.

A execução de um código defeituoso, pode causar uma falha, mas não necessariamente em todas as circunstâncias. Por exemplo, alguns defeitos exigem entradas ou pré-condições muito específicas para desencadear uma falha, o que pode ocorrer raramente ou nunca.

Erros podem ocorrer por vários motivos, como:

- Pressão do tempo;
- Falha humana;
- Participantes do projeto inexperientes ou insuficientemente qualificados;
- Falta de comunicação entre os participantes do projeto, incluindo falta de comunicação sobre os requisitos e a modelagem;
- Complexidade do código, modelagem, arquitetura, o problema a ser resolvido ou as tecnologias utilizadas;
- Mal-entendidos sobre interfaces intra-sistema e entre sistemas, especialmente quando tais interações são em grande número;
- Tecnologias novas, ou desconhecidas.

Além das falhas causadas devido a defeitos no código, falhas também podem ser causadas por condições ambientais. Por exemplo, radiação, campos eletromagnéticos e poluição podem causar defeitos no firmware ou influenciar a execução do software alterando as condições do hardware.

Nem todos os resultados inesperados de teste são considerados falhas. Falsos positivos podem ocorrer devido a erros na forma como os testes foram executados ou devido a defeitos nos dados de teste, no ambiente de teste ou em outro testware ou por outros motivos. A situação inversa também pode ocorrer, onde erros ou defeitos similares levam a falsos negativos. Falsos negativos são testes que não detectam defeitos que deveriam ser detectados; falsos positivos são relatados como defeitos, mas na verdade não são defeitos.

### 1.2.4 Defeitos, causas-raiz e efeitos

As causas-raiz dos defeitos são as primeiras ações ou condições que contribuíram para a criação dos defeitos. Os defeitos podem ser analisados para identificar suas causas-raiz, de modo a reduzir a ocorrência de defeitos similares no futuro. Ao realçar as causas-raiz mais significativas, a análise de causa-raiz pode levar à melhoria de processo que evitam a introdução de um número significativo de defeitos futuros.

Por exemplo, suponha que pagamentos de juros incorretos, devido a uma única linha incorreta de código, resulte em reclamações de clientes. O código defeituoso foi escrito para uma história do usuário que era ambígua, devido ao equívoco do proprietário do produto sobre como calcular juros. Se houver uma grande porcentagem de defeitos nos cálculos de juros e esses defeitos tiverem sua causa-raiz em mal-entendidos semelhantes, os proprietários do produto poderão ser treinados em como calcular juros para reduzir tais defeitos no futuro.

Neste exemplo, as reclamações dos clientes são efeitos. Os pagamentos de juros incorretos são considerados falhas. O cálculo incorreto no código é um defeito e resultou do defeito original, a ambiguidade na história do usuário. A causa-raiz do defeito original foi a falta de conhecimento por

parte do proprietário do produto, o que resultou no erro do proprietário do produto ao escrever a história do usuário. O processo de análise de causa-raiz é discutido nos syllabi [ISTQB_EL_TM] e [ISTQB_EL_TP].

## 1.3 Os sete princípios de testes

Vários princípios de testes foram sugeridos nos últimos 50 anos e oferecem diretrizes gerais comuns para todos os testes.

### 1. O teste mostra a presença de defeitos e não a sua ausência

O teste reduz a probabilidade de defeitos não descobertos permanecerem no software, mas, mesmo se nenhum defeito for encontrado, o teste não é uma prova de correção.

### 2. Testes exaustivos são impossíveis

Testar tudo (todas as combinações de entradas e pré-condições) não é viável, exceto em casos triviais. Em vez de tentar testar exaustivamente, a análise de risco, as técnicas de teste e as prioridades devem ser usadas para concentrar os esforços de teste.

### 3. O teste inicial economiza tempo e dinheiro

Para encontrar antecipadamente os defeitos, as atividades de teste estático e dinâmico devem iniciar o mais cedo possível no ciclo de vida de desenvolvimento de software. O teste inicial é por vezes referido como *shift left*. O teste no início do ciclo de vida de desenvolvimento de software ajuda a reduzir ou eliminar alterações dispendiosas (ver capítulo 3.1).

### 4. Defeitos se agrupam

Um pequeno número de módulos geralmente contém a maioria dos defeitos descobertos durante o teste de pré-lançamento ou é responsável pela maioria das falhas operacionais. Agrupamento de defeitos previstos e os agrupamentos de defeitos observados reais em teste ou produção, são uma entrada importante em uma análise de risco usada para focar o esforço de teste (como mencionado no princípio 2).

### 5. Cuidado com o paradoxo do pesticida

Se os mesmos testes forem repetidos várias vezes, esses testes não encontrarão novos defeitos. Para detectar novos defeitos, os testes existentes e os dados de teste podem precisar ser alterados e novos testes precisam ser gravados. (Testes não são mais eficazes em encontrar defeitos, assim como pesticidas não são mais eficazes em matar insetos depois de um tempo.) Em alguns casos, como o teste de regressão automatizado, o paradoxo do pesticida tem um resultado benéfico, que é o número relativamente baixo de defeitos de regressão.

### 6. O teste depende do contexto

O teste é feito de forma diferente em diferentes contextos. Por exemplo, o software de controle industrial de segurança crítica é testado de forma diferente de um aplicativo móvel de comércio

eletrônico. Como outro exemplo, o teste em um projeto ágil é feito de forma diferente do que o teste em um projeto de ciclo de vida sequencial (ver capítulo 2.1).

### 7. Ausência de erros é uma ilusão

Algumas organizações esperam que os testadores possam executar todos os testes possíveis e encontrar todos os defeitos possíveis, mas os princípios 2 e 1, respectivamente, nos dizem que isso é impossível. Além disso, é uma ilusão (isto é, uma crença equivocada) esperar que apenas encontrar e corrigir muitos defeitos assegure o sucesso de um sistema. Por exemplo, testar exaustivamente todos os requisitos especificados e corrigir todos os defeitos encontrados ainda pode produzir um sistema difícil de usar, que não atenda às necessidades e expectativas dos usuários ou que seja inferior em comparação com outros sistemas concorrentes.

Veja Myers (2011), Kaner (2002) e Weinberg (2008) para exemplos destes e outros princípios de teste.

## 1.4 Processos de teste

Não existe um processo universal de teste de software, mas há conjuntos comuns de atividades de teste sem as quais os testes terão menor probabilidade de atingir seus objetivos estabelecidos. Esses conjuntos de atividades de teste são um processo de teste. O processo de teste de software específico e apropriado em qualquer situação depende de muitos fatores. Quais atividades de teste estão envolvidas nesse processo de teste, como essas atividades são implementadas e quando essas atividades ocorrem podem ser discutidas na estratégia de teste de uma organização.

### 1.4.1 Processo de teste no contexto

Alguns fatores contextuais que influenciam o processo de teste de uma organização:

- Modelo de ciclo de vida de desenvolvimento de software e metodologias de projeto utililzados;
- Níveis de teste e tipos de teste considerados;
- Riscos de produto e projeto;
- Domínio do negócio;
- Algumas restrições operacionais:
    - Orçamentos e recursos;
    - Escalas de tempo;
    - Complexidade;
    - Requisitos contratuais e regulamentares.
- Políticas e práticas organizacionais;
- Normas internas e externas necessárias.

As seções a seguir descrevem aspectos gerais dos processos de teste organizacionais:

- Atividades e tarefas de teste;
- Produtos de trabalho de teste;
- Rastreabilidade entre a base de teste e os produtos de trabalho de teste.

É muito útil se a base de teste (para qualquer nível ou tipo de teste que está sendo considerado) tiver definida os critérios de cobertura mensuráveis. Os critérios de cobertura podem atuar efetivamente como indicadores-chave de performance (KPIs) para conduzir as atividades que demonstram a realização dos objetivos de teste de software (ver capítulo 1.1.1).

Por exemplo, para uma aplicação móvel, a base de teste pode incluir uma lista de requisitos e uma lista de dispositivos móveis suportados. Cada requisito é um elemento da base de teste. Cada dispositivo suportado também é um elemento da base de teste. Os critérios de cobertura podem exigir pelo menos um caso de teste para cada elemento da base de teste. Uma vez executados, os resultados desses testes informam aos interessados se os requisitos especificados são atendidos e se as falhas foram observadas nos dispositivos suportados.

A norma ISO [ISO29119-2] contém mais informações sobre os processos de teste.

## 1.4.2 Atividades e tarefas de teste

Um processo de teste consiste nos seguintes grupos principais de atividades:

- Planejamento do teste;
- Monitoramento e controle do teste;
- Análise do teste;
- Modelagem do teste;
- Implementação do teste;
- Execução do teste;
- Conclusão do teste.

Cada grupo de atividades é composto por atividades constituintes, que serão descritas nas subseções abaixo. Cada atividade dentro de cada grupo de atividades, por sua vez, pode consistir em várias tarefas individuais, que podem variar de um projeto ou lançamento para outro.

Além disso, embora muitos desses grupos de atividades possam parecer logicamente sequenciais, eles são frequentemente implementados iterativamente. Por exemplo, o desenvolvimento ágil envolve pequenas iterações de projeto, construção e teste de software que acontecem de forma contínua, suportadas pelo planejamento contínuo. Assim, as atividades de teste também estão acontecendo de forma contínua e iterativa dentro dessa abordagem de desenvolvimento. Mesmo em desenvolvimento sequencial, a sequência lógica de atividades escalonada envolverá sobreposição, combinação, simultaneidade ou omissão, de modo que a adaptação dessas atividades principais dentro do contexto do sistema e do projeto geralmente seja necessária.

### Planejamento do teste

O planejamento do teste envolve as atividades que definem os propósitos e a abordagem do teste para atender aos objetivos do teste dentro das restrições impostas pelo contexto (p. ex., especificar técnicas e tarefas de teste adequadas e formular um cronograma de teste para cumprir um prazo). Os planos de teste podem ser revisitados com base no feedback das atividades de monitoramento e controle. O planejamento do teste é explicado no capítulo 5.2.

### Monitoramento e controle do teste

O monitoramento de teste envolve a comparação contínua do progresso real com o plano de teste usando qualquer métrica de monitoramento definida no plano de teste. O controle do teste engloba a tomada de ações necessárias para atender aos objetivos do plano de teste (que pode ser atualizado ao longo do tempo). O monitoramento e o controle dos testes são apoiados pelos critérios de avaliação das saídas, que são referidos como "feito" em alguns ciclos de vida (veja o [ISTQB_FL_AT]). Por exemplo, a avaliação dos critérios de saída para a execução dos testes como parte de um determinado nível de teste pode incluir:

- Verificar os resultados do teste e os registros em relação aos critérios especificados de cobertura;
- Avaliar o nível de qualidade do componente ou sistema com base nos resultados e registros dos testes;
- Determinar se são necessários mais testes (p. ex., se os testes originalmente destinados a atingir um nível de cobertura de risco do produto falharam, exigindo que testes adicionais sejam escritos e executados).

O progresso do teste em relação ao plano é comunicado aos *stakeholders* nos relatórios de progresso do teste, incluindo os desvios do plano e as informações para apoiar qualquer decisão de interromper o teste.

O monitoramento e controle dos testes são explicados no capítulo 5.3.

### Análise do teste

Durante a análise do teste, a base de teste é analisada para identificar recursos testáveis e definir as condições de teste associadas. Em outras palavras, a análise do teste determina "o que testar" em termos dos critérios de cobertura mensuráveis.

A análise do teste inclui as seguintes atividades principais:

- Analisar a base de teste apropriada ao nível de teste que está sendo utilizado, por exemplo:
    - As especificações de requisitos, como requisitos de negócios, requisitos funcionais, requisitos do sistema, histórias de usuários, épicos, casos de uso ou produtos de trabalho semelhantes que especificam o componente funcional ou não funcional desejado ou o comportamento do sistema;
    - A modelagem e a implementação de informações, como diagramas ou documentos de arquitetura de sistema ou software, especificações de modelagem, fluxos de chamadas, diagramas de modelagem (p. ex., diagramas de UML ou de entidade), especificações de interface ou produtos de trabalho semelhantes que especifiquem componentes ou estrutura do sistema;
    - A implementação do componente ou sistema em si, incluindo o código, metadados e consultas ao banco de dados e interfaces;
    - Os relatórios de análise de risco, que podem considerar os aspectos funcionais, não-funcionais e estruturais do componente ou sistema.

- Avaliar a base de teste e os itens de teste para identificar os vários tipos de defeitos, como:
  - Ambiguidades;
  - Omissões;
  - Inconsistências;
  - Imprecisões;
  - Contradições;
  - Declarações supérfluas.
- Identificar os recursos e os conjuntos de recursos a serem testados;
- Definir e priorizar as condições de teste para cada recurso com base na análise da base de teste e considerando as características funcionais, não-funcionais e estruturais, outros fatores comerciais e técnicos e níveis de riscos;
- Capturar a rastreabilidade bidirecional entre cada elemento da base de teste e as condições de teste associadas (ver capítulos 1.4.3 e 1.4.4).

A aplicação de técnicas de teste caixa-preta, caixa-branca e experiência pode ser útil no processo de análise do teste (ver capítulo 4) para reduzir a probabilidade de omitir as condições importantes de teste e definir as condições de teste mais corretas e precisas.

Em alguns casos, a análise do teste produz condições de teste que devem ser usadas como objetivos de teste em cartas de teste. As cartas de teste são típicos produtos de trabalho em alguns tipos de testes baseados na experiência (ver capítulo 4.4.2). Quando esses objetivos do teste são rastreáveis até a base de teste, a cobertura obtida durante esses testes baseados na experiência pode ser medida.

A identificação dos defeitos durante a análise do teste é um benefício potencial importante, especialmente quando nenhum outro processo de revisão está sendo usado e/ou o processo de teste está intimamente ligado ao processo de revisão. Essas atividades de análise do teste não apenas verificam se os requisitos são consistentes, expressos adequadamente e completos, mas também validam se os requisitos capturam adequadamente as necessidades do cliente, do usuário e *stakeholders*. Por exemplo, as técnicas como desenvolvimento orientado pelo comportamento (BDD) e desenvolvimento orientado por teste de aceite (ATDD), que envolvem a geração de condições de teste e casos de teste de histórias de usuários e critérios de aceite antes da codificação, também verificam, validam e detectam defeitos nas histórias do usuário e nos critérios de aceite (consulte [ISTQB_FL_AT]).

### Modelagem do teste

Durante a modelagem de teste, as condições de teste são elaboradas em casos de teste de alto nível, em conjuntos de casos de teste de alto nível e outros *testwares*. Assim, a análise do teste responde à pergunta “o que testar?”, enquanto a modelagem de teste responde à pergunta “como testar?”

A modelagem de teste inclui as seguintes atividades principais:

- Projetar e priorizar casos de teste e conjuntos de casos de teste;

- Identificar os dados de teste necessários para comportar as condições de teste e os casos de teste;
- Projetar o ambiente de teste e identificar qualquer infraestrutura e ferramenta necessária;
- Capturar a rastreabilidade bidirecional entre a base de teste, as condições de teste, os casos de teste e os procedimentos de teste (ver capítulo 1.4.4).

A elaboração das condições de teste em casos de teste e conjuntos de casos de teste durante a modelagem do teste envolve muitas vezes o uso de técnicas de teste (ver capítulo 4).

Assim como na análise do teste, a modelagem do teste também pode resultar na identificação de tipos semelhantes de defeitos na base de teste. Também como na análise do teste, a identificação dos defeitos durante a modelagem do teste é um benefício potencial importante.

### Implementação do teste

Durante a implementação do teste, o testware necessário para a execução do teste é criado ou concluído, incluindo o sequenciamento dos casos de teste nos procedimentos de teste. Portanto, a modelagem de teste responde à pergunta "*como testar?*", enquanto a implementação do teste responde à pergunta "*agora temos tudo para executar os testes?*"

A implementação do teste inclui principalmente as seguintes atividades:

- Desenvolver e priorizar os procedimentos de teste e, potencialmente, criar os scripts de teste automatizados;
- Criar as suítes de teste a partir dos procedimentos de teste e (se houver) os scripts de teste automatizados;
- Organizar os conjuntos de testes dentro de um cronograma de maneira que resulte em maior eficiência a execução dos testes (ver capítulo 5.2.4);
- Construir o ambiente de teste (incluindo, potencialmente, equipamentos de teste, virtualização de serviços, simuladores e outros itens de infraestrutura), e verificando se tudo o que é necessário foi configurado corretamente;
- Preparar os dados de teste e garantir que eles sejam carregados corretamente no ambiente de teste;
- Verificar e atualizar a rastreabilidade bidirecional entre a base de teste, as condições de teste, os casos de teste, procedimentos de teste e suítes de teste (ver capítulo 1.4.4).

As tarefas do projeto de teste e a implementação do teste são frequentemente combinadas.

Nos testes exploratórios e em outros tipos de testes baseados na experiência, a modelagem e a implementação do teste podem ocorrer e serem documentadas como parte da execução do teste. Os testes exploratórios podem ser baseados em cartas de teste (produzidas como parte da análise dos testes) e os executados imediatamente à medida que são projetados e implementados (ver capítulo 4.4.2).

### Execução do teste

Durante a execução do teste, os conjuntos de testes são executados de acordo com a programação de execução do teste.

A execução do teste inclui principalmente as seguintes atividades:

- Gravar os identificadores e versões do(s) item(ns) de teste ou do objeto de teste, da(s) ferramenta(s) de teste e testware;
- Executar os testes manualmente ou usando ferramentas de execução do teste;
- Comparar os resultados reais com os resultados esperados;
- Analisar as anomalias para estabelecer suas prováveis causas (p. ex., falhas podem ocorrer devido a defeitos no código, mas falsos positivos também podem ocorrer [ver capítulo 1.2.3]);
- Comunicar os defeitos com base nas falhas observadas (ver capítulo 5.6);
- Registrar o resultado da execução do teste (p. ex., passar, falhar, bloquear);
- Repetir as atividades de teste como resultado de uma ação tomada por uma anomalia, ou como parte do planejado para o teste (p. ex., execução de um teste corrigido, teste de confirmação ou teste de regressão);
- Verificar e atualizar a rastreabilidade bidirecional entre a base de teste, as condições de teste, os casos de teste, os procedimentos de teste e os resultados de teste.

### Conclusão do teste

As atividades de conclusão do teste coletam os dados das atividades de teste já concluídas para consolidar a experiência, o testware e qualquer outra informação relevante. As atividades de conclusão do teste ocorrem nos marcos do projeto, como quando um sistema de software é lançado, um projeto de teste é concluído (ou cancelado), uma iteração de projeto ágil é concluída (p. ex., como parte de uma reunião retrospectiva), um nível de teste é concluído ou uma liberação de manutenção foi concluída.

A conclusão do teste inclui principalmente as seguintes atividades:

- Verificar se todos os relatórios de defeitos estão fechados, inserindo as solicitações de mudança ou itens de lista não processada do produto para quaisquer defeitos que não foram resolvidos no final da execução do teste;
- Criar um relatório de resumo de teste para ser comunicado aos *stakeholders*;
- Finalizar e arquivar o ambiente de teste, os dados de teste, a infraestrutura de teste e outros *testwares* para posterior reutilização;
- Entregar o testware para as equipes de manutenção, outras equipes de projeto ou *stakeholders* que poderiam se beneficiar de seu uso;
- Analisar as lições aprendidas das atividades de teste concluídas para determinar as alterações necessárias para futuras iterações, releases e projetos;
- Usar as informações coletadas para melhorar a maturidade do processo de teste.

### 1.4.3 Produtos de trabalho do teste

Os produtos de teste são criados como parte do processo de teste. Assim como há uma variação significativa na maneira como as organizações implementam o processo de teste, há também uma variação significativa nos tipos de produtos de trabalho criados durante esse processo, nas formas como esses produtos são organizados e gerenciados e nos nomes usados. Este syllabus segue o processo de teste descrito acima e os produtos de trabalho descritos neste syllabus e no Glossário de Testes [ISTQB_Glossary]. O padrão [ISO29119-3] também pode servir como diretriz para produtos de trabalho de teste.

Muitos dos produtos de trabalho de teste descritos neste capítulo podem ser capturados e gerenciados usando ferramentas de gerenciamento de teste e ferramentas de gerenciamento de defeitos (ver capítulo 6).

#### Produtos de trabalho do planejamento do teste

Os produtos de trabalho de planejamento do teste geralmente incluem um ou mais planos de teste. O plano de teste inclui informações sobre a base de teste, com as quais os outros produtos de teste serão relacionados através das informações de rastreabilidade (ver abaixo e capítulo 1.4.4), bem como os critérios de saída (ou definidos como “feito”) que serão usados durante monitoramento e controle do teste. Planos de teste são descritos no capítulo 5.2.

#### Produtos de trabalho de monitoramento e controle do teste

Os produtos de trabalho de monitoramento e controle do teste geralmente incluem vários tipos de relatórios, incluindo relatórios de progresso do teste (produzidos em uma base contínua ou regular) e relatórios de resumo do teste (produzidos em vários marcos de conclusão). Todos os relatórios de teste devem fornecer detalhes relevantes do público sobre o progresso do teste a partir da data do relatório, incluindo o resumo dos resultados da execução do teste assim que eles estiverem disponíveis.

Os produtos de trabalho de monitoramento e controle do teste também devem abordar as preocupações de gerenciamento de projetos, como conclusão das tarefas, a alocação e uso de recursos e o esforço.

O monitoramento e controle do teste, e os produtos de trabalho criados durante essas atividades, são explicados mais detalhadamente no capítulo 5.3 deste syllabus.

#### Produtos de trabalho da análise do teste

Os produtos de trabalho de análise do teste incluem condições de teste definidas e priorizadas, preferencialmente onde cada uma das quais é bidirecionalmente rastreável para o(s) elemento(s) específico(s) da base de teste que a cobre. Para testes exploratórios, a análise do teste pode envolver a criação de cartas de teste. A análise do teste também pode resultar na descoberta e no relato de defeitos na base de teste.

### Produtos de trabalho da modelagem do teste

A modelagem do teste resulta em casos de teste e conjuntos de casos de teste para exercer as condições de teste definidas na análise do teste. Geralmente, é uma boa prática projetar casos de teste de alto nível, sem valores concretos para dados de entrada e resultados esperados. Esses casos de teste de alto nível são reutilizáveis em vários ciclos de teste com diferentes dados concretos, enquanto ainda documentam adequadamente o escopo do caso de teste. Preferencialmente, cada caso de teste é bidirecionalmente rastreável às condições de teste cobertas.

A modelagem do teste também resulta no projeto ou na identificação dos dados necessários de teste, na modelagem do ambiente de teste, e na identificação de infraestrutura e ferramentas, embora a extensão na qual esses resultados sejam documentados varie significativamente.

As condições de teste definidas na análise do teste podem ser mais refinadas na modelagem do teste.

### Produtos de trabalho da implementação do teste

Os produtos de trabalho de implementação do teste incluem:

- Os procedimentos de teste e seu sequenciamento;
- As suítes de teste;
- Um cronograma de execução do teste.

Preferencialmente, uma vez concluída a implementação do teste, a obtenção dos critérios de cobertura estabelecidos no plano de teste pode ser demonstrada por meio da rastreabilidade bidirecional entre os procedimentos e os elementos específicos da base de teste, através dos casos de teste e das condições de teste.

Em alguns casos, a implementação do teste envolve a criação de produtos de trabalho usando ou que serão usados por ferramentas específicas, como virtualização de serviços e scripts de teste automatizados.

A implementação do teste também pode resultar na criação e verificação dos dados de teste e do ambiente de teste. A integridade da documentação dos resultados, da verificação dos dados, ou do ambiente pode variar significativamente.

Os dados de teste servem para atribuir valores concretos às entradas e aos resultados esperados da execução dos casos de teste. Esses valores concretos, juntamente com instruções explícitas sobre o uso destes valores, transformam casos de teste de alto nível em casos de teste executáveis de baixo nível. O mesmo caso de teste de alto nível pode usar dados de teste diferentes quando são executados em diferentes *releases* do objeto de teste. Os resultados esperados que estão associados aos dados de teste são identificados usando um oráculo de teste.

Nos testes exploratórios, alguns produtos de trabalho da modelagem e implementação do teste podem ser criados durante a execução do teste, embora a extensão na qual os testes exploratórios (e sua rastreabilidade para elementos específicos da base de teste) estejam documentados possa variar significativamente.

As condições de teste definidas na análise do teste podem ser refinadas na implementação do teste.

### Produtos de trabalho da execução do teste

Os produtos de trabalho da execução do teste incluem:

- A documentação do status dos casos de teste individuais ou procedimentos de teste (p. ex., pronto para executar, passar, falhar, bloquear, ignorar deliberadamente etc.);
- Os relatórios de defeitos (ver capítulo 5.6);
- A documentação sobre quais os itens de teste, o(s) objeto(s) de teste, as ferramentas de teste e o testware estavam envolvidos no teste.

Preferencialmente, uma vez concluída a execução do teste, o status de cada elemento da base de teste pode ser determinado e relatado via rastreabilidade bidirecional com o(s) procedimento(s) de teste associado(s). Por exemplo, podemos dizer quais requisitos passaram em todos os testes planejados, quais requisitos falharam nos testes ou que possuem defeitos associados a eles e quais requisitos têm testes planejados ainda aguardando para serem executados. Isso permite a verificação de que os critérios de cobertura foram atendidos e permite o relato de resultados de testes em termos que sejam compreensíveis para os *stakeholders*.

### Produtos de trabalho de conclusão do teste

Os produtos de trabalho de conclusão do teste incluem os relatórios de resumo de teste, os itens de ação para melhoria de projetos subsequentes ou iterações (p. ex., após um projeto de retrospectiva ágil), as solicitações de mudança ou os itens finalilzados de backlog de produto e testware.

## 1.4.4 Rastreabilidade entre a base de teste e os produtos de trabalho de teste

Conforme mencionado no capítulo 1.4.3, os produtos de trabalho de teste e os nomes desses produtos de trabalho variam significativamente. Independentemente dessas variações, a fim de implementar o monitoramento e controle efetivo do teste, é importante estabelecer e manter a rastreabilidade durante todo o processo de teste entre cada elemento da base de teste e os vários produtos de teste associados a esse elemento, conforme descrito acima. Além da avaliação da cobertura de teste, a boa rastreabilidade suporta:

- Analisar o impacto das mudanças;
- Tornar o teste auditável;
- Atender aos critérios de governança de TI;
- Melhorar a compreensibilidade dos relatórios de progresso do teste e dos relatórios de resumo do teste para incluir o status dos elementos da base de teste (p. ex., requisitos que passaram em seus testes, requisitos que falharam em seus testes e requisitos que têm testes pendentes);
- Relacionar os aspectos técnicos do teste com os *stakeholders* em termos que eles possam entender;
- Fornecer informações para avaliar a qualidade do produto, a capacidade do processo e o progresso do projeto em relação às metas de negócios.

Algumas ferramentas de gerenciamento de teste fornecem modelos de produtos de trabalho de teste que correspondem a parte ou a totalidade dos produtos de trabalho de teste descritos neste capítulo. Algumas organizações criam seus próprios sistemas de gerenciamento para organizar os produtos de trabalho e fornecer a rastreabilidade de informações de que necessitam.

## 1.5 A psicologia do teste

O desenvolvimento de software, incluindo os testes de software, envolve seres humanos. Portanto, a psicologia humana tem efeitos importantes no teste de software.

### 1.5.1 Psicologia humana e os testes

A identificação de defeitos durante um teste estático, como uma revisão de requisitos ou sessão de refinamento da história do usuário, ou a identificação de falhas durante a execução do teste dinâmico, pode ser percebida como uma crítica ao produto e ao seu autor. Um elemento da psicologia humana chamado "viés de confirmação" pode dificultar o aceite das informações que não concordam com as crenças atualmente mantidas. Por exemplo, como os desenvolvedores esperam que o código esteja correto, eles têm um viés de confirmação que dificulta o aceite do código incorreto. Além do viés de confirmação, outros vieses cognitivos podem dificultar que as pessoas entendam ou aceitem informações produzidas por testes. Além disso, é um traço humano comum culpar o portador de más notícias, e as informações produzidas pelos testes muitas vezes contêm más notícias.

Como resultado desses fatores psicológicos, algumas pessoas podem perceber o teste como uma atividade destrutiva, embora contribua grandemente para o progresso do projeto e a qualidade do produto (ver capítulos 1.1 e 1.2). Para tentar reduzir essas percepções, as informações sobre os defeitos e as falhas devem ser comunicadas de maneira construtiva. Dessa forma, as tensões entre testadores, analistas, proprietários de produtos, *designers* e desenvolvedores podem ser reduzidas. Isso se aplica durante os testes estáticos e dinâmicos.

Os testadores e gerentes de teste precisam ter boas habilidades interpessoais para se comunicarem de forma eficaz sobre defeitos, falhas, resultados de testes, progresso de testes e riscos, e para construir relacionamentos positivos com os colegas. As maneiras de se comunicar bem incluem os exemplos:

- Comece com colaboração em vez de batalhas. Lembre a todos do objetivo comum de sistemas de melhor qualidade;
- Enfatize os benefícios do teste. Por exemplo, para os autores, informações sobre defeitos podem ajudá-los a melhorar seus produtos de trabalho e suas habilidades. Para a organização, os defeitos encontrados e corrigidos durante os testes economizarão tempo e dinheiro e reduzirão o risco geral à qualidade do produto;
- Comunique os resultados dos testes e outras descobertas de uma maneira neutra, focada no fato, sem criticar a pessoa que criou o item com defeito. Escreva relatórios de defeitos objetivos e factuais e revise os resultados;

- Tente entender como a outra pessoa se sente e as razões pelas quais ela pode reagir negativamente à informação;
- Confirme se a outra pessoa entendeu o que foi dito e vice-versa.

Os objetivos típicos do teste foram discutidos anteriormente (ver capítulo 1.1). Definir claramente o conjunto certo dos objetivos do teste tem implicações psicológicas importantes. A maioria das pessoas tende a alinhar seus planos e comportamentos com os objetivos definidos pela equipe, pela gerência e por stakeholders. Também é importante que os testadores sigam esses objetivos com um mínimo viés pessoal.

### 1.5.2 A mentalidade do testador e do desenvolvedor

Os desenvolvedores e testadores geralmente pensam de forma diferente. O objetivo principal do desenvolvimento é projetar e construir um produto. Como discutido anteriormente, os objetivos do teste incluem verificar e validar o produto, encontrar os defeitos antes da liberação e assim por diante. Estes são conjuntos diferentes de objetivos que requerem diferentes mentalidades. Reunir essas mentalidades ajuda a alcançar um nível mais alto de qualidade do produto.

Uma mentalidade reflete as suposições de um indivíduo e os métodos preferidos para a tomada de decisões e a solução de problemas. A mentalidade de um testador deve incluir curiosidade, pessimismo profissional, olho crítico, atenção aos detalhes e motivação para comunicações e relacionamentos bons e positivos. A mentalidade de um testador tende a crescer e amadurecer à medida que o testador ganha experiência.

A mentalidade de um desenvolvedor pode incluir alguns dos elementos da mentalidade de um testador, mas os desenvolvedores bem-sucedidos geralmente estão mais interessados em projetar e criar soluções do que em contemplar o que pode estar errado nessas soluções. Além disso, o viés de confirmação torna difícil encontrar erros em seu próprio trabalho.

Com a mentalidade certa, os desenvolvedores podem testar seu próprio código. Diferentes modelos de ciclo de vida de desenvolvimento de software geralmente têm maneiras diferentes de organizar os testadores e as atividades de teste. Ter algumas das atividades de teste feitas por testadores independentes aumenta a eficácia da detecção de defeitos, o que é particularmente importante para sistemas grandes, complexos ou de segurança crítica. Testadores independentes trazem uma perspectiva que é diferente daquela dos autores de produtos de trabalho (isto é, analistas de negócios, proprietários de produtos, *designers* e programadores), uma vez que eles têm diferentes vieses cognitivos dos autores.

# 2 Teste durante o ciclo de vida de desenvolvimento de software [100 min]

## *Conceitos-chave*

testes de aceite, alfa teste, beta teste, COTS, teste de integração de componentes, teste de componentes, teste de confirmação, teste de aceite contratual, testes funcionais, análise de impacto, teste de integração, teste de manutenção, testes não funcionais, testes de aceite operacional, teste de regressão, teste de aceite regulatório, modelo de desenvolvimento sequencial, teste de integração de sistemas, teste de sistema, teste relacionado a mudança, base de teste, caso de teste, ambiente de teste, nível de teste, objeto de teste, objetivo de teste, tipo de teste, teste de aceite do usuário.

## *Objetivos de aprendizagem*

### *2.1 Modelos de ciclo de vida de desenvolvimento de software*

FL-2.1.1 (K2) Explicar as relações entre as atividades de desenvolvimento de software e as atividades de teste no ciclo de vida de desenvolvimento de software.

FL-2.1.2 (K1) Identificar os motivos pelos quais os modelos de ciclo de vida de desenvolvimento de software devem ser adaptados ao contexto das características do projeto e do produto.

### *2.2 Níveis de teste*

FL-2.2.1 (K2) Comparar os diferentes níveis de teste na perspectiva de seus objetivos, na base de teste, nos artefatos de teste, nas falhas e defeitos típicos e nas abordagens e responsabilidades.

### *2.3 Tipos de teste*

FL-2.3.1 (K2) Comparar os testes funcionais dos não-funcionais e caixa-branca

FL-2.3.2 (K1) Reconhecer que os testes funcionais, não funcionais e caixa-branca ocorrem em qualquer nível de teste.

FL-2.3.3 (K2) Comparar as finalidades do teste de confirmação e teste de regressão.

### *2.4 Teste de manutenção*

FL-2.4.1 (K2) Resumir os gatilhos para o teste de manutenção

FL-2.4.2 (K2) Descrever o papel da análise de impacto no teste de manutenção

## 2.1 Modelos de ciclo de vida de desenvolvimento de software

Um modelo de ciclo de vida de desenvolvimento de software descreve os tipos de atividades realizadas em cada estágio de um projeto de desenvolvimento de software e como as atividades se relacionam umas com as outras de forma lógica e cronológica. Há vários modelos de ciclo de vida de desenvolvimento de software, cada um dos quais requer abordagens diferentes para o teste.

### 2.1.1 Desenvolvimento de software e teste de software

É uma parte importante do papel de um testador estar familiarizado com os modelos mais comuns de ciclo de vida de desenvolvimento de software para que as atividades apropriadas de teste possam ocorrer.

Em qualquer modelo de ciclo de vida de desenvolvimento de software, existem várias características para um bom teste:

- Para cada atividade de desenvolvimento, existe uma atividade de teste correspondente;
- Cada nível de teste tem objetivos de teste específicos;
- A análise e a modelagem de teste para um determinado nível de teste começam durante a atividade de desenvolvimento correspondente;
- Os testadores participam de discussões para definir e refinar os requisitos e a modelagem, e estão envolvidos na revisão dos produtos de trabalho (p. ex., requisitos, modelagem, histórias de usuários etc.) assim que os rascunhos estiverem disponíveis.

Independentemente do modelo de ciclo de vida de desenvolvimento de software escolhido, as atividades de teste devem começar nos estágios iniciais do ciclo de vida, aderindo ao princípio de testar do início.

Este syllabus categoriza os modelos mais comuns de ciclo de vida de desenvolvimento de software da seguinte forma:

- Modelos de desenvolvimento sequencial;
- Modelos de desenvolvimento iterativo e incremental.

Um Modelo de Desenvolvimento Sequencial descreve o processo de desenvolvimento de software como um fluxo sequencial e linear de atividades. Isso significa que qualquer fase do processo de desenvolvimento deve começar quando a fase anterior estiver concluída. Em teoria, não há sobreposição de fases, mas, na prática, é benéfico ter antecipadamente o feedback da fase seguinte.

No Modelo Cascata, as atividades de desenvolvimento (p. ex., análise de requisitos, projeto, codificação, teste) são concluídas uma após a outra. Nesse modelo, as atividades de teste só ocorrem após todas as outras atividades de desenvolvimento terem sido concluídas.

Ao contrário do Modelo Castaca, o Modelo V integra o processo de teste ao longo do processo de desenvolvimento, implementando o princípio de testar do início. Além disso, o Modelo V inclui níveis de teste associados a cada fase de desenvolvimento correspondente, ainda suportando o princípio de testar do início, (consulte o capítulo 2.2 para uma discussão sobre níveis de teste). Nesse modelo,

a execução dos testes associados a cada nível de teste ocorre sequencialmente, mas em alguns casos pode ocorrer a sobreposição.

Os modelos de desenvolvimento sequencial fornecem softwares que contêm o conjunto completo de recursos, mas normalmente exigem meses ou anos para serem entregues aos stakeholders e aos usuários.

O desenvolvimento incremental envolve o estabelecimento de requisitos, a modelagem, a construção e o teste de um sistema em partes, o que significa que os recursos do software crescem de forma incremental. O tamanho desses incrementos de recurso varia, com alguns métodos tendo partes maiores e algumas partes menores. Os incrementos de recurso podem ser tão pequenos quanto uma única alteração para uma tela de interface do usuário ou uma nova opção de consulta.

O desenvolvimento iterativo ocorre quando grupos de recursos são especificados, projetados, construídos e testados juntos em uma série de ciclos, geralmente com duração fixa. Iterações podem envolver mudanças em recursos desenvolvidos em iterações anteriores, juntamente com mudanças no escopo do projeto. Cada iteração fornece software funcional que é um subconjunto crescente do conjunto geral de recursos até que o software final seja entregue ou o desenvolvimento seja interrompido.

Exemplos incluem:

- **Rational Unified Process**: cada iteração tende a ser relativamente longa (p. ex., dois a três meses) e os incrementos de recursos são correspondentemente grandes, como dois ou três grupos de recursos relacionados;
- **Scrum**: cada iteração tende a ser relativamente curta (p. ex., horas, dias ou algumas semanas) e os incrementos de recursos são correspondentemente pequenos, como alguns aprimoramentos e / ou dois ou três novos recursos;
- **Kanban**: implementado com ou sem iterações de tamanho fixo, que podem fornecer um único aprimoramento ou recurso após a conclusão ou agrupar recursos para serem liberados imediatamente;
- **Espiral** (ou prototipagem): envolve a criação de incrementos experimentais, alguns dos quais podem ser muito retrabalhados ou até mesmo abandonados em trabalhos subsequentes de desenvolvimento.

Componentes ou sistemas desenvolvidos usando esses métodos geralmente envolvem a sobreposição e a interação dos níveis de teste ao longo do desenvolvimento. Idealmente, cada recurso é testado em vários níveis de teste à medida que se aproxima da entrega. Em alguns casos, as equipes usam entrega contínua ou implantação contínua, as quais envolvem automação significativa de vários níveis de teste como parte de seus canais de entrega. Muitos esforços de desenvolvimento usando esses métodos também incluem o conceito de equipes de auto-organização, que podem mudar a maneira como o trabalho de teste é organizado, bem como a relação entre testadores e desenvolvedores.

Esses métodos formam um sistema crescente, que pode ser liberado para os usuários finais em uma base de recurso por recurso, em uma iteração por iteração, ou em uma moda de liberação principal mais tradicional. Independentemente de os incrementos de software serem liberados para os usuários finais, o teste de regressão é cada vez mais importante à medida que o sistema cresce.

Em contraste com os modelos sequenciais, os modelos iterativo e incremental podem fornecer software utilizável em semanas ou até dias, mas podem entregar o conjunto completo de requisitos ao longo de meses ou até anos.

Para obter mais informações sobre o teste de software no contexto do desenvolvimento ágil, consulte o [ISTQB_FL_AT], Black (2017), Crispin (2008) e Gregory (2015).

### 2.1.2 Modelos de ciclo de vida de desenvolvimento de software em contexto

Os modelos de ciclo de vida de desenvolvimento de software devem ser selecionados e adaptados ao contexto das características do projeto e do produto. Um modelo de ciclo de vida de desenvolvimento de software apropriado deve ser selecionado e adaptado com base na meta do projeto, no tipo de produto que está sendo desenvolvido, nas prioridades de negócios (p. ex., no tempo de fabricação) e nos riscos identificados de produto e projeto. Por exemplo, o desenvolvimento e teste de um pequeno sistema administrativo interno deve diferir do desenvolvimento e teste de um sistema crítico de segurança, como o sistema de controle de freio de um automóvel. Como outro exemplo, em alguns casos, questões organizacionais e culturais podem inibir a comunicação entre os membros da equipe, o que pode impedir o desenvolvimento iterativo.

Dependendo do contexto do projeto, pode ser necessário combinar ou reorganizar os níveis e/ou as atividades de teste. Por exemplo, na integração de um produto de software comercial de prateleira (COTS) em um sistema maior, o comprador pode realizar testes de interoperabilidade no nível de teste de integração do sistema (p. ex., integração à infraestrutura e outros sistemas) e nível de teste de aceite (funcional e não funcional, juntamente com os testes de aceite do usuário e operacional). Veja o capítulo 2.2 para uma discussão dos níveis de teste e o capítulo 2.3 para uma discussão dos tipos de teste.

Além disso, os próprios modelos de ciclo de vida de desenvolvimento de software podem ser combinados. Por exemplo, um Modelo V pode ser usado para o desenvolvimento e teste dos sistemas de apoio e suas integrações, enquanto um modelo de desenvolvimento ágil pode ser usado para desenvolver e testar a interface do usuário e a funcionalidade. A prototipagem pode ser usada no início de um projeto, com um modelo de desenvolvimento incremental adotado uma vez que a fase experimental esteja completa.

Em sistemas de internet das coisas (IoT), que consistem em muitos objetos diferentes, como dispositivos, produtos e serviços, geralmente aplicam modelos de ciclo de vida de desenvolvimento de software separados para cada objeto. Isso representa um desafio particular para o desenvolvimento de versões do sistema. Além disso, o ciclo de vida de desenvolvimento de software de tais objetos coloca uma ênfase mais forte nas fases posteriores do ciclo depois de terem sido introduzidos no uso operacional (p. ex., fases de operação, atualização e desativação).

Os motivos pelos quais os modelos de desenvolvimento de software devem ser adaptados ao contexto das características do projeto e do produto podem ser:

- Diferença nos riscos de produto dos sistemas (projeto complexo ou simples);
- Muitas unidades de negócios podem fazer parte de um projeto ou programa (combinação de desenvolvimento sequencial e ágil);
- Pouco tempo para entregar um produto ao mercado (mesclagem de níveis de teste ou integração de tipos de teste nos níveis de teste).

## 2.2 Níveis de teste

Os níveis de teste são grupos de atividades de teste que são organizados e gerenciados juntos. Cada nível de teste é uma instância do processo de teste, consistindo nas atividades descritas no capítulo 1.4, executadas em relação ao software em um determinado nível de desenvolvimento, desde as unidades individuais ou componentes até os sistemas completos ou, quando aplicável, em sistemas de sistemas. Os níveis de teste estão relacionados com outras atividades dentro do ciclo de vida de desenvolvimento de software. Os níveis de teste usados neste syllabus são:

- Teste de componentes;
- Teste de integração;
- Teste do sistema;
- Teste de aceite.

Os níveis de teste são caracterizados pelos seguintes atributos:

- Objetivos específicos;
- Base de teste, referenciada para derivar casos de teste;
- Objeto de teste (ou seja, o que está sendo testado);
- Defeitos e falhas típicas;
- Abordagens e responsabilidades específicas.

Para cada nível de teste, é necessário um ambiente de teste adequado. No teste de aceite, por exemplo, um ambiente de teste com características mais próximas do ambiente de produção é ideal, enquanto no teste de componentes os desenvolvedores normalmente usam seu próprio ambiente de desenvolvimento.

### 2.2.1 Teste de componente

#### Objetivos do teste de componente

O teste de componente (também conhecido como teste de unidade ou módulo) se concentra em componentes que são testáveis separadamente. Os objetivos incluem:

- Reduzir o risco;
- Verificar os comportamentos funcional e não funcional do componente ao projetado e especificado;

- Construir a confiança na qualidade do componente;
- Encontrar defeitos no componente;
- Evitar que os defeitos espalhem para níveis mais altos de teste.

Em alguns casos, especialmente em modelos de desenvolvimento incremental e iterativo (p. ex., Ágil), em que as alterações de código estão em andamento, os testes de regressão de componentes automatizados desempenham um papel fundamental na criação da confiança de que as alterações não impactaram os componentes existentes.

O teste de componente geralmente é realizado isoladamente do resto do sistema, dependendo do modelo de ciclo de vida de desenvolvimento de software e de sistema, que pode exigir objetos simulados, virtualização de serviços, equipamentos, simuladores e controladores. O teste de componente pode cobrir funcionalidade (p. ex., correção de cálculos), características não funcionais (p. ex., busca de vazamentos de memória) e propriedades estruturais (p. ex., teste de decisão).

### Base de teste

Exemplos de produtos de trabalho que podem ser usados como base de teste para testes de componentes incluem:

- Projeto detalhado;
- Código;
- Modelo de dados;
- Especificações de componentes.

### Objetos de teste

Objetos de teste típicos para testes de componentes incluem:

- Componentes, unidades ou módulos;
- Estruturas de código e dados;
- Classes;
- módulos de banco de dados.

### Defeitos típicos e falhas

Exemplos de defeitos e falhas típicas para o teste de componentes incluem:

- Funcionalidade incorreta (p. ex., não conformidade descrita nas especificações do projeto);
- Problemas no fluxo de dados;
- Código e lógica incorretos.

Os defeitos geralmente são corrigidos assim que são encontrados, geralmente sem gerenciamento formal dos defeitos. No entanto, quando os desenvolvedores relatam defeitos, isso fornece informações importantes para análise de causa raiz e melhoria de processo.

### Abordagens e responsabilidades específicas

O teste de componente geralmente é executado pelo desenvolvedor que escreveu o código, mas obrigatoriamente requer acesso ao código que está sendo testado. Os desenvolvedores podem

alternar o desenvolvimento de componentes com a localização e correção de defeitos. Os desenvolvedores geralmente escrevem e executam testes depois de codificarem um componente. No entanto, especialmente no desenvolvimento ágil, escrever os casos de teste de componentes automatizados pode preceder a gravação do código do aplicativo.

Por exemplo, considere o desenvolvimento orientado por teste (TDD). Esse desenvolvimento é altamente iterativo e baseado em ciclos de desenvolvimento de casos de teste automatizados, em seguida, na criação e integração de pequenos trechos de código, executando os testes de componentes, corrigindo quaisquer problemas e regerando o código. Esse processo continua até que o componente tenha sido completamente construído e todos os testes de componentes tenham passado. O desenvolvimento orientado por teste é um exemplo de uma abordagem de teste inicial. Embora ele tenha se originado no *eXtreme Programming* (XP), se espalhou para outras formas de desenvolvimento ágil, e para ciclos de vida sequenciais (consulte [ISTQB_FL_AT]).

## 2.2.2 Teste de integração

### Objetivos do teste de integração

O teste de integração se concentra nas interações entre componentes ou sistemas. Os objetivos do teste de integração incluem:

- Reduzir risco;
- Verificar se os comportamentos funcionais e não funcionais das interfaces estão projetados e especificados;
- Construir confiança na qualidade das interfaces;
- Encontrar defeitos (que podem estar nas próprias interfaces ou nos componentes ou sistemas);
- Evitar que os defeitos espalhem para níveis mais altos de teste.

Assim como no teste de componentes, em alguns casos, o teste de regressão de integração automatizada garante que as alterações não interromperam as interfaces, os componentes ou sistemas existentes.

Existem dois níveis diferentes de teste de integração descritos neste *syllabus*, que podem ser realizados em objetos de teste de tamanho variável, como segue:

- O teste de integração de componentes foca nas interações e interfaces entre componentes integrados. O teste de integração de componentes é executado após o teste do componente e geralmente é automatizado. No desenvolvimento iterativo e incremental, os testes de integração de componentes geralmente fazem parte do processo de integração contínua;
- O teste de integração do sistema concentra-se nas interações e interfaces entre sistemas, pacotes e microserviços. O teste de integração do sistema também pode abranger interações e interfaces fornecidas por entidades externas (p. ex., serviços da Web). Nesse caso, a entidade em desenvolvimento não controla as interfaces externas, o que pode criar vários desafios para testes (p. ex., garantir que os defeitos de bloqueio de teste no código da organização externa sejam resolvidos, organizar ambientes de teste etc.). O teste de integração do sistema pode

ser feito após o teste do sistema ou em paralelo com as atividades de teste do sistema em andamento (tanto no desenvolvimento sequencial quanto no desenvolvimento iterativo e incremental).

### Base de teste

Exemplos de produtos de trabalho que podem ser usados como base de teste para testes de integração incluem:

- Software e modelagem do sistema;
- Diagramas de sequência;
- Especificações de interface e protocolo de comunicação;
- Casos de uso;
- Arquitetura no nível de componente ou sistema;
- Fluxos de trabalho;
- Definições de interface externa.

### Objetos de teste

Objetos de teste típicos para testes de integração incluem:

- Subsistemas;
- Bancos de dados;
- Infraestrutura;
- Interfaces;
- APIs;
- Micro-serviços.

### Defeitos típicos e falhas

Os exemplos de defeitos e falhas típicos para testes de integração de componentes incluem:

- Dados incorretos, dados ausentes ou codificação de dados incorreta;
- Sequenciamento ou temporização incorretos de chamadas de interface;
- Incompatibilidade de interface;
- Falhas na comunicação entre componentes;
- Falha de comunicação não manipulada ou tratada de forma errada entre componentes;
- Suposições incorretas sobre o significado, as unidades ou limites dos dados que estão sendo transmitidos entre os componentes.

Os exemplos de defeitos e falhas típicos para testes de integração de sistemas incluem:

- Estruturas de mensagens inconsistentes entre sistemas;
- Dados incorretos, dados ausentes ou codificação de dados incorreta;
- Incompatibilidade de interface;
- Falhas na comunicação entre sistemas;
- Falha de comunicação não manipulada ou tratada de forma errada entre sistemas;

- Suposições incorretas sobre o significado, unidades ou limites dos dados que estão sendo transmitidos entre sistemas;
- Falha no cumprimento dos regulamentos de segurança obrigatórios.

### Abordagens e responsabilidades específicas

Os testes de integração de componentes e os testes de integração de sistemas devem se concentrar na própria integração. Por exemplo, se integrar o módulo A com o módulo B, os testes devem focar na comunicação entre ambos e não na funcionalidade dos módulos individualmente, como deveria ter sido coberto durante o teste de componentes. Se integrar o sistema X ao sistema Y, os testes devem focar na comunicação entre os sistemas, não na funcionalidade dos sistemas individualmente, como isso deveria ter sido coberto durante o teste do sistema. Tipos de testes funcionais, não funcionais e estruturais são aplicáveis.

O teste de integração de componentes geralmente é de responsabilidade dos desenvolvedores, e o teste de integração do sistema geralmente é de responsabilidade dos testadores. No formato ideal, os testadores que realizam o teste de integração do sistema devem entender da arquitetura do sistema e devem ter influenciado no planejamento da integração.

Se os testes de integração e a estratégia de integração forem planejados antes de componentes ou sistemas serem construídos, esses componentes ou sistemas podem ser construídos na ordem exigida para a maioria dos testes eficientes. Estratégias sistemáticas de integração podem ser baseadas na arquitetura do sistema (p.e, de cima para baixo e de baixo para cima), tarefas funcionais, sequências de processamento de transações ou algum outro aspecto do sistema ou componentes. Para simplificar o isolamento de defeitos e detectar defeitos precocemente, a integração deve normalmente ser incremental (ou seja, um pequeno número de componentes adicionais ou sistemas por vez) em vez de "*big bang*" (ou seja, integrar todos os componentes ou sistemas em uma única etapa). Uma análise de risco das interfaces mais complexas pode ajudar a focar o teste de integração.

Quanto maior o escopo da integração, mais difícil se torna isolar os defeitos em um componente ou sistema específico, o que pode levar a um aumento do risco e a um tempo adicional para a solução de problemas. Esse é um dos motivos pelos quais a integração contínua, onde o software é integrado componente por componente (ou seja, integração funcional), tornou-se prática comum. Essa integração contínua geralmente inclui testes de regressão automatizados, preferencialmente em vários níveis de teste.

## 2.2.3 Teste de sistema

### Objetivos do teste de sistema

O teste de sistema se concentra no comportamento e nas capacidades de todo um sistema ou produto, geralmente considerando as execuções das tarefas de ponta a ponta do sistema e os comportamentos não funcionais exibidos ao executar tais tarefas. Os objetivos do teste do sistema incluem:

- Reduzir o risco;

- Verificar se os comportamentos funcionais e não funcionais do sistema estão como projetados e especificados;
- Validar se o sistema está completo e funcionará como esperado;
- Criar confiança na qualidade do sistema como um todo;
- Encontrar defeitos;
- Evitar que os defeitos espalhem para níveis mais altos de teste ou produção.

Para determinados sistemas, a verificação da qualidade dos dados pode ser um objetivo. Assim como no teste de componente e no teste de integração, e em alguns casos os testes automatizados de regressão do sistema fornecem a confiabilidade de que as alterações não quebraram os recursos existentes ou a execução dos recursos de ponta a ponta. O teste do sistema geralmente produz informações que são usadas pelos *stakeholders* para tomar decisões de liberação. O teste do sistema também pode satisfazer requisitos ou padrões legais e regulatórios.

O ambiente de teste deve preferencialmente corresponder ao seu destino, ou ao ambiente de produção.

### Base de teste

Os exemplos de produtos de trabalho que podem ser usados como base de teste para teste de sistema incluem:

- Especificações de requisitos de sistema e software (funcionais e não funcionais);
- Relatórios de análise de risco;
- Casos de uso;
- Épicos e histórias de usuários;
- Modelos de comportamento do sistema;
- Diagramas de estado;
- Sistema e manuais do usuário;

### Objetos de teste

Os objetos de teste típicos para teste de sistema incluem:

- Aplicações;
- Sistemas de hardware e software;
- Sistemas operacionais;
- Sistema sob teste (SUT);
- Configuração do sistema e dados de configuração.

### Defeitos típicos e falhas

Os exemplos de defeitos e falhas típicas para o teste de sistema incluem:

- Cálculos incorretos;
- Comportamento funcional ou não funcional do sistema incorreto ou inesperado;
- Controle e/ou fluxos de dados dentro do sistema incorretos;
- Falha na execução correta e completa de tarefas funcionais de ponta a ponta;

- Falha do sistema em funcionar adequadamente no(s) ambiente(s) de produção;
- Falha do sistema para funcionar conforme descrito nos manuais do sistema e do usuário.

### Abordagens e responsabilidades específicas

O teste de sistema deve focar no comportamento geral, funcional ou não, de ponta a ponta do sistema como um todo. O teste de sistema deve usar as técnicas mais apropriadas (ver capítulo 4) para o(s) aspecto(s) do sistema a ser testado. Por exemplo, uma tabela de decisão pode ser criada para verificar se o comportamento funcional é conforme descrito nas regras de negócios.

Testadores independentes geralmente realizam testes no sistema. Defeitos nas especificações (p. ex., histórias de usuários ausentes, requisitos de negócios incorretamente declarados etc.) podem levar a uma falta de compreensão ou desacordo sobre o comportamento esperado do sistema. Tais situações podem causar falsos positivos e falsos negativos, que perdem tempo e reduzem a eficácia da detecção de defeitos, respectivamente. O envolvimento precoce de testadores no refinamento da história do usuário ou em atividades de testes estáticos, como revisões, ajuda a reduzir a incidência de tais situações.

## 2.2.4 Teste de aceite

### Objetivos do teste de aceite

O teste de aceite, como o teste do sistema, geralmente se concentra no comportamento e na capacidade de todo um sistema ou produto. Os objetivos do teste de aceite incluem:

- Estabelecer confiança na qualidade do sistema como um todo;
- Validar que o sistema está completo e funcionará como esperado;
- Verificar se os comportamentos funcionais e não funcionais do sistema são os especificados.

O teste de aceite pode produzir informações para avaliar a situação do sistema para implantação e uso pelo cliente (usuário final). Os defeitos podem ser encontrados durante o teste de aceite, mas encontrar defeitos muitas vezes não é um objetivo, e encontrar um número significativo de defeitos durante o teste de aceite pode, em alguns casos, ser considerado um grande risco de projeto. O teste de aceite também pode satisfazer requisitos ou padrões legais ou regulatórios.

Formas comuns de testes de aceite incluem o seguinte:

- Teste de aceite do usuário;
- Teste de aceite operacional;
- Teste de aceite contratual e regulatório;
- Alfa teste e Beta.

Cada um é descrito nas quatro subseções a seguir.

### Teste de aceite do usuário (UAT)

O teste de aceite do sistema pelos usuários é tipicamente focado em validar a adequação do uso do sistema pelos usuários pretendidos em um ambiente operacional real ou simulado. O objetivo principal é desenvolver a confiança de que os usuários podem usar o sistema para atender às suas

necessidades, atender aos requisitos e executar processos de negócios com o mínimo de dificuldade, custo e risco.

### Teste de aceite operacional (OAT)

O teste de aceite do sistema pelas operações ou pela equipe de administração de sistemas geralmente é realizado em um ambiente de produção (simulado). Os testes se concentram em aspectos operacionais e podem incluir:

- Teste de backup e restauração;
- Instalar, desinstalar e atualizar;
- Recuperação de desastres;
- Gerenciamento de usuários;
- tarefas de manutenção;
- Carregamento de dados e tarefas de migração;
- Verifica vulnerabilidades de segurança;
- Teste de performance.

O principal objetivo do teste de aceite operacional é criar confiança de que os operadores ou administradores do sistema possam manter o sistema funcionando adequadamente para os usuários no ambiente operacional, mesmo sob condições excepcionais ou difíceis.

### Teste de aceite contratual e regulatório

O teste de aceite contratual é realizado com base nos critérios de aceite de um contrato para desenvolver softwares específicos. Os critérios de aceite devem ser definidos quando as partes concordam com o contrato. O teste de aceite contratual é frequentemente realizado por usuários ou por testadores independentes.

O teste de aceite regulatório é realizado em relação a quaisquer regulamentos que devem ser seguidos, como governamentais, legais ou de segurança. O teste de aceite regulatório é frequentemente realizado por usuários ou por testadores independentes, às vezes com os resultados sendo testemunhados ou auditados por agências reguladoras.

O principal objetivo do teste de aceite contratual e regulatório é a criação de confiança de que a conformidade contratual ou regulatória foi alcançada.

### Alfa e beta teste

Os "*alfa*" e "*beta*" teste são normalmente usados por desenvolvedores de software comercial de prateleira (COTS) que desejam obter feedback de usuários, clientes ou operadores em potencial ou existentes antes que o produto de software seja colocado no mercado. O alfa teste é realizado no site da organização em desenvolvimento, não pela equipe de desenvolvimento, mas por clientes em potencial, existentes, operadores, ou por uma equipe de teste independente. O beta teste é realizado por clientes em potencial, existentes, operadores em seus próprios locais. O beta teste pode ocorrer após o alfa teste ou sem que este seja realizado.

Um objetivo dos testes alfa e beta é a construção da confiança entre os clientes ou operadores de que podem usar o sistema em condições normais no dia-a-dia para atingir seus objetivos com o mínimo de dificuldade, custo e risco. Outro objetivo pode ser a detecção dos defeitos relacionados às condições e ambiente(s) em que o sistema será utilizado, especialmente quando essas são difíceis de replicar pela equipe de desenvolvimento.

### Base de teste

Os exemplos de produtos de trabalho que podem ser usados como base de teste para qualquer forma de teste de aceite incluem:

- Processos de negócios;
- Requisitos do usuário ou de negócios;
- Regulamentos, contratos legais e normas;
- Casos de uso;
- Requisitos de sistema;
- Documentação do sistema ou usuário;
- Procedimentos de instalação;
- Relatórios de análise de risco.

Além disso, como base de teste para derivar casos de teste para testes de aceite operacional, um ou mais dos seguintes produtos de trabalho podem ser usados:

- Procedimentos de *backup* e restauração;
- Procedimentos de recuperação de desastres;
- Requisitos não funcionais;
- Documentação de operações;
- Instruções de implantação e instalação;
- Metas de performance;
- Pacotes de banco de dados;
- Normas ou regulamentos de segurança.

### Objetos de teste típicos

Os objetos de teste típicos para qualquer forma de teste de aceite incluem:

- Sistema sob teste;
- Configuração do sistema e dados de configuração;
- Processos de negócios para um sistema totalmente integrado;
- Sistemas de recuperação e hot sites (para continuidade de negócios e testes de recuperação de desastres);
- Processos operacionais e de manutenção;
- Formulários;
- Relatórios;
- Dados de produção existentes e convertidos.

### Defeitos típicos e falhas

Os exemplos de defeitos típicos para qualquer forma de teste de aceite incluem:

- Fluxos de trabalho do sistema não atendem aos requisitos do negócio ou do usuário;
- Regras de negócios não são implementadas corretamente;
- O sistema não satisfaz os requisitos contratuais ou regulatórios;
- Falhas não funcionais, como vulnerabilidades de segurança, eficiência de performance inadequada sob altas cargas ou operação inadequada em uma plataforma suportada.

### Abordagens e responsabilidades específicas

O teste de aceite é geralmente responsabilidade dos clientes, usuários de negócios, proprietários de produtos ou operadores de um sistema, além de *stakeholders* que também podem estar envolvidos.

O teste de aceite é frequentemente considerado como o último nível de teste em um ciclo de vida de desenvolvimento sequencial, mas também pode ocorrer em outros momentos, por exemplo:

- O teste de aceite de um produto de software de prateleira (COTS) pode ocorrer quando instalado ou integrado;
- O teste de aceite de um novo aprimoramento funcional pode ocorrer antes do teste do sistema.

No desenvolvimento iterativo, as equipes de projeto podem empregar várias formas de testes de aceite durante e ao final de cada iteração, como aqueles focados em verificar um novo recurso em relação aos critérios de aceite e aqueles focados em validar que um novo recurso satisfaz as necessidades dos usuários. Além disso, o alfa e beta teste podem ocorrer, no final de cada iteração, após a conclusão de cada iteração ou após uma série de iterações. Teste de aceite do usuário, testes de aceite operacional, testes de aceite regulatória e testes de aceite contratual também podem ocorrer, seja no final de cada iteração, após a conclusão de cada iteração ou após uma série de iterações.

## 2.3 Tipos de teste

Um tipo de teste é um grupo de atividades de teste destinado a testar características específicas de um sistema de software, ou parte de um sistema, com base em objetivos de teste específicos. Tais objetivos podem incluir:

- Avaliar as características de qualidade funcional, tais como integridade, correção e adequação;
- Avaliar as características de qualidade não funcionais, como confiabilidade, eficiência de performance, segurança, compatibilidade e usabilidade;
- Avaliar se a estrutura ou arquitetura do componente ou sistema está correta, completa e especificada;
- Avaliar os efeitos das alterações, como a confirmação da correção dos defeitos (teste de confirmação) e procurar alterações não intencionais no comportamento como resultado de alterações no software ou no ambiente (teste de regressão).

### 2.3.1 Teste funcional

O teste funcional de um sistema envolve testes que avaliam as funções que o sistema deve executar. Os requisitos funcionais podem ser descritos em produtos de trabalho, como especificações de requisitos, de negócios, épicos, histórias de usuários, casos de uso ou especificações funcionais, podendo ainda não estarem documentados. As funções são “o que” o sistema deve fazer.

Os testes funcionais devem ser realizados em todos os níveis de teste (p.e, testes para componentes podem ser baseados em uma especificação de componente), embora o foco possa ser diferente em cada nível (ver capítulo 2.2).

O teste funcional considera o comportamento do software, portanto, técnicas caixa-preta podem ser usadas para derivar condições de teste e casos de teste para a funcionalidade do componente ou sistema (ver capítulo 4.2).

A eficácia dos testes funcionais pode ser medida através da cobertura funcional. A cobertura funcional é a medida em que algum tipo de elemento funcional foi exercido por testes e é expresso como uma porcentagem do(s) tipo(s) de elemento a ser coberto. Por exemplo, usando a rastreabilidade entre os testes e os requisitos funcionais, a porcentagem desses requisitos que são abordados pelos testes pode ser calculada, identificando potencialmente as lacunas de cobertura.

O projeto e a execução de testes funcionais podem envolver habilidades ou conhecimentos especiais, como o conhecimento específico de um problema de negócios que o software resolve (p. ex., software de modelagem geológica para as indústrias de petróleo e gás) ou o papel específico que o software desempenha (p. ex., fornecendo entretenimento interativo).

### 2.3.2 Teste não funcional

Os testes não funcionais de um sistema avaliam as características de sistemas e de softwares, como usabilidade, eficiência de performance ou segurança. Consulte o padrão [ISO25010] para obter uma classificação das características de qualidade do produto de software. O teste não funcional é o teste de "quão bem" o sistema se comporta.

Ao contrário das percepções errôneas comuns, o teste não-funcional pode e geralmente deve ser realizado em todos os níveis de teste, e feito o mais cedo possível. A descoberta tardia de defeitos não funcionais pode ser extremamente perigosa para o sucesso de um projeto.

As técnicas caixa-preta (ver capítulo 4.2) podem ser usadas para derivar condições de teste e casos de teste para testes não funcionais. Por exemplo, a análise do valor limite pode ser usada para definir as condições de tensão para testes de performance.

A eficácia dos testes não funcionais pode ser medida por meio de cobertura não funcional. A cobertura não funcional é a medida em que algum tipo de elemento não funcional foi exercido por testes e é expressa como uma porcentagem do tipo de elemento a ser coberto. Por exemplo, usando a rastreabilidade entre testes e dispositivos suportados para um aplicativo móvel, a porcentagem de dispositivos que são abordados pelo teste de compatibilidade pode ser calculada, identificando potencialmente as lacunas de cobertura.

A modelagem e execução de testes não funcionais podem envolver habilidades ou conhecimentos especiais, como o conhecimento das fraquezas inerentes a um projeto ou tecnologia (p. ex., vulnerabilidades de segurança associadas a determinadas linguagens de programação) ou a uma base de usuários específica (p. ex., os usuários dos funcionários de sistemas de gerenciamento de unidades de saúde).

Consulte os syllabi de teste do nível avançado [ISTQB_AL_TA], [ISTQB_AL_TTA], [ISTQB_AL_SEC], e outros módulos especializados do *ISTQB®* para obter mais detalhes sobre o teste de características de qualidade não funcionais.

### 2.3.3 Teste caixa-branca

O teste caixa-branca é derivado de testes com base na estrutura interna ou na implementação do sistema. A estrutura interna pode incluir código, arquitetura, fluxos de trabalho e fluxos de dados dentro do sistema (ver capítulo 4.3).

A eficácia dos testes caixa-branca pode ser medida através da cobertura estrutural. A cobertura estrutural é a extensão em que algum tipo de elemento estrutural foi testado e é expresso como uma porcentagem do tipo de elemento a ser coberto.

No nível de teste de componente, a cobertura de código é baseada na porcentagem do código do componente que foi testado e pode ser medida em termos de aspectos diferentes do código (itens de cobertura), como a porcentagem de instruções executáveis que foram testadas no componente ou a porcentagem dos resultados da decisão que foram testados. Esses tipos de cobertura são chamados coletivamente de cobertura de código. No nível de teste de integração de componentes, o teste caixa-branca pode ser baseado na arquitetura do sistema, como interfaces entre componentes, e a cobertura estrutural pode ser medida em termos da porcentagem de interfaces exercidas pelos testes.

A modelagem e a execução do teste caixa-branca podem envolver habilidades ou conhecimentos especiais, como a maneira como o código é construído (p. ex., usar ferramentas de cobertura de código), como os dados são armazenados (p. ex., avaliar possíveis consultas de banco de dados) e como usar ferramentas de cobertura interpretando corretamente seus resultados.

### 2.3.4 Teste relacionado à mudança

Quando são feitas alterações em um sistema, seja para corrigir um defeito ou por causa de uma funcionalidade nova ou variável, deve-se testar para confirmar se as alterações corrigiram o defeito ou implementaram a funcionalidade corretamente e não causaram consequências adversas imprevistas.

- **Teste de confirmação**: Depois que um defeito é corrigido, o software pode ser testado com todos os casos de teste que falharam devido ao defeito, que devem ser executados novamente na nova versão do software. O software também pode ser testado com novos testes se, por exemplo, para um defeito estiver faltando uma funcionalidade. No mínimo, as etapas para reproduzir as falhas causadas pelo defeito devem ser executadas novamente na nova versão

do software. A finalidade de um teste de confirmação é confirmar se o defeito original foi corrigido com sucesso.

- **Teste de regressão**: É possível que uma alteração feita em uma parte do código, seja uma correção ou outro tipo de alteração, possa afetar acidentalmente o comportamento de outras partes do mesmo código, seja dentro do mesmo componente, em outros componentes do código. mesmo sistema, ou mesmo em outros sistemas. As alterações podem incluir alterações no ambiente, como uma nova versão de um sistema operacional ou sistema de gerenciamento de banco de dados. Tais efeitos colaterais não intencionais são chamados de regressões. O teste de regressão envolve a execução de testes para detectar esses efeitos colaterais indesejados.

O teste de confirmação e o teste de regressão são realizados em todos os níveis de teste.

Especialmente em ciclo de vida de desenvolvimento iterativo e incremental (p. ex., Ágil), novos recursos, alterações nos recursos existentes e recompilação de código resultam em alterações frequentes no código, o que também requer testes relacionados a estas alterações. Devido à natureza evolutiva do sistema, os testes de confirmação e regressão são muito importantes. Isso é particularmente relevante para os sistemas da Internet das Coisas (IoT), nos quais objetos individuais (p. ex., dispositivos) são atualizados ou substituídos com frequência.

Os conjuntos de testes de regressão são executados muitas vezes e geralmente evoluem lentamente, portanto, o teste de regressão é um forte candidato à automação. A automação desses testes deve começar no início do projeto (ver capítulo 6).

### 2.3.5 Tipos e níveis de teste

É possível executar qualquer um dos tipos de teste mencionados acima em qualquer nível de teste. Para ilustrar, exemplos de testes funcionais, não funcionais, caixa-branca e relacionados a alterações serão fornecidos em todos os níveis de teste, para um aplicativo bancário, começando com testes funcionais:

- No teste de componente, os testes são modelados com base em como um componente deve calcular os juros compostos;
- No teste de integração de componentes, os testes são modelados com base em como os dados da conta capturada na interface do usuário são transmitidas para a lógica de negócios;
- No teste de sistema, os testes são modelados com base em como os titulares de conta podem solicitar uma linha de crédito em suas contas correntes;
- No teste de integração de sistemas, os testes são modelados com base em como o sistema usa um microserviço externo para verificar a pontuação de crédito de um titular de conta;
- No teste de aceite, os testes são modelados com base em como o banqueiro manipula a aprovação ou não de um pedido de crédito.

Exemplos de testes não funcionais:

- No teste de componente, os testes de performance são modelados para avaliar o número de ciclos de CPU necessários para realizar um cálculo complexo de juros total;

- No teste de integração de componentes, os testes de segurança são modelados para vulnerabilidades de estouro de buffer devido aos dados transmitidos da interface do usuário para a lógica de negócios;
- No teste de sistema, os testes de portabilidade são modelados para verificar se a camada de apresentação funciona em todos os navegadores e dispositivos móveis suportados;
- No teste de integração de sistemas, os testes de confiabilidade são modelados para avaliar a robustez do sistema se o microserviço de pontuação de crédito não responder;
- No teste de aceite, os testes de usabilidade são modelados para avaliar a acessibilidade da interface de processamento de crédito do banco para pessoas com deficiências.

Exemplos de testes caixa-branca:

- No teste de componentes, os testes são modelados para obter uma cobertura completa de instruções e decisões (capítulo 4.3) para todos os componentes que executam cálculos financeiros;
- No o teste de integração de componentes, os testes são modelados para exercitar como cada tela na interface do navegador passa os dados para a próxima tela e para a lógica de negócios;
- No teste de sistema, os testes são modelados para cobrir sequências de páginas da web que podem ocorrer durante um aplicativo de linha de crédito;
- No teste de integração do sistema, os testes são modelados para exercer todos os possíveis tipos de consulta enviados ao microserviço de pontuação de crédito;
- No teste de aceite, os testes são modelados para cobrir todas as estruturas de arquivos de dados financeiros suportados e intervalos de valores para transferências bancárias para bancos.

Exemplos de testes relacionados a mudanças:

- No teste de componentes, os testes de regressão automatizados são modelados para cada componente e incluídos na estrutura de integração contínua;
- No teste de integração de componentes, os testes são modelados para confirmar as correções dos defeitos relacionados à interface, à medida que as correções são verificadas no repositório de código;
- No teste do sistema, todos os testes para um determinado fluxo de trabalho são executados novamente se qualquer tela desse fluxo de trabalho for alterada;
- No teste de integração do sistema, os testes do aplicativo interagindo com o micro-serviço de pontuação de crédito são executados diariamente como parte da implantação contínua desse micro serviço;
- No teste de aceite, todos os testes com falhas anteriores são executados novamente após a correção de um defeito encontrado no teste de aceite.

Embora este capítulo forneça exemplos de todos os tipos de testes em todos os níveis, não é necessário, para todos os softwares, ter todos os tipos de testes representados em todos os níveis. No entanto, é importante executar os tipos de teste aplicáveis em cada nível, especialmente no nível mais antigo em que o tipo de teste ocorre.

## 2.4 Teste de manutenção

Depois de implantados em ambientes de produção, o software e os sistemas precisam ser mantidos. Vários tipos de mudanças são quase inevitáveis em softwares e sistemas entregues, seja para corrigir defeitos descobertos no uso operacional, para adicionar novas funcionalidades, ou para remover ou alterar funcionalidades já entregues. A manutenção também é necessária para preservar ou melhorar as características de qualidade não funcionais do componente ou do sistema ao longo de sua vida útil, especialmente a eficiência de performance, compatibilidade, confiabilidade, segurança e portabilidade.

Quando quaisquer alterações são feitas como parte da manutenção, testes de manutenção devem ser realizados, tanto para avaliar o sucesso com que as alterações foram feitas quanto para verificar possíveis efeitos colaterais (p. ex., regressões) em partes do sistema que permanecem inalteradas (o que geralmente é a maior parte do sistema). O teste de manutenção se concentra em testar as alterações no sistema, bem como em testar as partes inalteradas que podem ter sido afetadas pelas alterações. A manutenção pode envolver lançamentos planejados e liberações não planejadas (hot fixes).

Uma versão de manutenção pode exigir testes de manutenção em vários níveis de teste, usando vários tipos de teste, com base em seu escopo. O escopo do teste de manutenção depende do:

- Grau de risco da mudança, por exemplo, o grau em que a área alterada do software se comunica com outros componentes ou sistemas;
- Tamanho do sistema existente;
- Tamanho da mudança.

### 2.4.1 Gatilhos para manutenção

Existem várias razões pelas quais a manutenção de software e, portanto, os testes de manutenção, ocorrem tanto para alterações planejadas como não planejadas.

Podemos classificar os gatilhos para manutenção da seguinte forma:

- **Modificação**: como aprimoramentos planejados (p. ex., baseado na versão), alterações corretivas e emergenciais, alterações no ambiente operacional (como sistema operacional ou atualizações de banco de dados), atualizações de software de prateleira usados e correções para defeitos e vulnerabilidades;
- **Migração**: como de uma plataforma para outra, que pode exigir testes operacionais do novo ambiente, bem como do software alterado, ou testes de conversão de dados quando os dados de outro aplicativo serão migrados para o sistema que está sendo mantido;
- **Aposentadoria**: como quando um aplicativo atinge o fim de sua vida útil.

Quando um aplicativo ou sistema é retirado, isso pode exigir o teste de migração ou arquivamento dos dados, se forem necessários longos períodos de retenção para esses dados. Também pode ser necessário testar os procedimentos de recuperação após o arquivamento por longos períodos de

retenção. Além disso, testes de regressão podem ser necessários para garantir que qualquer funcionalidade que permaneça em serviço ainda funcione.

Para os sistemas da Internet das Coisas (IoT), o teste de manutenção pode ser acionado pela introdução de itens completamente novos ou modificados, como dispositivos de hardware e serviços de software, no sistema geral. O teste de manutenção para tais sistemas coloca ênfase particular nos testes de integração em diferentes níveis (p.e, nível de rede, nível de aplicação) e nos aspectos de segurança, em particular aqueles relacionados a dados pessoais.

### 2.4.2 Análise de impacto para manutenção

A análise de impacto avalia as alterações feitas para uma liberação de manutenção identificando os resultados esperados, os possíveis efeitos colaterais provocados e para identificar as áreas do sistema que serão afetadas. A análise de impacto também pode ajudar a identificar o impacto de uma mudança nos testes existentes. Os efeitos colaterais e áreas afetadas no sistema precisam ser testados por regressões, possivelmente após a atualização de quaisquer testes existentes afetados pela alteração.

A análise de impacto pode ser feita antes de uma mudança ser realizada, com base nas consequências potenciais em outras áreas do sistema para determinar se a mudança deve ser feita.

A análise de impacto pode ser difícil se:

- Especificações estão desatualizadas ou ausentes (p. ex., requisitos, histórias de usuários etc.);
- Casos de teste não estão documentados ou estão desatualizados;
- A rastreabilidade bidirecional entre os testes e a base de teste não foi mantida;
- O suporte das ferramentas é fraco ou inexistente;
- As pessoas envolvidas não possuem conhecimento do domínio ou sistema;
- Não foi dada atenção insuficiente à manutenção do software durante o seu desenvolvimento.

# 3 Teste estático [135 min]

### *Conceitos-chave*

revisão ad-hoc, revisão baseada em checklist, testes dinâmicos, revisão formal, revisão informal, inspeção, revisão baseada em perspectiva, revisão, revisão baseada na função, revisão baseada em cenário, análise estática, teste estático, revisão técnica, passo a passo

### *Objetivos de aprendizagem*

#### *3.1 Noções básicas sobre testes estáticos*

FL-3.1.1 (K1) Reconhecer os tipos de produtos de trabalho de software que podem ser examinados pelas diferentes técnicas de testes estáticos.

FL-3.1.2 (K2) Usar exemplos para descrever o valor do teste estático.

FL-3.1.3 (K2) Explicar a diferença entre técnicas estáticas e dinâmicas, considerando os objetivos, os tipos de defeitos a serem identificados e a função dessas técnicas no ciclo de vida do software.

#### *3.2 O processo de revisão*

FL-3.2.1 (K2) Resumir as atividades do processo de revisão do produto de trabalho.

FL-3.2.2 (K1) Reconhecer as diferentes funções e responsabilidades em uma revisão formal.

FL-3.2.3 (K2) Explicar as diferenças entre os diferentes tipos de revisão: revisão informal, passo a passo, revisão técnica e inspeção.

FL-3.2.4 (K3) Aplicar uma técnica de revisão a um produto de trabalho para encontrar defeitos.

FL-3.2.5 (K2) Explicar os fatores que contribuem para uma revisão bem-sucedida.

## 3.1 Noções básicas sobre testes estáticos

Em contraste com o teste dinâmico, que requer a execução do software a ser testado, o teste estático depende do exame manual dos produtos de trabalho (isto é, revisões) ou da avaliação orientada por ferramentas do código ou outros produtos de trabalho (isto é, análise estática). Os dois tipos de testes estáticos avaliam o código ou outro produto de trabalho que está sendo testado sem realmente executar o código ou o produto de trabalho que está sendo testado.

A análise estática é importante para sistemas críticos de segurança (p. ex., software de aviação, médico ou nuclear), mas a análise estática também se tornou importante e comum em outros ambientes. Por exemplo, a análise estática é uma parte importante dos testes de segurança. A análise estática também é frequentemente incorporada aos sistemas automatizados de criação e distribuição, por exemplo, no desenvolvimento ágil, na entrega contínua e na implantação contínua.

### 3.1.1 Produtos de trabalho que podem ser examinados por testes estáticos

Quase qualquer produto de trabalho pode ser examinado usando testes estáticos (revisões ou análise estática), por exemplo:

- Especificações, incluindo requisitos de negócios, requisitos funcionais e requisitos de segurança;
- Épicos, histórias de usuários e critérios de aceite;
- Especificações de arquitetura e design;
- Código;
- Testware, incluindo planos de teste, casos de teste, procedimentos de teste e scripts de teste automatizados;
- Guias de usuários;
- Páginas web;
- Contratos, planos de projeto, cronogramas e orçamentos;
- Modelos, como os diagramas de atividades, que podem ser usados para testes baseados em modelo (consulte [ISTQB_FL_MBT] e Kramer (2016))

As avaliações podem ser aplicadas a qualquer produto de trabalho que os participantes saibam ler e entender. A análise estática pode ser aplicada eficientemente a qualquer produto de trabalho com uma estrutura formal (normalmente código ou modelos) para a qual existe uma ferramenta de análise estática apropriada. A análise estática pode até ser aplicada com ferramentas que avaliam produtos de trabalho escritos em linguagem natural, como requisitos (p. ex., verificação de ortografia, gramática e legibilidade).

### 3.1.2 Benefícios do teste estático

As técnicas de testes estáticos fornecem uma variedade de benefícios. Quando aplicado no início do ciclo de vida de desenvolvimento de software, permite a detecção antecipada dos defeitos antes da realização dos testes dinâmicos (p. ex., em revisões de requisitos ou especificações de projeto, refinamento de backlog de produtos etc.). Os defeitos detectados precocmente costumam ser muito

mais baratos de serem removidos do que os defeitos encontrados posteriormente no ciclo de vida, especialmente em comparação aos defeitos encontrados após o software ser implantado e em uso ativo. Usar técnicas de testes estáticos para localizar e corrigir os defeitos rapidamente é quase sempre muito mais barato para a organização do que usar testes dinâmicos para encontrar defeitos no objeto de teste e corrigi-los, especialmente considerando os custos adicionais associados à atualização de outros produtos de trabalho e à realização de testes de confirmação e regressão.

Benefícios adicionais do teste estático podem incluir:

- Detectar e corrigir defeitos com mais eficiência e antes da execução dinâmica dos testes;
- Identificar os defeitos que não são facilmente encontrados por testes dinâmicos;
- Prevenir defeitos no desenho ou na codificação, revelando inconsistências, ambiguidades, contradições, omissões, imprecisões e redundâncias nos requisitos;
- Aumentar a produtividade no desenvolvimento (p. ex., devido ao desenho aprimorado, código mais sustentável);
- Reduzir o custo e o tempo de desenvolvimento;
- Reduzir o custo e o tempo de teste;
- Reduzir o custo total de qualidade durante a vida útil do software, devido a menos falhas nas etapas finais do ciclo de vida ou após a entrega em operação;
- Melhorar a comunicação entre os membros da equipe durante a participação nas revisões.

### 3.1.3 Diferenças entre teste estático e dinâmico

Os testes estático e dinâmico podem ter os mesmos objetivos (ver capítulo 1.1.1), como fornecer uma avaliação da qualidade dos produtos de trabalho e identificar os defeitos o mais cedo possível. Os testes estático e dinâmico se complementam, encontrando diferentes tipos de defeitos.

Uma das principais distinções é que o teste estático encontra os defeitos diretamente em produtos de trabalho, em vez de identificar as falhas causadas por defeitos quando o software é executado. Um defeito pode residir em um produto de trabalho por muito tempo sem causar uma falha. Se caminho onde o defeito está é difícil de alcançar ou de ser testado, então não será fácil construir e executar um teste dinâmico que o encontre. O Teste estático pode encontrar o defeito com muito menos esforço.

Outra distinção é que o teste estático pode ser usado para melhorar a consistência e a qualidade interna dos produtos de trabalho, enquanto o teste dinâmico geralmente se concentra em comportamentos externamente visíveis.

Em comparação com o teste dinâmico, os defeitos mais comuns que são mais fáceis e baratos de encontrar e corrigir por meio de teste estático inclue:

- Defeitos em requisitos (p. ex., inconsistências, ambiguidades, contradições, omissões, imprecisões e redundâncias);
- Defeitos de desenho (p. ex., algoritmos ineficientes ou estruturas de banco de dados, alto acoplamento, baixa coesão);

- Defeitos de codificação (p. ex., variáveis com valores indefinidos, variáveis declaradas e nunca utilizadas, código inacessível, código duplicado);
- Desvios dos padrões (p. ex., falta de aderência aos padrões de codificação);
- Especificações de interface incorretas (p. ex., unidades de medida diferentes usadas pelo sistema de chamada do que pelo sistema chamado);
- Vulnerabilidades de segurança (p. ex., suscetibilidade a estouro de buffer);
- Lacunas ou imprecisões na rastreabilidade ou cobertura da base de teste (p. ex., falta de testes para um critério de aceite).

Além disso, a maioria dos tipos de defeitos de manutenção só pode ser encontrada por testes estáticos (p. ex., modularização inadequada, reusabilidade ruim de componentes, código que é difícil de analisar e modificar sem introduzir novos defeitos).

## 3.2 Processo de revisão

As revisões variam de informal para formal. As revisões informais caracterizam-se por não seguir um processo definido e não ter um resultado formal documentado. Revisões formais são caracterizadas pela participação da equipe, com resultados documentados e procedimentos documentados para conduzir a revisão. A formalidade de um processo de revisão está relacionada a fatores como o modelo de ciclo de vida de desenvolvimento de software, a maturidade do processo de desenvolvimento, a complexidade do produto a ser revisado, e quaisquer requisitos legais ou regulatórios ou a necessidade de uma trilha de auditoria.

O foco de uma revisão depende dos objetivos acordados da revisão (p. ex., encontrar defeitos, obter entendimento, treinar os participantes, como testadores e novos membros da equipe, ou discutir e decidir por consenso).

O padrão [ISO20246] contém descrições mais detalhadas do processo de revisão de produtos de trabalho, incluindo funções e técnicas de revisão.

### 3.2.1 Processo de revisão do produto de trabalho

O processo de revisão compreende as seguintes atividades principais:

**Planejar**

- Definir o escopo, que inclui o propósito da revisão, quais documentos ou partes de documentos devem ser revisados e as características de qualidade a serem avaliadas;
- Estimar o esforço e o prazo;
- Identificar as características de revisão, como o tipo de revisão, as atividades e checklists;
- Selecionar as pessoas para participar da revisão e alocar funções;
- Definir os critérios de entrada e saída para os tipos de revisão mais formais (p. ex., inspeções);
- Verificar se os critérios de entrada foram atendidos (para tipos de revisão mais formais).

### Iniciar revisão

- Distribuir o produto de trabalho (fisicamente ou por meios eletrônicos) e outros materiais, como formulários de registro de problemas, listas de verificação e produtos de trabalho relacionados;
- Explicar o escopo, os objetivos, o processo, as funções e os produtos de trabalho aos participantes;
- Responder a quaisquer perguntas que os participantes possam ter sobre a revisão.

### Revisão individual (ou seja, preparação individual)

- Rever todo ou parte do produto de trabalho;
- Observar possíveis defeitos, recomendações e questões.

### Analisar e comunicar

- Comunicar possíveis defeitos identificados (p. ex., em uma reunião de revisão);
- Analisar possíveis defeitos, atribuindo responsável e status a eles;
- Avaliar e documentar características de qualidade;
- Avaliar as conclusões da revisão em relação aos critérios de saída para tomar uma decisão de revisão ("rejeitar" mudanças importantes necessárias, "aceitar" possivelmente com pequenas alterações).

### Corrigir e reportar

- Criar relatórios de defeitos para as descobertas que exigem mudanças;
- Corrigir os defeitos encontrados (normalmente feitos pelo autor) no produto de trabalho revisado;
- Comunicar os defeitos à pessoa ou equipe apropriada (quando encontrado em um produto de trabalho relacionado ao produto de trabalho revisado);
- Registrar o status atualizado dos defeitos (em revisões formais), principalmente incluindo o "de acordo" do autor do comentário;
- Métricas de coleta (para tipos de revisão mais formais);
- Verificar se os critérios de saída foram atendidos (para tipos de revisão mais formais);
- Aceitar o produto de trabalho quando os critérios de saída são atingidos.

Os resultados de uma revisão de produto de trabalho variam, dependendo do tipo de revisão e da formalidade, conforme descrito no capítulo 3.2.3.

## 3.2.2 Funções e responsabilidades em uma revisão formal

Uma revisão formal típica incluirá as funções a seguir:

### Autor

- Criar o produto de trabalho sob revisão;
- Corrigir os defeitos no produto de trabalho sob revisão (se necessário).

### Gestor

- Responsável pelo planejamento da revisão;
- Decidir sobre a execução das revisões;
- Atribuir pessoal, orçamento e tempo;
- Monitorar a rentabilidade contínua.

### Facilitador (geralmente chamado de moderador)

- Garantir a execução eficaz das reuniões de revisão (quando realizada);
- Mediar, se necessário, entre os vários pontos de vista;
- É frequentemente a pessoa sobre quem o sucesso da revisão depende.

### Líder de revisão

- Assumir a responsabilidade geral pela revisão;
- Decidir quem será envolvido e organizar quando e onde acontecerá a revisão.

### Revisor

- Podem ser especialistas no tema, pessoas trabalhando no projeto, *stakeholders* do produto de trabalho ou indivíduos com formação técnica ou profissional específica;
- Identificar possíveis defeitos no produto de trabalho sob revisão;
- Pode representar diferentes perspectivas (p. ex., testador, programador, usuário, operador, analista de negócios, especialista em usabilidade etc.).

### Redator (ou registrador)

- Coletar possíveis defeitos encontrados durante a atividade de revisão individual;
- Registrar novos defeitos em potencial, pontos em aberto e decisões da reunião de revisão (quando realizada).

Em alguns tipos de revisão, uma pessoa pode desempenhar mais de uma função e as ações associadas a cada função também podem variar com base no tipo de revisão. Além disso, com o advento de ferramentas para apoiar o processo de revisão, especialmente o registro de defeitos, os pontos em aberto e decisões, muitas vezes não há necessidade de um redator.

Além disso, funções mais detalhadas são possíveis, conforme descrito no padrão [ISO20246].

## 3.2.3 Tipos de revisão

Embora as revisões possam ser usadas para vários propósitos, um dos principais objetivos é descobrir defeitos. Todos os tipos de revisão podem auxiliar na detecção de defeitos, e o tipo de revisão selecionado deve ser baseado nas necessidades do projeto, recursos disponíveis, tipo de produto e riscos, domínio do negócio e cultura da empresa, entre outros critérios de seleção.

Um único produto de trabalho pode ser objeto de mais de um tipo de revisão. Se mais de um tipo de revisão for usado, o pedido pode variar. Por exemplo, uma revisão informal pode ser realizada antes

de uma revisão técnica, para garantir que o produto de trabalho esteja pronto para uma revisão técnica.

Os tipos de revisão descritos acima podem ser feitos como revisões por pares, ou seja, feitas por colegas em um nível organizacional aproximado semelhante.

Os tipos de defeitos encontrados em uma revisão variam, dependendo especialmente do produto de trabalho que está sendo revisado. Ver capítulo 3.1.3 para exemplos de defeitos que podem ser encontrados por revisões em diferentes produtos de trabalho e consulte Gilb (1993) para obter informações sobre inspeções formais. As revisões podem ser classificadas de acordo com vários atributos. A seguir, lista dos quatro tipos mais comuns de revisões e seus atributos associados.

### Revisão informal (p. ex., verificação de parceiros, pareamento, revisão de pares)

- O objetivo principal é detectar defeitos potenciais;
- Outros objetivos seriam gerar novas ideias ou soluções, resolvendo rapidamente pequenos problemas;
- Não baseado em um processo formal (documentado);
- Não pode envolver uma reunião de revisão;
- Pode ser realizado por um colega do autor (verificação de parceiro) ou por mais pessoas;
- Os resultados podem ser documentados;
- Varia em utilidade dependendo dos revisores;
- O uso de checklists é opcional;
- Muito comumente usado no desenvolvimento ágil.

### Acompanhamento (walkthough)

- Os principais objetivos são encontrar defeitos, melhorar o produto de software, considerar implementações alternativas, avaliar a conformidade com padrões e especificações;
- Objetivos adicionais seriam a troca de ideias sobre técnicas ou variações de estilo, treinamento de participantes, obtenção de consenso;
- A preparação individual antes da reunião de revisão é opcional;
- A reunião de revisão é normalmente liderada pelo autor do produto de trabalho;
- O redator é obrigatório;
- O uso de checklists é opcional;
- Pode assumir a forma de cenários, teste prático (*dry run*) ou simulações;
- Possíveis logs de defeitos e relatórios de revisão podem ser produzidos;
- Pode variar na prática de bastante informal para muito formal.

### Revisão técnica

- Os principais objetivos são a obtenção de consenso e a detecção de defeitos potenciais.
- Outros objetivos possíveis são: avaliar a qualidade e criar confiança no produto de trabalho, gerando novas ideias, motivando e capacitando os autores a melhorar os produtos de trabalho futuros, considerando implementações alternativas.

- Os revisores devem ser pares técnicos do autor e especialistas técnicos nas mesmas disciplinas ou em outras disciplinas.
- É necessária a preparação individual antes da reunião.
- A reunião de revisão é opcional, preferencialmente conduzida por um facilitador treinado (normalmente que não seja o autor).
- O redator é obrigatório e preferencialmente que não seja o autor.
- O uso de checklists é opcional.
- São produzidos registros de defeitos potenciais e o relatório de revisão.

#### Inspeção

- Os principais objetivos são detectar os defeitos potenciais e avaliar a qualidade e criar confiança no produto de trabalho, evitando futuros defeitos semelhantes por meio do aprendizado do autor e da análise da causa-raiz.
- Outros possíveis objetivos são: motivar e capacitar os autores a melhorar os futuros produtos de trabalho e o processo de desenvolvimento de software, alcançando consenso.
- Segue um processo definido com saídas documentadas formais, com base em regras e checklists.
- Utiliza funções claramente definidas, como aquelas especificadas no capítulo 3.2.2, que são obrigatórias, e podem incluir um leitor dedicado (que lê o produto de trabalho em voz alta durante a reunião de revisão).
- É necessária a preparação individual antes da reunião.
- Os revisores são pares do autor ou especialistas em outras disciplinas relevantes para o produto de trabalho.
- São usados critérios de entrada e saída especificados.
- O redator é obrigatório.
- A reunião de revisão é liderada por um facilitador treinado (não pelo autor).
- O autor não pode atuar como líder de revisão, leitor ou redator.
- São produzidos registros de defeitos potenciais e o relatório de revisão.
- As métricas são coletadas e usadas para melhorar todo o processo de desenvolvimento de software, incluindo o processo de inspeção.

### 3.2.4 Aplicando técnicas de revisão

Há uma série de técnicas de revisão que podem ser aplicadas durante a atividade de revisão individual (ou seja, preparação individual) para descobrir defeitos. Essas técnicas podem ser usadas nos tipos de revisão descritos acima. A eficácia das técnicas pode variar dependendo do tipo da revisão utilizada. Abaixo são listados exemplos de diferentes técnicas de revisão individuais para vários tipos de revisão.

#### Revisão ad hoc

Em uma revisão ad hoc, os revisores recebem pouca ou nenhuma orientação sobre como essa tarefa deve ser executada. Os revisores geralmente leem o produto de trabalho sequencialmente,

identificando e documentando os problemas à medida que os encontram. A revisão ad hoc é uma técnica comumente usada que requer pouca preparação. Essa técnica é altamente dependente das habilidades do revisor e pode levar a muitos problemas relatados em duplicidade por revisores diferentes.

### Revisão baseada em checklist

Uma revisão baseada em checklist é uma técnica sistemática, na qual os revisores detectam os problemas com base em checklists que são distribuídos no início da revisão (p. ex., pelo facilitador). Um checklist de revisão consiste em um conjunto de perguntas baseadas em possíveis defeitos, que podem ser derivados da experiência. Os checklists devem ser específicos para o tipo de produto de trabalho sob teste e devem ser mantidos regularmente para cobrir os tipos de problemas perdidos nas revisões anteriores. A principal vantagem da técnica baseada em checklist é uma cobertura sistemática dos típicos tipos de defeitos. Deve-se tomar cuidado para não seguir simplesmente o checklist na revisão individual, mas também para procurar defeitos fora do checklist.

### Revisão baseada em cenários

Em uma revisão baseada em cenário, os revisores recebem orientações estruturadas sobre como ler o produto de trabalho. Uma abordagem baseada no cenário dá suporte aos revisores na execução de "sequências" no produto de trabalho com base no uso esperado do produto de trabalho (se o produto de trabalho for documentado em um formato adequado, como casos de uso). Esses cenários fornecem aos revisores melhores diretrizes sobre como identificar tipos específicos de defeitos do que simples entradas no checklist. Assim como nas revisões baseadas em checklist, para não perder outros tipos de defeitos (p. ex., ausência de recursos), os revisores não devem ser restritos aos cenários documentados.

### Revisão baseada na perspectiva

Na revisão baseada na perspectiva, semelhante a uma revisão baseada em papéis, os revisores assumem os pontos de vista de diferentes *stakeholders* na revisão individual. Os pontos de vista comuns dos *stakeholders* incluem o usuário final, marketing, designer, testador ou operador. Usar esses diferentes pontos de vista leva a uma maior profundidade na revisão individual, com menos duplicidade de problemas entre os revisores.

Além disso, esse modelo também exige que os revisores tentem usar o produto de trabalho sob revisão para gerar o produto que eles obteriam. Por exemplo, um testador tentaria gerar testes de aceite de rascunho ao executar uma revisão baseada na perspectiva em uma especificação de requisitos para ver se todas as informações necessárias foram incluídas. Além disso, nesse modelo, espera-se que os checklists sejam usados.

Estudos empíricos mostraram que essa revisão é a técnica mais eficaz para revisar os requisitos e os produtos de trabalho técnico. Um fator-chave para o sucesso é incluir e ponderar adequadamente os diferentes pontos de vista dos *stakeholders*, baseando-se nos riscos. Veja Shul (2000) para detalhes sobre leitura baseada em perspectiva e Sauer (2000) para a eficácia de diferentes tipos de revisão.

### Revisão baseada em papéis

Uma revisão baseada em papéis é uma técnica na qual os revisores avaliam o produto de trabalho na perspectiva dos papéis individuais dos *stakeholders*. Os papéis típicos incluem tipos específicos de usuário final (experiente, inexperiente, sênior, jovem etc.) e papéis específicos na organização (administrador do usuário, administrador do sistema, testador de performance etc.).

## 3.2.5 Fatores de sucesso para revisões

Para ter uma revisão bem-sucedida, devem ser considerados o tipo e as técnicas apropriadas de revisão. Além disso, há vários outros fatores que afetarão o resultado.

Fatores organizacionais de sucesso para revisões incluem:

- Cada revisão tem objetivos claros, definidos durante o seu planejamento e usados como critérios mensuráveis de saída.
- São aplicados tipos de revisão que são adequados para alcançar os objetivos e são apropriados ao tipo e nível de produtos de trabalho de software e participantes.
- Quaisquer técnicas usadas de revisão, como baseada em checklist ou baseada em papéis, são adequadas para a identificação efetiva de defeitos no produto de trabalho a ser revisado.
- Todos os checklists utilizados abordam os principais riscos e deverão estar atualizados.
- Documentos grandes são escritos e revisados em pequenos pedaços, para que o controle de qualidade seja exercido através do fornecimento de feedback antecipado e frequente aos autores sobre os defeitos.
- Os participantes devem ter tempo suficiente para se preparar.
- As revisões são agendadas com aviso adequado.
- A gerência apóia o processo de revisão (p. ex., incorporando tempo adequado para atividades de revisão nos cronogramas do projeto).

Os fatores de sucesso relacionados às pessoas para revisões incluem:

- As pessoas certas estão envolvidas para atender aos objetivos de revisão, por exemplo, pessoas com diferentes conjuntos de habilidades ou perspectivas, que podem usar o documento como uma entrada de trabalho.
- Os testadores são vistos como revisores valiosos que contribuem para a revisão e aprendem sobre o produto de trabalho, o que permite que eles preparem testes mais eficazes e preparem esses testes antes.
- Os participantes dedicam tempo e atenção adequados aos detalhes.
- As revisões são realizadas em pequenos trechos, para que os revisores não percam a concentração durante a revisão individual ou na reunião de revisão (quando realizada).
- Os defeitos encontrados são reconhecidos, apreciados e manipulados objetivamente.
- A reunião é bem administrada, para que os participantes considerem um uso valioso do seu tempo.
- A revisão é conduzida em uma atmosfera de confiança; o resultado não será usado para a avaliação dos participantes.

- Os participantes evitam linguagem corporal e comportamentos que possam indicar tédio, exasperação ou hostilidade a outros participantes.
- O treinamento adequado é fornecido, especialmente para os tipos de revisão mais formais, como as inspeções.
- Uma cultura de aprendizado e melhoria de processos é promovida.

Veja Gilb (1993), Wiegers (2002), e van Veenendaal (2004) para mais sobre revisões bem-sucedidas.

# 4 Técnicas de teste [330 min]

## *Conceitos-chave*

técnica de teste caixa-preta, análise de valor limite, teste baseado em checklist, cobertura, cobertura de decisão, teste de tabela de decisão, suposição de erro, particionamento de equivalência, técnica de teste baseada na experiência, teste exploratório, teste de transição de estado, técnica de teste de caso de teste, técnica de teste caixa-branca

## *Objetivos de aprendizagem*

### *4.1 Categorias de técnicas de teste*

FL-4.1.1 (K2) Explicar as características, semelhanças e diferenças entre as técnicas de teste caixa-preta, técnicas de teste caixa-branca e técnicas de teste baseadas na experiência.

### *4.2 Técnicas de teste caixa-preta*

FL-4.2.1 (K3) Aplicar o particionamento de equivalência para derivar casos de teste de requisitos específicos.

FL-4.2.2 (K3) Aplicar a análise do valor limite para derivar os casos de teste de requisitos específicos.

FL-4.2.3 (K3) Aplicar teste de tabela de decisão para derivar casos de teste de requisitos específicos.

FL-4.2.4 (K3) Aplicar teste de transição de estado para derivar casos de teste de requisitos específicos.

FL-4.2.5 (K2) Explicar como derivar casos de teste de um caso de uso.

### *4.3 Técnicas de teste caixa-branca*

FL-4.3.1 (K2) Explicar a cobertura de instruções.

FL-4.3.2 (K2) Explicar a cobertura de decisão.

FL-4.3.3 (K2) Explicar o valor da instrução e a cobertura da decisão.

### *4.4 Técnicas de teste baseadas na experiência*

FL-4.4.1 (K2) Explicar a suposição de erro.

FL-4.4.2 (K2) Explicar o teste exploratório.

FL-4.4.3 (K2) Explicar os testes baseados em checklists.

## 4.1 Categorias de técnicas de teste

O objetivo de uma técnica de teste, incluindo as discutidas nesse capítulo, é ajudar a identificar as condições de teste, os casos de teste e os dados de teste.

A escolha de quais técnicas de teste usar depende de vários fatores, incluindo:

- Complexidade do componente ou do sistema;
- Normas regulatórias;
- Requisitos contratuais ou do cliente;
- Níveis e tipos de risco;
- Documentação disponível;
- Conhecimento e habilidades do testador;
- Ferramentas disponíveis;
- Tempo e orçamento;
- Modelo de ciclo de vida de desenvolvimento de software.
- Os tipos de defeitos esperados no componente ou sistema

Algumas técnicas são mais aplicáveis em determinadas situações e níveis de teste, outras são aplicáveis em todos os níveis de teste. Ao criar casos de teste, os testadores geralmente usam uma combinação de técnicas de teste para obter os melhores resultados do esforço de teste.

O uso de técnicas de teste nas atividades de análise do teste, projeto de teste e implementação do teste pode variar de muito informal (pouca ou nenhuma documentação) a muito formal. O nível adequado da formalidade depende do contexto dos testes, incluindo a maturidade dos processos de teste e de desenvolvimento, das restrições de tempo, dos requisitos normativos ou de segurança, do conhecimento e das habilidades das pessoas envolvidas e do modelo de ciclo de vida de desenvolvimento de software que está sendo seguido.

### 4.1.1 Categorias de técnicas de teste e suas características

Neste syllabus, as técnicas de teste são classificadas como caixa-preta, caixa-branca ou baseada na experiência.

As técnicas de teste caixa-preta (também chamadas de técnicas comportamentais ou baseadas no comportamento) são fundamentadas em uma análise da base de teste apropriada (p. ex., documentos de requisitos formais, especificações, casos de uso, histórias de usuários ou processos de negócios). Essas técnicas são aplicáveis a testes funcionais e não funcionais. As técnicas de teste caixa-preta se concentram nas entradas e saídas do objeto de teste sem referência a sua estrutura interna.

As técnicas de teste caixa-branca (também chamadas de técnicas estruturais ou baseadas na estrutura) são baseadas em uma análise da arquitetura, do detalhamento do projeto, da estrutura interna ou o código do objeto de teste. Ao contrário das técnicas de teste caixa-preta, as técnicas de teste caixa-branca concentram-se na estrutura e no processamento dentro do objeto de teste.

As técnicas de teste baseadas na experiência aproveitam a conhecimento dos desenvolvedores, testadores e usuários para projetar, implementar e executar os testes. Estas técnicas são frequentemente combinadas com técnicas de teste caixa-preta e caixa-branca.

As características comuns das técnicas de teste caixa-preta incluem o seguinte:

- As condições de teste, os casos de teste e os dados de teste são derivados de uma base de teste que pode incluir requisitos de software, especificações, casos de uso e histórias de usuários;
- Os casos de teste podem ser usados para detectar lacunas entre os requisitos e as suas implementações, bem como desvios nos requisitos;
- A cobertura é medida com base nos itens testados na base de teste e na técnica aplicada nesta base.

As características comuns das técnicas de teste caixa-branca incluem o seguinte:

- As condições de teste, os casos de teste e os dados de teste são derivados de uma base de teste que pode incluir código, arquitetura de software, o detalhamento do projeto ou qualquer outra fonte de informação relacionada à estrutura do software;
- A cobertura é medida com base nos itens testados em uma estrutura selecionada (p.ex., o código ou interfaces) e a técnica aplicada à base de teste.

As características comuns das técnicas de teste baseadas na experiência incluem o seguinte:

- As condições de teste, os casos de teste e os dados de teste são derivados de uma base de teste que pode incluir conhecimento e a experiência de testadores, desenvolvedores, usuários e *stakeholders*.

Esse conhecimento e experiência inclui o uso esperado do software, seu ambiente, possíveis defeitos e a distribuição desses defeitos.

O padrão internacional (ISO / IEC / IEEE 29119-4) contém descrições de técnicas de teste e suas medidas de cobertura correspondentes (veja Craig 2002 e Copeland 2004 para mais informações sobre técnicas).

## 4.2 Técnicas de teste caixa-preta

### 4.2.1 Particionamento de equivalência

O particionamento de equivalência divide os dados em partições (também conhecidas como classes de equivalência), de tal forma que todos os membros de uma determinada partição deve ser processado da mesma maneira (ver Kaner (2013) e Jorgensen (2014)). Existem partições de equivalência para valores válidos e inválidos.

- Valores válidos são valores que devem ser aceitos pelo componente ou sistema. Uma partição de equivalência contendo este tipo de valores é chamada de "partição de equivalência válida".

- Valores inválidos são valores que devem ser rejeitados pelo componente ou sistema. Uma partição de equivalência contendo estes valores é chamada de "partição de equivalência inválida".
- As partições podem ser identificadas para qualquer elemento de dados relacionado ao objeto de teste, incluindo entradas, saídas, valores internos, valores relacionados a tempo (p. ex., antes ou depois de um evento) e para parâmetros de interface (p. ex., componentes integrados sendo testados durante o teste de integração).
- Se necessário, qualquer partição pode ser dividida em subpartições.
- Cada valor deve pertencer a uma e apenas uma partição de equivalência.
- Quando partições de equivalência inválidas são usadas em casos de teste, elas devem ser testadas individualmente, ou seja, não podem ser combinadas com outras partições de equivalência inválidas, para garantir que as falhas não sejam mascaradas. Falhas podem ser mascaradas quando várias falhas ocorrem ao mesmo tempo, mas apenas uma é visível, fazendo com que as outras falhas não sejam detectadas.

Para obter uma cobertura de 100% com essa técnica, os casos de teste devem cobrir todas as partições identificadas (incluindo partições inválidas) usando no mínimo um valor de cada partição. A cobertura é medida como o número de partições de equivalência testadas por pelo menos um valor, dividido pelo número total de partições de equivalência identificadas, normalmente expresso como uma porcentagem. O particionamento de equivalência é aplicável em todos os níveis de teste.

### 4.2.2 Análise de valor limite

A análise de valor limite é uma extensão do particionamento de equivalência, mas só pode ser usada quando a partição é ordenada, consistindo em dados numéricos ou sequenciais. Os valores mínimo e máximo (ou primeiro e último valores) de uma partição são seus valores limites (Beizer, 1990).

Por exemplo, suponha que um campo de entrada aceite um único valor inteiro como uma entrada, usando um teclado para limitar que entradas não inteiras sejam impossíveis. O intervalo válido é de 1 a 5, inclusive. Portanto, existem três partições de equivalência: inválidas (muito baixas); válido; inválido (muito alto). Para a partição de equivalência válida, os valores limites são 1 e 5. Para a partição inválida (muito alta), os valores limites são 6 e 9. Para a partição inválida (muito baixa), existe apenas um valor limite, 0, porque esta é uma partição com apenas um membro.

No exemplo acima, identificamos dois valores limites por limite. O limite entre inválido (muito baixo) e válido fornece os valores de teste 0 e 1. A fronteira entre válido e inválido (muito alto) fornece os valores de teste 5 e 6. Algumas variações dessa técnica podem identificar três valores de limite por limite de dados: o valor anterior ao limite, o próprio valor limite e o imediatamente superior a ele. No exemplo anterior, usando valores limites de três pontos, os valores do limite inferior são 0, 1 e 2, e os valores dos limites superiores são 4, 5 e 6 (Jorgensen (2014)).

O comportamento nos limites das partições de equivalência é mais provável que seja incorreto do que o comportamento dentro das partições. É importante lembrar que os limites especificados e implementados podem ser deslocados para posições acima ou abaixo de suas posições pretendidas,

podem ser omitidos por completo ou podem ser complementados com limites adicionais indesejados. A análise e o teste do valor-limite revelarão quase todos esses defeitos forçando o software a mostrar comportamentos de uma partição diferente daquela à qual o valor limite deveria pertencer.

A análise de valor limite pode ser aplicada em todos os níveis de teste. Essa técnica é geralmente usada para testar requisitos que exigem um intervalo de números (incluindo datas e horas). A cobertura de limite para uma partição é medida como o número de valores limites testados, dividido pelo número total de valores de teste de limite identificados, normalmente expressos como uma porcentagem.

### 4.2.3 Teste de tabela de decisão

Técnicas de teste combinatórias são úteis para testar a implementação de requisitos do sistema que especificam como diferentes condições de combinações levam à resultados diferentes. Uma abordagem para esse teste é o teste de tabela de decisão.

As tabelas de decisão são uma boa maneira de registrar regras de negócios complexas que um sistema deve implementar. Ao criar tabelas de decisão, o testador identifica as condições (geralmente entradas) e as ações resultantes (geralmente saídas) do sistema. Estes formam as linhas da tabela, geralmente com as condições no topo e as ações na parte inferior. Cada coluna corresponde a uma regra de decisão que define uma combinação exclusiva de condições que resultam na execução das ações associadas a essa regra. Os valores das condições e ações são geralmente mostrados como valores booleanos (verdadeiro ou falso) ou valores discretos (p. ex., vermelho, verde, azul), mas também podem ser números ou intervalos numéricos. Esses diferentes tipos de condições e ações podem ser encontrados juntos na mesma tabela.

A notação comum nas tabelas de decisão é a seguinte:

Para condições:

- “S”: significa que a condição é verdadeira (também pode ser mostrada como “V” ou “1”)
- “N”: significa que a condição é falsa (também pode ser mostrada como “F” ou “0”)
- “-“: significa que o valor da condição não importa (também pode ser mostrado como “N/A”)

Para ações:

- “X” significa que a ação deve ocorrer (também pode ser mostrada como “S” ou “V” ou “1”)
- Em branco significa que a ação não deve ocorrer (também pode ser mostrada como “-“ ou “N” ou “F” ou “0”)

Uma tabela de decisão completa tem colunas suficientes para abranger todas as combinações de condições. A tabela pode ser reduzida excluindo-se as colunas contendo combinações impossíveis, colunas contendo possíveis combinações de condições, mas inviáveis, e colunas que testam as combinações de condições que não afetam o resultado. Para obter mais informações sobre como recolher as tabelas de decisão, consulte o Syllabus [ISTQB_AL_TA].

O padrão comumente usado de cobertura mínima para o teste de tabela de decisão é ter pelo menos um caso de teste por regra de decisão na tabela. Isso normalmente envolve cobrir todas as combinações de condições. A cobertura é medida como o número de regras de decisão testadas por pelo menos um caso de teste, dividido pelo número total de regras de decisão, normalmente expressa como uma porcentagem.

O poder do teste por tabela de decisão é que ela ajuda a identificar todas as combinações importantes de condições, algumas das quais poderiam ser negligenciadas. Também ajuda a encontrar quaisquer lacunas nos requisitos. Pode ser aplicada a todas as situações em que o comportamento do software depende de uma combinação de condições, em qualquer nível de teste.

### 4.2.4 Teste de transição de estado

Componentes ou sistemas podem responder de maneira diferente a um evento, dependendo das condições atuais ou do histórico anterior (p. ex., os eventos que ocorreram desde que o sistema foi inicializado). O histórico anterior pode ser resumido usando o conceito de estados. Um diagrama de transição de estado mostra os possíveis estados do software, bem como a forma como o software entra, sai e transita entre os estados. Uma transição é iniciada por um evento (p. ex., entrada do usuário de um valor em um campo). O evento resulta em uma transição. Se o mesmo evento pode resultar em duas ou mais transições diferentes do mesmo estado, esse evento pode ser qualificado por uma condição de proteção. A mudança de estado pode fazer com que o software execute uma ação (p. ex., gerar uma mensagem de cálculo ou de erro).

Uma tabela de transição de estado mostra todas as transições válidas e potencialmente inválidas entre estados, bem como os eventos, condições de proteção e ações resultantes para transições válidas. Os diagramas de transição de estado normalmente mostram apenas as transições válidas e excluem as transições inválidas.

Os testes podem ser projetados para cobrir uma sequência típica de estados, para exercitar todos os estados, para exercitar cada transição, para executar sequências específicas de transições ou para testar transições inválidas.

O teste de transição de estado é usado em aplicativos baseados em menus e é amplamente usado na indústria de software de prateleira. A técnica também é adequada para modelar um cenário de negócios com estados específicos ou para testar a navegação na tela. O conceito de um estado é abstrato - pode representar algumas linhas de código ou um processo de negócios inteiro.

A cobertura é geralmente medida como o número de estados identificados ou transições testadas, dividido pelo número total de estados ou transições identificadas no objeto de teste, normalmente expresso como uma porcentagem. Para obter mais informações sobre os critérios de cobertura para testes de transição de estado, consulte o Syllabus [ISTQB_AL_TA].

### 4.2.5 Teste de caso de uso

Os testes podem ser derivados de casos de uso, que são uma maneira específica de projetar interações com itens de software, incorporando requisitos para as funções de software representadas

pelos casos de uso. Os casos de uso estão associados a atores (usuários humanos, hardware externo ou outros componentes ou sistemas) e assuntos (o componente ou sistema ao qual o caso de uso é aplicado).

Cada caso de uso especifica algum comportamento que um assunto pode realizar em colaboração com um ou mais atores (UML 2.5.1 2017). Um caso de uso pode ser descrito por interações e atividades, bem como condições prévias, pós-condições e linguagem natural, quando apropriado. Interações entre os atores e o sujeito podem resultar em mudanças no estado do sujeito. As interações podem ser representadas graficamente por fluxos de trabalho, diagramas de atividades ou modelos de processos de negócios.

Um caso de uso pode incluir possíveis variações de seu comportamento básico, incluindo comportamento excepcional e tratamento de erros (resposta do sistema e recuperação de erros de programação, aplicação e comunicação, p.e., resultando em uma mensagem de erro). Os testes são projetados para exercitar os comportamentos definidos (básico, excepcional ou alternativo e tratamento de erros). A cobertura pode ser medida pela porcentagem de comportamentos de casos de uso testados dividida pelo número total de comportamentos de casos de uso, normalmente expressos como uma porcentagem.

Para obter mais informações sobre critérios de cobertura para o teste de casos de uso, consulte Syllabus [ISTQB_AL_TA].

## 4.3 Técnicas de teste caixa-branca

O teste caixa-branca é baseado na estrutura interna do objeto de teste. As técnicas de teste caixa-branca podem ser usadas em todos os níveis de teste, mas as duas técnicas relacionadas a códigos discutidas neste capítulo são as mais comumente usadas no nível de teste de componente. Existem técnicas mais avançadas que são usadas em alguns ambientes de segurança crítica, de missão crítica ou de alta integridade para obter uma cobertura mais completa, mas essas não são discutidas aqui. Para obter mais informações sobre essas técnicas, consulte o Syllabus [ISTQB_AL_TTA].

### 4.3.1 Teste e cobertura de instruções

Esta técnica testa as instruções executáveis do código. A cobertura é medida como o número de instruções executadas pelos testes dividido pelo número total de instruções executáveis existentes, normalmente expresso como uma porcentagem.

### 4.3.2 Teste de decisão e cobertura

Esta técnica testa as decisões existentes no código e o código executado com base nos resultados da decisão. Para fazer isso, os casos de teste seguem os fluxos de controle que ocorrem de um ponto de decisão (p. ex., para uma instrução IF, um para o resultado verdadeiro e outro para o resultado falso; para uma instrução CASE, os casos de teste seriam necessários para todos os possíveis resultados, incluindo o resultado padrão).

A cobertura é medida como o número de resultados de decisão executados pelos testes dividido pelo número total de resultados de decisão no objeto de teste, normalmente expresso como uma porcentagem.

### 4.3.3 O valor da instrução e teste de decisão

Quando a cobertura de 100% das instruções é alcançada, garante-se que todas as instruções executáveis no código tenham sido testadas pelo menos uma vez, mas não garante que toda a lógica de decisão tenha sido testada. Das duas técnicas caixa-branca discutidas neste syllabus, o teste de instrução pode fornecer menos cobertura do que o teste de decisão.

Quando uma cobertura de decisão de 100% é alcançada, representa-se que todos os resultados de decisão foram testados, o que inclui testar os resultados verdadeiro e falso, mesmo quando não há uma instrução falsa explícita (p. ex., no caso de uma instrução IF sem outra no código). A cobertura de instrução ajuda a encontrar defeitos no código que não foi exercido por outros testes. A cobertura de decisão ajuda a encontrar defeitos no código, em que outros testes não obtiveram resultados verdadeiros ou falsos.

Atingir 100% de cobertura de decisão garante 100% de cobertura de instrução (mas não vice-versa).

## 4.4 Técnicas de teste baseadas na experiência

Ao aplicar as técnicas de teste baseadas na experiência, os casos de teste são derivados da habilidade e intuição do testador e de sua experiência com aplicativos e tecnologias semelhantes. Essas técnicas podem ser úteis na identificação de testes que não foram facilmente identificados por outras técnicas mais sistemáticas. Dependendo da abordagem e experiência do testador, essas técnicas podem alcançar graus de cobertura e eficácia amplamente variados. A cobertura pode ser difícil de avaliar e pode não ser mensurável com essas técnicas.

Técnicas baseadas em experiência comumente usadas são discutidas nas seções seguintes.

### 4.4.1 Suposição de erro

A suposição de erro é uma técnica usada para prever a ocorrência de erros, defeitos e falhas, com base no conhecimento do testador, incluindo:

- Como o aplicativo funcionou no passado;
- Que tipos de erro tendem a ser cometidos;
- Falhas ocorridas em outros aplicativos.

Uma abordagem metódica para a técnica de suposição de erro é criar uma lista de possíveis erros, defeitos e falhas e modelar os testes que exporão essas falhas e os defeitos que as causaram. Esses erros, defeitos, listas de falhas podem ser construídos com base na experiência, nos dados de defeitos e falhas ou no conhecimento comum sobre o motivo da falha do software.

### 4.4.2 Teste exploratório

Nesta técnica, os testes informais (não pré-definidos) são modelados, executados, registrados e avaliados dinamicamente durante a execução do teste. Os resultados do teste são usados para aprender mais sobre o componente ou sistema e para criar testes para as áreas que podem precisar de mais testes.

Às vezes, o teste exploratório é realizado usando testes baseados em sessão para estruturar as atividades de teste. No teste baseado em sessão, o teste exploratório é conduzido dentro de uma janela de tempo definida, e o testador usa um termo de teste contendo objetivos de teste para orientar o teste. O testador pode usar folhas de sessão de teste para documentar as etapas seguidas e as descobertas feitas.

O teste exploratório é mais útil quando há poucas ou inadequadas especificações ou pressão de tempo significativa nos testes. O teste exploratório também é útil para complementar outras técnicas de teste mais formais.

O teste exploratório está fortemente associado a estratégias de teste reativo (ver capítulo 5.2.2). O teste exploratório pode incorporar o uso de outras técnicas de caixa-preta, caixa-branca e experiência.

### 4.4.3 Teste baseado em checklist

Em testes baseados em checklist, os testadores modelam, implementam e executam testes para cobrir as condições de teste encontradas em uma lista. Como parte da análise, os testadores criam uma lista ou expandem uma existente, mas os testadores também podem usar um checklist existente sem modificá-lo. Esses checklists podem ser construídos com base na experiência, conhecimento sobre o que é importante para o usuário ou uma compreensão de por que e como o software falha.

Os checklists podem ser criados para dar suporte a vários tipos de teste, incluindo testes funcionais e não funcionais. Na ausência de casos de teste detalhados, os testes baseados em checklist podem fornecer diretrizes e consistência. Como essas listas são de alto nível, é provável que ocorra alguma variabilidade nos testes reais, resultando em uma cobertura potencialmente maior, mas com menos repetibilidade.

# 5 Gerenciamento de teste [225 min]

### *Conceitos-chave*

gerenciamento de configuração, gerenciamento de defeitos, relatório de defeitos, critério de entrada, critério de saída, risco de produto, risco de projeto, risco, nível de risco, teste baseado em risco, abordagem de teste, controle de teste, estimativa de teste, gerente de teste, monitoramento de teste, plano de teste, planejamento de teste, relatório de progresso, estratégia de teste, relatório de resumo de teste, testador

### *Objetivos de Aprendizagem*

#### *5.1 Organização de teste*

FL-5.1.1 (K2) Explicar os benefícios e as desvantagens de testes independentes.

FL-5.1.2 (K1) Identificar as tarefas do gerente de teste e do testador.

#### *5.2 Planejamento e estimativa de testes*

FL-5.2.1 (K2) Resumir o objetivo e o conteúdo de um plano de teste.

FL-5.2.2 (K2) Diferenciar entre várias estratégias de teste.

FL-5.2.3 (K2) Dar exemplos de possíveis critérios de entrada e saída.

FL-5.2.4 (K3) Aplicar o conhecimento de priorização e dependências técnicas e lógicas para agendar a execução do teste para um determinado conjunto de casos de teste.

FL-5.2.5 (K1) Identificar fatores que influenciam o esforço relacionado ao teste.

FL-5.2.6 (K2) Explicar a diferença entre as técnicas de estimativa baseada em métricas e em especialistas.

#### *5.3 Monitoramento e controle de testes*

FL-5.3.1 (K1) Recordar as métricas usadas nos testes

FL-5.3.2 (K2) Resumir os propósitos, o conteúdo e o público dos relatórios de teste

#### *5.4 Gerenciamento de configurações*

FL-5.4.1 (K2) Resumir como o gerenciamento de configuração apoia os testes

#### *5.5 Riscos e testes*

FL-5.5.1 (K1) Definir o nível de risco usando, a probabilidade e o impacto.

FL-5.5.2 (K2) Distinguir entre riscos de projeto e do produto.

FL-5.5.3 (K2) Exemplificar como a análise de risco pode influenciar a eficácia e o alcance do teste.

#### *5.6 Gerenciamento de defeitos*

FL-5.6.1 (K3) Escrever um relatório de defeito, cobrindo os defeitos encontrados durante o teste

## 5.1 Organização do teste

### 5.1.1 Teste independente

As tarefas de teste podem ser executadas por pessoas em um papel de teste específico ou por pessoas em outros papéis (p. ex., clientes). Um certo grau de independência geralmente torna o testador mais eficaz em encontrar os defeitos devido às diferenças entre os vieses cognitivos do autor e do testador (ver capítulo 1.5). A independência não é, no entanto, um substituto para a familiaridade com o produto, e os desenvolvedores podem encontrar com eficiência muitos defeitos em seu próprio código.

Os graus de independência nos testes incluem (de baixo nível de independência a alto nível):

- Sem testadores independentes, a única forma de teste disponível são os desenvolvedores testando seu próprio código.
- Desenvolvedores independentes ou testadores dentro das equipes de desenvolvimento ou da equipe do projeto; isso poderiam ser desenvolvedores testando os produtos de seus colegas.
- Equipe de teste independente ou grupo dentro da organização, reportando-se ao gerenciamento de projetos ou ao gerenciamento executivo.
- Testadores independentes da organização empresarial ou da comunidade de usuários, ou com especializações em tipos de testes específicos, como usabilidade, segurança, performance, regulamentação, conformidade ou portabilidade.
- Testadores independentes externos à organização, trabalhando dentro ou fora do escritório.

Para a maioria dos tipos de projetos, geralmente é melhor ter vários níveis de teste, com alguns desses níveis gerenciados por testadores independentes. Os desenvolvedores devem participar dos testes, especialmente nos níveis mais baixos, para exercer controle sobre a qualidade de seu próprio trabalho.

A maneira pela qual a independência dos testes é implementada varia dependendo do modelo de ciclo de vida de desenvolvimento de software. Por exemplo, no desenvolvimento ágil, os testadores podem fazer parte de uma equipe de desenvolvimento. Em algumas organizações que usam métodos ágeis, esses testadores também podem ser considerados parte de uma equipe maior de testes independentes. Além disso, em tais organizações, os proprietários de produtos podem realizar testes de aceite para validar as histórias de usuários ao final de cada iteração.

Os benefícios potenciais da independência do teste incluem:

- Os testadores independentes provavelmente reconhecerão diferentes tipos de falhas em comparação a os desenvolvedores, devido a diferentes históricos, perspectivas técnicas e vieses.
- Um testador independente pode verificar, desafiar ou refutar as suposições feitas pelos *stakeholders* durante a especificação e a implementação do sistema.

As desvantagens potenciais da independência do teste incluem:

- Isolamento da equipe de desenvolvimento, levando a uma falta de colaboração, atrasos no fornecimento de feedback para a equipe de desenvolvimento ou um relacionamento adverso com a equipe de desenvolvimento.
- Os desenvolvedores podem perder o senso de responsabilidade pela qualidade.
- Testadores independentes podem ser vistos como gargalo ou culpados por atrasos na liberação.
- Testadores independentes podem não ter algumas informações importantes (p. ex., sobre o objeto de teste).

Muitas organizações são capazes de alcançar com sucesso os benefícios da independência do teste, evitando todas as desvantagens.

### 5.1.2 Tarefas de um gerente de teste e do testador

Neste syllabus, dois papéis no teste são cobertos, os gerentes de teste e os testadores. As atividades e tarefas desempenhadas por esses dois papéis dependem do contexto do projeto e do produto, das habilidades das pessoas nas funções e da organização.

O gerente de teste tem a responsabilidade geral do processo de teste e da liderança das atividades bem-sucedidas de teste. O papel de gerente de teste pode ser executado por um gerente de teste profissional ou por um gerente de projeto, um gerente de desenvolvimento ou um gerente de qualidade. Em projetos ou organizações maiores, várias equipes de teste podem se reportar a um gerente de teste, um técnico de testes ou um coordenador de testes, cada equipe sendo liderada por um líder de teste ou um testador líder.

As tarefas típicas do gerente de teste podem incluir:

- Desenvolver ou revisar uma política de teste e uma estratégia de teste para a organização.
- Planejar as atividades de teste considerando o contexto e entendendo os objetivos e riscos do teste. Isso pode incluir a seleção de abordagens de teste, a estimativa do tempo, o esforço e o custo, a aquisição de recursos, a definição de níveis e ciclos de teste e o planejamento da gestão dos defeitos.
- Escrever e atualizar o(s) plano(s) de teste.
- Coordenar o(s) plano(s) de teste com gerentes de projeto, proprietários dos produtos e outros.
- Compartilhar as perspectivas de teste com outras atividades do projeto, como planejamento da integração.
- Iniciar a análise, o projeto, a implementação e a execução dos testes, monitorar o progresso e os resultados obtidos e verificar o status dos critérios de saída (ou definição de feito).
- Preparar e entregar os relatórios de progresso do teste e resumo com base nas informações coletadas.
- Adaptar o planejamento com base nos resultados e no progresso dos testes (às vezes documentados em relatórios de andamento de testes ou em relatórios de resumo de teste para outros testes já concluídos no projeto) e tomar as ações necessárias para o controle de testes.

- Suporte para configurar o sistema de gerenciamento de defeitos e o gerenciamento de configuração adequado do testware.
- Introduzir as métricas adequadas para medir o progresso do teste e avaliar a qualidade do teste e do produto.
- Apoiar a seleção e implementação de ferramentas para apoiar o processo de teste, incluindo a recomendação orçamentária para seleção de ferramentas (e possivelmente compra ou suporte), alocando tempo e esforço para projetos-piloto e fornecendo suporte contínuo no uso de ferramenta.
- Decidir sobre a implementação do(s) ambiente(s) de teste.
- Promover e defender os testadores, a equipe de teste e a profissão de teste dentro da organização.
- Desenvolver as habilidades e carreiras dos testadores (p. ex., através de planos de treinamento, avaliações de performance, treinamento etc.).

A maneira como a função de gerente de teste é executada varia de acordo com o ciclo de vida de desenvolvimento de software. Por exemplo, no desenvolvimento ágil, algumas das tarefas mencionadas acima são tratadas pela equipe do Ágil, especialmente aquelas relacionadas ao teste diário feito dentro da equipe, geralmente por um testador que trabalha na equipe. Algumas das tarefas que abrangem várias equipes ou toda a organização, ou que têm a ver com gerenciamento de pessoal, podem ser executadas por gerentes de teste fora da equipe de desenvolvimento, que às vezes são chamados de treinadores de teste. Veja Black (2009) para mais informações sobre como gerenciar o processo de teste.

As tarefas típicas do testador podem incluir:

- Revisar e contribuir para os planos de teste.
- Analisar, revisar e avaliar os requisitos, as histórias de usuários e os critérios de aceite, as especificações e modelos para testabilidade (ou seja, a base de teste).
- Identificar e documentar as condições de teste e capturar a rastreabilidade entre casos de teste, as condições de teste e a base de teste.
- Projetar, configurar e verificar o(s) ambiente(s) de teste, geralmente coordenando com a administração do sistema e gerenciamento da rede.
- Projetar e implementar casos de teste e procedimentos de teste.
- Preparar e adquirir os dados de teste.
- Criar o cronograma detalhado de execução dos testes.
- Executar os testes, avaliar os resultados e documentar os desvios dos resultados esperados.
- Usar ferramentas apropriadas para facilitar o processo de teste.
- Automatizar os testes conforme necessário (pode ser suportado por um desenvolvedor ou por um especialista em automação de testes).
- Avaliar as características não funcionais, como eficiência de performance, confiabilidade, usabilidade, segurança, compatibilidade e portabilidade.
- Revisar os testes desenvolvidos por outros.

As pessoas que trabalham em análise de teste, projeto de teste, tipos de teste específicos ou automação de teste podem ser especialistas nessas funções. Dependendo dos riscos relacionados ao produto e ao projeto, e o modelo de ciclo de vida de desenvolvimento de software selecionado, pessoas diferentes podem assumir o papel de testador em diferentes níveis de teste. Por exemplo, no nível de teste de componente e no nível de teste de integração de componente, o papel de um testador é geralmente executado por desenvolvedores. No nível de teste de aceite, o papel de um testador geralmente é feito por analistas de negócios, especialistas no assunto e usuários. No nível de teste do sistema e no nível de teste de integração do sistema, o papel de um testador é geralmente executado por uma equipe de teste independente. No nível de teste de aceite operacional, o papel de um testador geralmente é executado pelas equipes de administração de operações ou sistemas.

## 5.2 Planejamento e estimativa de testes

### 5.2.1 Objetivo e conteúdo de um plano de teste

O planejamento é influenciado pela política de teste e a estratégia de teste da organização, pelos ciclos de vida e métodos de desenvolvimento usados (ver capítulo 2.1), pelo escopo dos testes, objetivos, riscos, restrições, criticidade, testabilidade e disponibilidade dos recursos.

Conforme o andamento do projeto e do planejamento do teste, mais informações ficam disponíveis e mais detalhes podem ser incluídos no plano de teste. O planejamento do teste é uma atividade contínua e é executado durante todo o ciclo de vida do produto (observe que o ciclo de vida do produto pode se estender além do escopo de um projeto para incluir a fase de manutenção). O feedback das atividades de teste deve ser usado para reconhecer a alteração de riscos para que o planejamento possa ser ajustado. O planejamento pode ser documentado em um plano de teste principal e em planos de teste separados para cada nível de teste, como testes de sistema e testes de aceite, ou para cada tipo de teste, como teste de usabilidade e teste de performance. As atividades do planejamento do teste podem ser documentadas em um plano de teste e incluir:

- Determinar o escopo, os objetivos e os riscos do teste.
- Definir a abordagem geral do teste.
- Integrar e coordenar as atividades de teste nas atividades do ciclo de vida do software.
- Tomar decisões sobre o que testar, as pessoas e outros recursos necessários para realizar as várias atividades de teste e como essas atividades serão realizadas.
- Programar as atividades de análise, projeto, implementação, execução e avaliação de testes, em datas específicas (p. ex., em desenvolvimento sequencial) ou no contexto de cada iteração (p. ex., no desenvolvimento iterativo).
- Selecionar as métricas para monitoramento e controle de testes.
- Orçar as atividades de teste.
- Determinar o nível de detalhes e a estrutura da documentação de teste (p. ex., fornecendo modelos ou exemplos de documentos).

O conteúdo dos planos de teste varia e pode se estender além dos tópicos identificados acima. Uma amostragem de planos de teste pode ser encontrada no padrão [ISO29119-3].

### 5.2.2 Estratégia de teste e abordagem de teste

Uma estratégia de teste fornece uma descrição geral do processo de teste, comumente no nível do produto ou organizacional. Tipos comuns de estratégias de teste incluem:

- **Analítica**: este tipo de estratégia de teste é baseado em uma análise de algum fator (p. ex., exigência ou risco). O teste baseado em risco é um exemplo de abordagem analítica, em que os testes são projetados e priorizados com base no nível de risco.
- **Baseada em modelo**: neste tipo de estratégia de teste, os testes são projetados com base em algum modelo de algum aspecto necessário do produto, como uma função, um processo empresarial, uma estrutura interna ou uma característica não funcional (p. ex., confiabilidade). Exemplos de tais modelos incluem os modelos de processos de negócios, os modelos de estado e os modelos de crescimento de confiabilidade.
- **Metódica**: esse tipo de estratégia de teste depende do uso sistemático de um conjunto predefinido de testes ou condições de teste, como uma taxonomia de tipos comuns ou prováveis de falhas, uma lista de características de qualidade importantes, ou padrões de aparência e comportamento de aplicativos móveis ou páginas da web da empresa.
- **Compatível com processo** (ou compatível com um padrão): esse tipo de estratégia envolve análise, projeto e implementação do teste baseado em regras e padrões externos, como aqueles determinados por padrões específicos do setor, pela documentação do processo, pela identificação e uso da base de teste, ou por qualquer processo ou padrão imposto pela organização.
- **Dirigida** (ou consultivo): esse tipo de estratégia de teste é orientado principalmente pelo aconselhamento, orientação ou instruções dos *stakeholders*, especialistas no domínio do negócio ou especialistas em tecnologia, que podem estar fora da equipe de teste ou fora da própria organização.
- **Contrarregressão**: esse tipo de estratégia de teste é motivado pelo desejo de evitar a regressão de recursos existentes. Essa estratégia de teste inclui a reutilização do *testware* existente (especialmente casos de teste e dados de teste), da automação extensiva de testes de regressão e de conjuntos padão de teste.
- **Reativa**: Nesse tipo de estratégia de teste, o teste é reativo ao componente ou sistema que está sendo testado e aos eventos que ocorrem durante a execução do teste, em vez de serem pré-planejados (como as estratégias anteriores). Os testes são planejados e implementados e podem ser imediatamente executados em resposta ao conhecimento adquirido em resultados de testes anteriores. O teste exploratório é uma técnica comum empregada em estratégias reativas.

Uma estratégia de teste apropriada é frequentemente criada pela combinação de vários desses tipos de estratégias de teste. Por exemplo, testes baseados em risco (estratégia analítica) podem ser combinados com testes exploratórios (estratégia reativa), se complementando e podendo alcançar testes mais eficazes quando usados juntos.

Embora a estratégia de teste forneça uma descrição geral do processo de teste, a abordagem de teste adapta a estratégia de teste para um projeto ou lançamento específico. A abordagem de teste é o ponto de partida para selecionar as técnicas de teste, os níveis de teste e os tipos de teste, e para definir os critérios de entrada e saída (ou da definição de "pronto" ou "feito", respectivamente). A adaptação da estratégia baseia-se em decisões tomadas em relação à complexidade e objetivos do projeto, do tipo de produto que está sendo desenvolvido e da análise de risco do produto. A abordagem selecionada depende do contexto e pode considerar fatores como riscos, segurança, recursos e habilidades disponíveis, tecnologia, natureza do sistema (p. ex., versão customizada ou software de prateleira), objetivos de teste e regulamentos.

### 5.2.3 Critérios de entrada e saída (definição de "*pronto*" e "*feito*")

Para exercer o controle efetivo sobre a qualidade do software e dos testes, é aconselhável ter critérios que definam quando uma determinada atividade de teste deve iniciar e quando a atividade é concluída. Os critérios de entrada (chamado de "*pronto*" no desenvolvimento ágil) definem as condições prévias para a realização de uma determinada atividade de teste. Se os critérios de entrada não forem cumpridos, é provável que a atividade se mostre mais difícil, mais demorada, mais cara e mais arriscada. Os critérios de saída (chamado de "*feito*" no desenvolvimento ágil) definem quais condições devem ser atingidas para declarar que um nível de teste ou um conjunto de testes foram concluídos. Os critérios de entrada e saída devem ser definidos para cada nível e tipo de teste e serão diferentes com base nos objetivos do teste.

Critérios típicos de entrada incluem:

- Disponibilidade dos requisitos testáveis, histórias de usuários e/ou modelos (p. ex., ao seguir uma estratégia de teste baseada em modelo).
- Disponibilidade dos itens de teste que atendam aos critérios de saída para quaisquer níveis de teste anteriores.
- Disponibilidade do ambiente de teste:
- Disponibilidade de ferramentas de teste necessárias.
- Disponibilidade de dados de teste e outros recursos necessários.

Os critérios típicos de saída incluem:

- Que os testes planejados foram executados.
- Foi alcançado um nível definido de cobertura (p. ex., de requisitos, histórias de usuários, critérios de aceite, riscos, código).
- O número de defeitos não resolvidos está dentro de um limite acordado.
- O número de defeitos remanescentes estimados é suficientemente baixo.
- Os níveis avaliados de confiabilidade, eficiência de performance, usabilidade, segurança e outras características de qualidade relevantes são suficientes.

Mesmo sem que os critérios de saída sejam satisfeitos, também é comum que as atividades de teste sejam reduzidas devido ao orçamento gasto, à conclusão do prazo programado ou da pressão para levar o produto ao mercado. Pode ser aceitável encerrar os testes sob tais circunstâncias, se os

*stakeholders* e os donos das empresas do projeto tiverem revisado e aceito o risco de entrar em vigor sem mais testes.

### 5.2.4 Cronograma de execução de teste

Depois que vários casos de teste e procedimentos de teste são produzidos (com alguns procedimentos de teste potencialmente automatizados) e montados em suítes de testes, os conjuntos de testes podem ser organizados em um cronograma da execução do teste que define a ordem em que devem ser executados. O cronograma deve levar em consideração fatores como priorização, dependências, testes de confirmação, testes de regressão e a sequência mais eficiente para se executar os testes.

O ideal seria que os casos de teste fossem ordenados para serem executados com base em seus níveis de prioridade, geralmente executando os casos de teste com a prioridade mais alta primeiro. No entanto, essa prática pode não funcionar se os casos de teste tiverem dependências ou os recursos que estão sendo testados tiverem dependências. Se um caso de teste com prioridade mais alta depender de um caso de teste com prioridade mais baixa, o caso de teste de prioridade mais baixa deverá ser executado primeiro. Da mesma forma, se houver dependências entre os casos de teste, elas devem ser ordenadas adequadamente, independentemente de suas prioridades relativas. Os testes de confirmação e regressão também devem ser priorizados, com base na importância de feedback rápido sobre as mudanças, mas aqui novamente as dependências podem ser aplicadas.

Em alguns casos, várias sequências de testes são possíveis, com diferentes níveis de eficiência associados a essas sequências. Nesses casos, as compensações entre a eficiência da execução do teste e a aderência à priorização devem ser feitas.

### 5.2.5 Fatores que influenciam o esforço do teste

A estimativa de esforço do teste envolve a previsão da quantidade de trabalho relacionado ao teste que será necessário para atender aos objetivos do teste de um projeto, release ou iteração em particular. Os fatores que influenciam o esforço de teste podem incluir características do produto, características do processo de desenvolvimento, características das pessoas e resultados do teste, como mostrado abaixo.

#### Características do produto

- Riscos associados com o produto.
- Qualidade da base de teste
- Tamanho do produto
- Complexidade do domínio do produto
- Requisitos das características de qualidade (p. ex., segurança, confiabilidade)
- Nível necessário de detalhes para documentação de teste
- Requisitos para conformidade legal e regulatória

### Características do processo de desenvolvimento

- Estabilidade e maturidade da organização.
- O modelo de desenvolvimento em uso.
- A abordagem de teste.
- As ferramentas usadas.
- O processo de teste.
- A pressão sobre o tempo de finalização.

### Características das pessoas

- As habilidades e a experiência das pessoas envolvidas, especialmente com projetos e produtos semelhantes (p. ex., conhecimento de domínio).
- Coesão e liderança da equipe.

### Resultados dos testes

- O número e a gravidade dos defeitos encontrados.
- A quantidade de retrabalho necessário.

### 5.2.6 Técnicas de estimativa de teste

Existem várias técnicas de estimativa usadas para determinar o esforço necessário adequado para os testes. Duas das mais utilizadas são:

- **Técnica baseada em métricas**: estimar o esforço do teste com base nas métricas de projetos anteriores, ou com base em valores típicos.
- **Técnica baseada em especialistas**: estimar o esforço do teste com base na experiência dos responsáveis pelas tarefas de teste ou por especialistas.

Por exemplo, no desenvolvimento ágil, os gráficos *burndown* são exemplos da técnica baseada em métricas à medida que o esforço é capturado e relatado e, em seguida, usado para alimentar a velocidade da equipe para determinar a quantidade de trabalho que ela pode realizar na próxima iteração; considerando que o *planning poker* é um exemplo da técnica baseada em especialistas, já que os membros da equipe estão estimando o esforço para entregar um recurso com base em sua experiência. [ISTQB_FL_AT]

Em projetos sequenciais, os modelos de remoção de defeitos são exemplos da técnica baseada em métricas, onde o volume de defeitos e o tempo para removê-los são capturados e relatados, o que fornece uma base para estimar projetos futuros de natureza semelhante. Considerando que a técnica de estimativa *Wideband Delphi* é um exemplo da técnica baseada em especialistas, na qual grupos de especialistas fornecem estimativas com base em sua experiência. [ISTQB_AL_TM]

## 5.3 Monitoramento e controle dos testes

O objetivo do monitoramento de testes é coletar informações e fornecer feedback e visibilidade sobre as atividades de teste. As informações a serem monitoradas podem ser coletadas manualmente ou

automaticamente e devem ser usadas para avaliar o progresso do teste e medir se os critérios de saída ou as tarefas associadas à definição de um projeto Ágil são atendidos, como atingir as metas de cobertura riscos de produtos, requisitos ou critérios de aceite.

O controle do teste descreve quaisquer ações orientadoras ou corretivas tomadas como resultado de informações e métricas coletadas e (possivelmente) relatadas. As ações podem cobrir qualquer atividade de teste e podem afetar qualquer outra atividade do ciclo de vida do software.

Exemplos de ações de controle de teste incluem:

- Repriorizar os testes quando ocorrer um risco identificado (p. ex., software entregue tarde);
- Alterar o cronograma do teste devido à disponibilidade ou indisponibilidade de um ambiente de teste ou outros recursos;
- Reavaliar se um item de teste atende a um critério de entrada ou saída devido ao retrabalho.

### 5.3.1 Métricas usadas no teste

As métricas podem ser coletadas durante e no final das atividades de teste para avaliar:

- Relação entre o planejado e o orçado em um cronograma;
- Qualidade atual do objeto de teste;
- Adequação da abordagem de teste;
- Eficácia das atividades de teste em relação aos objetivos.

As métricas de teste comuns incluem:

- Porcentagem do trabalho planejado executado na preparação do caso de teste (ou porcentagem de casos de teste planejados implementados);
- Porcentagem do trabalho planejado executado na preparação do ambiente de teste;
- Execução de caso de teste (p. ex., número de casos de teste executados/não executados, casos de teste aprovados/com falha ou condições de teste aprovadas/com falha);
- Informações sobre defeitos (p. ex., densidade de defeitos, defeitos encontrados e corrigidos, taxa de falhas e resultados de testes de confirmação);
- Cobertura de teste de requisitos, de histórias de usuários, de critérios de aceite, de riscos ou de código;
- Conclusão de tarefas, alocação e uso de recursos e esforço;
- Custo do teste, incluindo o custo comparado ao benefício de encontrar o próximo defeito ou o custo comparado ao benefício de executar o próximo teste.

### 5.3.2 Finalidades, conteúdo e públicos-alvo para os relatórios de teste

O objetivo do relatório de teste é resumir e comunicar informações de atividade de teste, durante e no final de uma atividade de teste (p. ex., um nível de teste). O relatório de teste preparado durante uma atividade de teste pode ser referido como um relatório de progresso do teste, enquanto um relatório de teste preparado no final de uma atividade de teste pode ser referido como um relatório de resumo do teste.

Durante o monitoramento e controle do teste, o gerente de testes publica regularmente os relatórios de progresso de teste para os *stakeholders*. Além do conteúdo comum nos relatórios de progresso e relatórios de resumo de teste, os relatórios de progresso de teste típicos também podem incluir:

- O status das atividades de teste e o progresso em relação ao plano de teste;
- Fatores impedindo o avanço;
- O teste planejado para o próximo período do relatório;
- A qualidade do objeto de teste.

Quando os critérios de saída são atingidos, o gerente de teste faz o relatório de resumo do teste. Este relatório fornece um resumo dos testes realizados, com base no último relatório de progresso de teste e qualquer outra informação relevante.

Os relatórios típicos de progresso de testes e relatórios de resumo de testes podem incluir:

- Resumo dos testes realizados;
- Informações sobre o que ocorreu durante um período de teste;
- Desvios do plano, incluindo desvios no cronograma, duração ou esforço das atividades de teste.
- Status do teste e da qualidade do produto com relação aos critérios de saída ou definição de completo;
- Fatores que bloquearam ou continuam bloqueando o progresso;
- Métricas de defeitos, casos de teste, cobertura de teste, progresso da atividade e consumo de recursos. (p. ex., conforme descrito em 5.3.1);
- Riscos residuais (ver capítulo 5.5);
- Produtos de teste reutilizáveis produzidos.

O conteúdo de um relatório de teste irá variar dependendo do projeto, dos requisitos organizacionais e do ciclo de vida de desenvolvimento de software. Por exemplo, um projeto complexo com muitos *stakeholders* ou um projeto regulamentado pode exigir relatórios mais detalhados e rigorosos do que uma rápida atualização de software. Como outro exemplo, no desenvolvimento ágil, relatórios de progresso de testes podem ser incorporados em quadros de tarefas, resumos de defeitos e gráficos *burndown*, que podem ser discutidos durante uma reunião *stand-up* diária. [ISTQB_FL_AT]

Além de personalizar os relatórios de teste com base no contexto do projeto, os relatórios de teste devem ser personalizados com base no público do relatório. O tipo e a quantidade de informações que devem ser incluídas para uma audiência técnica ou uma equipe de teste podem ser diferentes daquelas que seriam incluídas em um relatório de resumo executivo. No primeiro caso, informações detalhadas sobre tipos e tendências de defeitos podem ser importantes. No último caso, um relatório de alto nível (p. ex., um resumo de status de defeitos por prioridade, orçamento, cronograma e condições de teste aprovados/reprovados/não testados) pode ser mais apropriado.

O padrão [ISO29119-3] refere-se a dois tipos de relatórios de teste, relatórios de progresso de testes e relatórios de conclusão do teste (chamados de relatórios de resumo de teste neste *syllabus*) e contém estruturas e exemplos para cada tipo.

## 5.4 Gerenciamento de configurações

O objetivo do gerenciamento de configuração é estabelecer e manter a integridade do componente ou do sistema, o testware e seus relacionamentos entre si durante o ciclo de vida do projeto e do produto.

Para suportar adequadamente o teste, o gerenciamento de configuração pode envolver o seguinte:

- Todos os itens de teste são identificados de forma exclusiva, controlados por versão, rastreados para alterações e relacionados entre si;
- Todos os itens de testware são identificados de forma exclusiva, controlados por versão, rastreados para alterações, relacionados uns aos outros e relacionados às versões dos itens de teste, de forma que a rastreabilidade possa ser mantida durante todo o processo de teste;
- Todos os documentos e itens de software identificados são referenciados sem ambiguidade na documentação de teste.

Durante o planejamento do teste, os procedimentos de gerenciamento de configuração e a infraestrutura (ferramentas) devem ser identificados e implementados.

## 5.5 Riscos e testes

### 5.5.1 Definição de risco

O risco envolve a possibilidade da ocorrência futura de um evento com consequências negativas. O nível de risco é determinado pela probabilidade do evento e pelo impacto (o dano) desse evento.

### 5.5.2 Riscos de produtos e projetos

O risco do produto envolve a possibilidade de que um produto de trabalho (p. ex., uma especificação, componente, sistema ou teste) possa falhar em satisfazer as necessidades legítimas de seus usuários ou *stakeholders*. Quando os riscos do produto estão associados às características específicas de qualidade de um produto (p. ex., adequação funcional, confiabilidade, eficiência de performance, usabilidade, segurança, compatibilidade, manutenibilidade e portabilidade), também são chamados de riscos de qualidade. Exemplos de riscos do produto incluem:

- O software pode não executar as funções de acordo com a sua especificação;
- O software pode não executar as funções pretendidas de acordo com as necessidades do usuário, do cliente ou dos *stakeholders*;
- Uma arquitetura de sistema pode não suportar adequadamente alguns requisitos não-funcionais;
- Um cálculo específico pode ser executado incorretamente em algumas circunstâncias.
- Uma estrutura de controle de loop pode ser codificada incorretamente;
- Os tempos de resposta podem ser inadequados para um sistema de processamento de transações de alta performance;
- O feedback da experiência do usuário (UX) pode não atender às expectativas do produto.

O risco do projeto envolve situações que, caso ocorram, podem ter um efeito negativo na capacidade de um projeto atingir seus objetivos. Exemplos de riscos do projeto incluem:

**Questões do projeto:**

- Atrasos podem ocorrer na entrega, na conclusão da tarefa ou na satisfação dos critérios de saída ou definição de *feito*;
- Estimativas imprecisas, realocação de fundos para projetos de maior prioridade ou corte geral de custos na organização podem resultar em recursos financeiros inadequados;
- Alterações tardias podem resultar em um substancial retrabalho.

**Questões organizacionais:**

- Insuficiência de equipe, habilidades e treinamento;
- Questões pessoais podem causar problemas e conflitos;
- Prioridades comerciais conflitantes podem causar indisponibilidade de usuários, equipe de negócios ou especialistas no assunto.

**Questões políticas:**

- Os testadores podem não comunicar adequadamente suas necessidades ou os resultados do teste;
- Os desenvolvedores ou testadores podem falhar em acompanhar as informações encontradas nos testes e revisões (p. ex., não melhorar as práticas de desenvolvimento e teste);
- Pode haver uma atitude imprópria em relação às expectativas de testes (p. ex., não apreciar o valor de encontrar defeitos durante o teste).

**Questões técnicas:**

- Os requisitos podem não estar bem definidos;
- Os requisitos podem não ser cumpridos por restrições existentes;
- O ambiente de teste pode não estar pronto no prazo;
- A conversão de dados, o planejamento de migração e o suporte de ferramentas podem atrasar;
- Fraquezas no processo de desenvolvimento podem afetar a consistência ou a qualidade dos produtos de trabalho do projeto, como desenho, código, configuração, dados de teste e casos de teste;
- Má gestão de defeitos e problemas semelhantes podem resultar em defeitos acumulados e outras obrigações técnicas;

**Questões de fornecedores:**

- Um terceiro pode deixar de entregar um produto ou serviço necessário ou ir à falência.
- Questões contratuais podem causar problemas ao projeto.

Os riscos do projeto podem afetar tanto as atividades de desenvolvimento quanto as atividades de teste. Em alguns casos, os gerentes de projeto são responsáveis por lidar com todos os riscos do

projeto, mas não é incomum que os gerentes de teste tenham responsabilidade pelos riscos do projeto relacionados ao teste.

### 5.5.3 Teste baseado em risco e qualidade do produto

O risco é usado para concentrar o esforço necessário durante o teste. Ele é usado para decidir onde e quando começar a testar e identificar áreas que precisam de mais atenção. O teste é usado para reduzir a probabilidade da ocorrência de um evento adverso ou para reduzir seu o impacto. O teste é usado como uma atividade de mitigação de risco, para fornecer feedback sobre os riscos identificados, bem como sobre os riscos residuais (não resolvidos).

Uma abordagem de teste baseada em risco fornece oportunidades proativas para reduzir os níveis de risco do produto. Envolve a análise de risco, que inclui a identificação e a avaliação da probabilidade e impacto de cada risco. As informações resultantes sobre o risco do produto são usadas para orientar o planejamento, a especificação, a preparação e a execução dos casos de teste, além do monitoramento e controle dos testes. A análise antecipada dos riscos do produto contribui para o sucesso de um projeto.

Em uma abordagem de teste baseada em risco, os resultados da análise de risco do produto são usados para:

- Determinar as técnicas de teste a serem empregadas;
- Determinar os níveis e tipos específicos de testes a serem realizados (p. ex., testes de segurança, testes de acessibilidade);
- Determinar a extensão do teste a ser realizado;
- Priorizar o teste na tentativa de encontrar antecipadamente os defeitos críticos;
- Determinar se quaisquer atividades além do teste poderiam ser empregadas para reduzir o risco (p. ex., fornecer treinamento para projetistas inexperientes).

O teste baseado em risco baseia-se no conhecimento coletivo e no conhecimento dos *stakeholders* do projeto para realizar a análise de risco do produto. Para garantir que a probabilidade de uma falha seja minimizada, as atividades de gerenciamento de risco fornecem uma abordagem disciplinada para:

- Analisar (e reavaliar regularmente) o que pode dar errado (riscos);
- Determinar quais riscos são importantes para lidar;
- Implementar ações para atenuar esses riscos;
- Planejar a contingência para lidar com os riscos, caso eles se tornem eventos reais.

Além disso, os testes podem identificar novos riscos, ajudar a determinar quais riscos devem ser atenuados e reduzir a incerteza sobre os riscos.

## 5.6 Gerenciamento de defeitos.

Como um dos objetivos do teste é encontrar defeitos, estes devem ser registrado ao serem descobertos. A maneira como os defeitos são registrados pode variar, dependendo do contexto do

componente ou sistema que está sendo testado, do nível de teste e do modelo de ciclo de vida de desenvolvimento de software. Quaisquer defeitos identificados devem ser investigados e rastreados desde a descoberta e classificação até sua resolução (p. ex., correção dos defeitos e testes de confirmação bem-sucedidos da solução, adiamento para uma liberação subsequente, aceite como limitação permanente do produto etc.). Para gerenciar todos os defeitos para serem solucionados, uma organização deve estabelecer um processo de gerenciamento de defeitos que inclua um fluxo de trabalho e regras para sua classificação. Esse processo deve ser acordado com todos os participantes do gerenciamento de defeitos, incluindo designers, desenvolvedores, testadores e proprietários de produtos. Em algumas organizações, o registro e o rastreamento de defeitos podem ser muito informais.

Durante o processo de gerenciamento de defeitos, alguns dos relatórios podem revelar falsos positivos, e não falhas reais devido a defeitos. Por exemplo, um teste pode falhar quando uma conexão de rede é interrompida ou atinge o tempo limite. Esse comportamento não resulta de um defeito no objeto de teste, mas é uma anomalia que precisa ser investigada. Os testadores devem tentar minimizar o número de falsos positivos relatados como defeitos.

Os defeitos podem ser relatados durante a codificação, análise estática, revisões, testes dinâmicos ou uso de um produto de software. Os defeitos podem ser relatados para problemas no código ou nos sistemas operacionais ou em qualquer tipo de documentação, incluindo requisitos, histórias de usuários e critérios de aceite, documentos de desenvolvimento, documentos de teste, manuais do usuário ou guias de instalação. Para ter um processo eficaz e eficiente de gerenciamento de defeitos, as organizações podem definir padrões para os atributos, classificação e fluxo de trabalho de defeitos.

Típicos relatórios de defeitos têm os seguintes objetivos:

- Fornecer aos desenvolvedores e a outros, informações sobre qualquer evento adverso ocorrido, para que possam identificar efeitos específicos, isolar o problema com um teste mínimo de reprodução e corrigir o defeito potencial, conforme necessário ou resolver o problema;
- Fornecer aos gerentes de teste um meio de rastrear a qualidade do produto de trabalho e o impacto no teste (p. ex., se muitos defeitos forem relatados, os testadores terão gastado muito tempo relatando ao invés de executando testes, e serão necessários mais testes de confirmação);
- Fornecer ideias para desenvolvimento e melhoria de processos de teste.

Um relatório de defeitos arquivado durante o teste dinâmico geralmente inclui:

- Um identificador;
- Um título e um breve resumo do defeito relatado;
- A data do relatório de defeitos, a organização emissora e autor;
- A identificação do item de teste (item de configuração sendo testado) e ambiente;
- A(s) fase(s) do ciclo de vida de desenvolvimento em que o defeito foi observado;
- Uma descrição do defeito para permitir a reprodução e resolução, incluindo logs, capturas de tela de *dump* de banco de dados ou gravações (se encontradas durante a execução do teste);

- Resultados esperados e reais;
- Escopo ou grau de impacto (gravidade) do defeito sobre os interesses do(s) *stakeholder*(s);
- Urgência/prioridade para corrigir;
- O estado do relatório de defeitos (p. ex., aberto, diferido, duplicado, aguardando correção, aguardando confirmação, reaberto, fechado);
- Conclusões, recomendações e aprovações;
- Questões globais, como outras áreas que podem ser afetadas por uma alteração resultante do defeito;
- Histórico de alterações, como a sequência de ações tomadas pelos membros da equipe do projeto com relação ao defeito para isolar, reparar e confirmar como corrigido;
- Referências, incluindo o caso de teste que revelou o problema.

Alguns desses detalhes podem ser automaticamente incluídos ou gerenciados ao usar ferramentas de gerenciamento de defeitos, por exemplo, atribuição automática de um identificador, atribuição e atualização do estado do relatório de defeitos durante o fluxo de trabalho etc. Os defeitos encontrados durante os testes estáticos, particularmente revisões, normalmente serão documentados de uma maneira diferente, por exemplo, nas notas da reunião de revisão.

Um exemplo do conteúdo de um relatório de defeitos pode ser encontrado no padrão [ISO29119-3] (onde se refere a relatórios de defeitos como relatórios de incidentes)

# 6 Ferramenta de suporte ao teste [40 min]

### *Conceitos-chave*

teste orientado por dados, teste orientado por palavras-chave, automação de teste, ferramenta de execução de teste, ferramenta de gerenciamento de teste

### *Objetivos de aprendizagem*

#### *6.1 Considerações sobre as ferramentas de teste*

FL-6.1.1 (K2) Classificar as ferramentas de teste de acordo com a sua finalidade e as atividades de teste que elas suportam.

FL-6.1.2 (K1) Identificar os benefícios e riscos da automação de testes.

FL-6.1.3 (K1) Lembrar-se de considerações especiais para a execução do teste e as ferramentas de gerenciamento.

#### *6.2 Uso efetivo de ferramentas*

FL-6.2.1 (K1) Identificar os principais princípios para selecionar uma ferramenta.

FL-6.2.2 (K1) Relembrar os objetivos de usar projetos piloto para introduzir ferramentas

FL-6.2.3 (K1) Identificar os fatores de sucesso para avaliação, implementação, implantação e suporte contínuo de ferramentas de teste em uma organização.

## 6.1 Considerações sobre a ferramenta de teste

As ferramentas de teste podem ser usadas para suportar uma ou mais atividades de teste. Tais ferramentas incluem:

- Ferramentas que são usadas diretamente nos testes, como ferramentas de execução e ferramentas de preparação de dados de teste;
- Ferramentas que ajudam a gerenciar os requisitos, os casos de teste, os procedimentos de teste, os scripts de teste automatizados, os resultados de testes, os dados de teste e defeitos, e para relatórios e monitoramento da execução dos testes;
- Ferramentas que são usadas para investigação e avaliação;
- Qualquer ferramenta que auxilie no teste (uma planilha também é uma ferramenta de teste nesse sentido).

### 6.1.1 Classificação das ferramentas de teste

As ferramentas de teste podem ter um ou mais dos seguintes propósitos, dependendo do contexto:

- Melhorar a eficiência das atividades de teste automatizando tarefas repetitivas ou tarefas que exigem recursos significativos quando feitas manualmente (p. ex., execução do teste, teste de regressão);
- Melhorar a eficiência das atividades de teste, apoiando as atividades de teste manuais durante todo o processo (ver capítulo 1.4);
- Melhorar a qualidade das atividades de teste, permitindo testes mais consistentes e um nível mais alto de reprodutibilidade de defeitos;
- Automatizar as atividades que não podem ser executadas manualmente (p. ex., testes de performance em grande escala);
- Aumentar a confiabilidade dos testes (p. ex., automatizando comparações de dados grandes ou simulando comportamentos).

As ferramentas podem ser classificadas com base em vários critérios, como finalidade, preço, modelo de licenciamento (p. ex., comercial ou código aberto) e a tecnologia utilizada. As ferramentas são classificadas neste syllabus de acordo com as atividades de teste que eles suportam.

Algumas ferramentas suportam claramente apenas ou principalmente uma atividade; outros podem apoiar mais de uma atividade, mas são classificados na atividade com a qual estão mais intimamente associados. As ferramentas de um único provedor, especialmente aquelas que foram projetadas para funcionarem juntas, podem ser fornecidas como um conjunto integrado.

Alguns tipos de ferramentas de teste podem ser intrusivos, o que significa que podem afetar o resultado real do teste. Por exemplo, os tempos de resposta reais para um aplicativo podem ser diferentes devido às instruções extras executadas por uma ferramenta de teste de performance ou a quantidade de cobertura de código alcançada pode ser distorcida devido ao uso de uma ferramenta de cobertura. A consequência do uso de ferramentas intrusivas é chamada de efeito de monitoração.

Algumas ferramentas oferecem suporte que é tipicamente mais apropriado para desenvolvedores (p. ex., ferramentas que são usadas durante testes de componentes e integração). Essas ferramentas estão marcadas com "[D]" nas seções abaixo.

### Ferramentas de suporte para gerenciamento de testes e testware

As ferramentas de gerenciamento podem ser aplicadas em qualquer atividade de teste durante todo o ciclo de vida de desenvolvimento de software. Os exemplos de ferramentas que suportam o gerenciamento de testes e testware incluem:

- Ferramentas de gerenciamento de teste e ferramentas de gerenciamento de ciclo de vida de aplicativos (ALM);
- Ferramentas de gerenciamento de requisitos (p. ex., rastreabilidade para testar objetos);
- Ferramentas de gerenciamento de defeitos;
- Ferramentas de gerenciamento de configuração;
- Ferramentas de integração contínua [D].

### Ferramentas de suporte ao teste estático

As ferramentas de teste estático estão associadas às atividades e benefícios descritos no capítulo 3. Exemplos de tais ferramentas incluem:

- Ferramentas de análise estática [D].

### Ferramentas de suporte para modelagem e implementação dos testes

As ferramentas de projeto de teste auxiliam na criação de produtos de trabalho sustentáveis no projeto e implementação dos testes, incluindo casos de teste, procedimentos de teste e dados de teste. Exemplos de tais ferramentas incluem:

- Ferramentas de teste baseadas em modelo;
- Ferramentas de preparação de dados de teste.

Em alguns casos, as ferramentas que suportam a modelagem e a implementação dos testes também podem suportar a execução e o registro de testes, ou fornecer suas saídas diretamente para outras ferramentas que suportam a execução e o registro de testes.

### Ferramentas de suporte para execução e registro de teste

Existem muitas ferramentas para apoiar e aprimorar as atividades de execução e registro de teste. Exemplos dessas ferramentas incluem:

- Ferramentas de execução do teste (p. ex., para executar testes de regressão);
- Ferramentas de cobertura (p. ex., cobertura de requisitos, cobertura de código [D]);
- Equipamentos de teste [D].

### Ferramentas de suporte para medição de performance e análise dinâmica

As ferramentas de medição de performance e análise dinâmica são essenciais para dar suporte às atividades de performance e de teste de carga, pois essas atividades não podem ser feitas manualmente com eficiência. Exemplos dessas ferramentas incluem:

- Ferramentas de teste de performance;
- Ferramentas de análise dinâmica [D].

### Ferramentas de suporte para necessidades de testes especiais

Além das ferramentas que suportam o processo geral de teste, existem muitas outras ferramentas que suportam testes mais específicos para características não funcionais.

## 6.1.2 Benefícios e riscos da automação de testes

Simplesmente adquirir uma ferramenta não garante o sucesso de seu projeto. Cada nova ferramenta introduzida em uma organização exigirá esforço para obter benefícios reais e duradouros. Existem benefícios e oportunidades potenciais com o uso de ferramentas nos testes, mas também há riscos. Isso é particularmente verdadeiro para as ferramentas de execução do teste (que geralmente são chamadas de automação de teste).

Os benefícios potenciais do uso de ferramentas para suporte à execução dos testes incluem:

- Redução no trabalho manual repetitivo (p.e, executando testes de regressão, tarefas de configuração e desmontagem de ambiente, redigitação dos mesmos dados de teste e verificação dos padrões de codificação), economizando tempo;
- Maior consistência e repetibilidade (p. ex., os dados de teste são criados de maneira coerente, os testes são executados por uma ferramenta na mesma ordem com a mesma frequência e os testes são consistentemente derivados dos requisitos);
- Avaliação mais objetiva (p. ex., medidas estáticas, cobertura);
- Acesso mais fácil a informações sobre testes (p. ex., estatísticas e gráficos sobre o progresso do teste, taxas de defeitos e performance).

Riscos potenciais no uso de ferramentas de suporte aos testes incluem:

- Expectativas irreais para a ferramenta (incluindo funcionalidade e facilidade de uso);
- O tempo, o custo e o esforço para a implantação de uma ferramenta podem ser subestimados (incluindo treinamento e experiência externa);
- O tempo e o esforço necessários para obter os benefícios significativos e contínuos da ferramenta podem ser subestimados (incluindo a necessidade de alterações no processo de teste e melhoria contínua na maneira como a ferramenta é usada);
- O esforço necessário para manter os ativos de teste gerados pela ferramenta pode ser subestimado;
- A ferramenta pode ser usada em demasia (vista como um substituto para o projeto ou execução dos testes, ou o uso de testes automatizados onde o teste manual seria melhor);
- O controle de versão dos ativos de teste pode ser negligenciado;

- Os relacionamentos e os problemas de interoperabilidade entre ferramentas críticas podem ser negligenciados, como ferramentas de gerenciamento de requisitos, ferramentas de gerenciamento de configuração, ferramentas de gerenciamento de defeitos e ferramentas de vários fornecedores;
- O fornecedor da ferramenta pode sair do negócio, aposentando a ferramenta ou vendendo seu código para um fornecedor diferente;
- O fornecedor pode possuir um suporte insatisfatório à ferramenta, às atualizações e as correções de defeitos;
- Um projeto de código aberto pode ser suspenso;
- Uma nova plataforma ou tecnologia pode não ser suportada pela ferramenta;
- Pode não haver uma propriedade clara da ferramenta (p. ex., para orientação, atualizações etc.).

### 6.1.3 Considerações especiais para execução de testes e ferramentas de gerenciamento de testes

Para ter uma implementação tranquila e bem-sucedida, há várias coisas que devem ser consideradas ao selecionar e integrar ferramentas de execução do teste e gerenciamento de teste em uma organização.

#### Ferramentas de execução do teste

As ferramentas de execução do teste executam objetos de teste usando scripts de teste automatizados. Esse tipo de ferramenta geralmente requer um esforço significativo para obter benefícios.

- **Abordagem de captura de teste**: Capturar testes gravando as ações de um testador manual parece atraente, mas essa abordagem não pode ser dimensionada para muitos scripts de teste. Um script capturado é uma representação linear com dados e ações específicos como parte de cada script. Esse tipo de script pode ser instável quando ocorrem eventos inesperados e exigir manutenção contínua à medida que a interface do usuário do sistema evolui com o tempo.
- **Abordagem de teste orientada por dados**: essa abordagem separa as entradas de teste e os resultados esperados, geralmente em uma planilha, e usa um script de teste mais genérico que pode ler os dados de entrada e executar o mesmo script de teste com dados diferentes.
- **Abordagem de teste orientada por palavras-chave**: essa abordagem de teste, um script genérico processa palavras-chave que descrevem as ações a serem executadas, que depois chama scripts de palavras-chave para processar os dados de teste associados.

As abordagens acima exigem que alguém tenha experiência na linguagem de script (testadores, desenvolvedores ou especialistas em automação de teste). Ao usar as abordagens de teste controladas por dados ou por palavras-chave, os testadores que não estão familiarizados com a linguagem de script também podem contribuir criando dados de teste ou palavras-chave para esses scripts predefinidos. Independentemente da técnica de script usada, os resultados esperados para

cada teste precisam ser comparados aos resultados reais do teste, dinamicamente (enquanto o teste está em execução) ou armazenados para comparação posterior (pós-execução).

Mais detalhes e exemplos de abordagens de testes orientados por dados e por palavras-chave são fornecidos no [ISTQB_AL_TAE], Fewster 1999 e Buwalda 2001.

As ferramentas de teste baseado em modelo (MBT) permitem que uma especificação funcional seja capturada na forma de um modelo, como um diagrama de atividades. Essa tarefa geralmente é executada por um arquiteto de sistema. A ferramenta MBT interpreta o modelo para criar especificações de caso de teste que podem ser salvas em uma ferramenta de gerenciamento de testes ou executadas por uma ferramenta de execução de teste (consulte [ISTQB_FL_MBT]).

#### Ferramentas de gerenciamento de teste

As ferramentas de gerenciamento de teste geralmente precisam interagir com outras ferramentas ou planilhas por vários motivos, incluindo:

- Para produzir informações úteis em um formato que atenda às necessidades da organização;
- Manter rastreabilidade consistente para os requisitos em uma ferramenta de gerenciamento de requisitos;
- Para vincular informações da versão do objeto de teste na ferramenta de gerenciamento de configuração.

Isso é particularmente importante quando usar uma ferramenta integrada (p. ex., gerenciamento de ciclo de vida de aplicativos), que inclui um módulo de gerenciamento de teste (e possivelmente um sistema de gerenciamento de defeitos), além de outros módulos (p. ex., cronograma do projeto e informações orçamentárias) usados por diferentes grupos dentro de uma organização.

## 6.2 Uso eficaz de ferramentas

### 6.2.1 Principais considerações para a escolha de ferramentas

As principais considerações para a escolha de uma ferramenta ideal para uma organização incluem:

- Avaliação da maturidade da organização, seus pontos fortes e fracos;
- Identificação de oportunidades para um processo de teste melhorado suportado por ferramentas;
- Compreensão das tecnologias usadas pelo(s) objeto(s) de teste, a fim de selecionar uma ferramenta que seja compatível com essa tecnologia;
- As ferramentas de integração contínua e construção já em uso dentro da organização, a fim de garantir a compatibilidade e integração de ferramentas;
- Avaliação da ferramenta contra requisitos claros e critérios objetivos;
- Consideração sobre se a ferramenta está ou não disponível (e por quanto tempo ficará) em um período de teste gratuito;
- Avaliação do fornecedor (incluindo treinamento, suporte e aspectos comerciais) ou suporte para ferramentas não comerciais (p. ex., código aberto);

- Identificação de requisitos internos para treinamento e *mentoring* no uso da ferramenta;
- Avaliação das necessidades de treinamento, considerando as habilidades de teste (e automação de testes) daqueles que trabalharão diretamente com a(s) ferramenta(s);
- Consideração de prós e contras de vários modelos de licenciamento (p. ex., comercial ou *open source*);
- Estimativa de uma relação custo-benefício baseada em um caso de negócios concreto (se necessário).

Como etapa final, uma prova de conceito deve ser feita para estabelecer se a ferramenta funciona efetivamente com o software em teste e dentro da infraestrutura atual ou, se necessário, para identificar as alterações necessárias para que a infraestrutura se adapte à ferramenta de forma eficaz.

### 6.2.2 Projeto piloto para introduzir uma ferramenta em uma organização

Depois de concluir a seleção de ferramentas e uma prova de conceito bem-sucedida, a introdução da ferramenta selecionada em uma organização geralmente começa com um projeto piloto, que tem os seguintes objetivos:

- Obter conhecimento aprofundado sobre a ferramenta, entendendo seus pontos fortes e fracos;
- Avaliar como a ferramenta se ajusta aos processos e práticas existentes e determinar o que precisaria mudar;
- Decidir sobre a forma padrão de usar, gerenciar, armazenar e manter a ferramenta e os recursos de teste (p. ex., decidir sobre convenções de nomenclatura de arquivos e testes, selecionar padrões de codificação, criar bibliotecas e definir a modularidade dos conjuntos de testes);
- Avaliar se os benefícios serão alcançados a um custo razoável;
- Compreender as métricas que você deseja que a ferramenta colete e relate e configure a ferramenta para garantir que essas métricas possam ser capturadas e relatadas.

### 6.2.3 Fatores de sucesso para ferramentas

Fatores de sucesso para avaliação, implementação, implantação e suporte contínuo de ferramentas dentro de uma organização incluem:

- Estender o uso da ferramenta para o resto da organização de forma incremental;
- Adaptar e melhorar os processos para se adequar ao uso da ferramenta;
- Fornecer treinamento, capacitação e orientação para os usuários das ferramentas;
- Definir as diretrizes para o uso da ferramenta (p. ex., padrões internos para automação);
- Implementar uma forma de coletar informações de uso real da ferramenta;
- Usar e beneficiar-se da ferramenta de monitoramento;
- Fornecer suporte aos usuários de uma determinada ferramenta;
- Reunir lições aprendidas de todos os usuários.

Também é importante garantir que a ferramenta seja técnica e organizacionalmente integrada ao ciclo de vida de desenvolvimento de software, o que pode envolver organizações separadas responsáveis por operações ou fornecedores terceirizados.

Veja Graham (2012) para experiências e conselhos sobre o uso de ferramentas de execução do teste.

# Referências

## Normas e Padrões

**[ISO29119-1]** ISO/IEC/IEEE 29119-1 (2013) Software and systems engineering, Software testing, Part 1, Concepts and definitions

**[ISO29119-2]** ISO/IEC/IEEE 29119-2 (2013) Software and systems engineering, Software testing, Part 2, Test processes

**[ISO29119-3]** ISO/IEC/IEEE 29119-3 (2013) Software and systems engineering, Software testing, Part 3, Test documentation

**[ISO29119-4]** ISO/IEC/IEEE 29119-4 (2015) Software and systems engineering, Software testing, Part 4, Test techniques

**[ISO25010]** ISO/IEC 25010, (2011) Systems and software engineering, Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

**[ISO20246]** ISO/IEC 20246: (2017) Software and systems engineering, Work product reviews

**[UML 2.5]** UML 2.5, Unified Modeling Language Reference Manual, http://www.omg.org/spec/UML/2.5.1/, 2017

## Documentos ISTQB®

**[ISTQB_FL_OVER]** ISTQB® Foundation Level Overview, 2018

**[ISTQB_FL]** ISTQB® CTFL Foundation Level, 2018br, https://www.bstqb.org.br/sobre-ctfl

**[ISTQB_FL_AT]** ISTQB® CTFL-AT Agile Tester, 2014br, https://www.bstqb.org.br/sobre-ctfl-at

**[ISTQB_FL_UT]** ISTQB® CTFL-UT Usability Testing, 2018br, https://www.bstqb.org.br/sobre-ctfl-ut

**[ISTQB_FL_PT]** ISTQB® CTFL-PT Performance Testing, 2018br, https://www.bstqb.org.br/sobre-ctfl-pt

**[ISTQB_FL_MAT]** ISTQB® CTFL-MAT Mobile Application Testing, 2019br, https://www.bstqb.org.br/sobre-ctfl-mat

**[ISTQB_FL_MBT]** ISTQB® CTFL-MBT Model-Based Testing, 2015br, https://www.bstqb.org.br/sobre-ctfl-mbt

**[ISTQB_ATT_OVIEW]** ISTQB® Advanced Agile Technical Tester Overview, 2019br

**[ISTQB_AL_TA]** ISTQB® CTAL-TA Test Analyst,2019br, https://www.bstqb.org.br/sobre-ctal-ta

**[ISTQB_AL_TM]** ISTQB® CTAL-TM Test Manager, 2012br, https://www.bstqb.org.br/sobre-ctal-tm

**[ISTQB_AL_SEC]** ISTQB® CTAL-SEC Security Testing, 2016br, https://www.bstqb.org.br/sobre-ctal-sec

**[ISTQB_AL_TAE]** ISTQB® CTAL-TAE Test Automation Engineer, 2017br, https://www.bstqb.org.br/sobre-ctal-tae

**[ISTQB_GLOSSARY]** ISTQB® Glossary of Terms used in Software Testing, v3.2, 2019br, https://glossary.istqb.org

## Livros de Artigos

**Beizer**, B. (1990) Software Testing Techniques (2e), Van Nostrand Reinhold: USA

**Black**, R. (2017) Agile Testing Foundations, BCS Learning & Development Ltd: UK

**Black**, R. (2009) Managing the Testing Process (3e), John Wiley & Sons: New York NY

**Buwalda**, H. et al. (2001) Integrated Test Design and Automation, Addison Wesley: USA

**Copeland**, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: USA

**Craig**, R. and Jaskiel, S. (2002) Systematic Software Testing, Artech House: USA

**Crispin**, L. and Gregory, J. (2008) Agile Testing, Pearson Education: USA

**Fewster**, M. and Graham, D. (1999) Software Test Automation, Addison Wesley: UK

**Gilb**, T. and Graham, D. (1993) Software Inspection, Addison Wesley: USA

**Graham**, D. and Fewster, M. (2012) Experiences of Test Automation, Pearson Education: USA

**Gregory**, J. and Crispin, L. (2015) More Agile Testing, Pearson Education: USA

**Jorgensen**, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: USA

**Kaner**, C., Bach, J. and Pettichord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons: USA

**Kaner**, C., Padmanabhan, S. and H offman, D. (2013) The Domain Testing Workbook, Context-Driven Press: USA

**Kramer**, A., Legeard, B. (2016) Model-Based Testing Essentials: Guide to the ISTQB® Certified ModelBased Tester: Foundation Level, John Wiley & Sons: USA

**Myers**, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: USA

**Sauer**, C. (2000) "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research," IEEE Transactions on Software Engineering, Volume 26, Issue 1, pp 1

**Shull**, F., Rus, I., Basili, V. July 2000. "How Perspective-Based Reading can Improve Requirement Inspections." IEEE Computer, Volume 33, Issue 7, pp 73-79

**van Veenendaal**, E. (2004) The Testing Practitioner (Chapters 8-10), UTN Publishers: The Netherlands

**Wiegers**, K. (2002) Peer Reviews in Software, Pearson Education: USA

**Weinberg**, G. (2008) Perfect Software and Other Illusions about Testing, Dorset House: USA

### Outras fontes

**Black**, R., van Veenendaal, E. and Graham, D. (2012) *Foundations of Software Testing: ISTQB Certification (3e)*, Cengage Learning: UK

**Hetzel**, W. (1993) *Complete Guide to Software Testing (2e)*, QED Information Sciences: USA

# [Apêncide A] Histórico desse syllabus

## História deste Documento

Este documento é o *ISTQB® CTFL Foundation Level* Syllabus, a qualificação internacional de nível fundamental aprovada pelo *ISTQB®* (www.istqb.org).

Este documento foi preparado entre junho e agosto de 2019 por um grupo de trabalho composto por membros nomeados pelo International Software Testing Qualifications Board (*ISTQB®*). As atualizações foram adicionadas após a análise dos conselhos membros que usaram a versão 2018 do syllabus.

O Syllabus Foundation 2018 anterior foi preparado entre 2014 e 2018 por um grupo de trabalho composto por membros nomeados pelo *International Software Testing Qualifications Board* (*ISTQB®*). A versão final foi revisada inicialmente por representantes de todos os conselhos membros do *ISTQB®* e, depois, por representantes da comunidade internacional de testes de software.

## Objetivos da qualificação da certificação do nível fundamental

- Obter reconhecimento para testes como uma especialização essencial e profissional em engenharia de software;
- Fornecer uma estrutura padrão para o desenvolvimento das carreiras dos testadores;
- Permitir que os testadores profissionalmente certificados sejam reconhecidos pelos empregadores, clientes e colegas, e para aumentar o perfil dos testadores;
- Promover boas práticas de testes consistentes em todas as disciplinas de engenharia de software
- Identificar os tópicos de teste relevantes e de valor para o setor;
- Permitir que fornecedores de software contratem testadores certificados e, assim, obter vantagem comercial sobre seus concorrentes, anunciando sua política de recrutamento de testadores;
- Proporcionar uma oportunidade para que os testadores e aqueles com interesse em testes adquiram uma qualificação internacionalmente reconhecida no assunto.

## Objetivos da certificação internacional

- Ser capaz de comparar o conhecimento de testes em diferentes países;
- Para permitir que os testadores se movimentem através das fronteiras do país com mais facilidade;
- Permitir que projetos internacionais tenham um entendimento comum dos problemas de teste
- Aumentar o número de testadores qualificados em todo o mundo;
- Ter mais impacto/valor como uma iniciativa baseada internacionalmente do que em qualquer abordagem específica do país;
- Desenvolver um corpo internacional comum de compreensão e conhecimento sobre testes através do *syllabus* e da terminologia, e para aumentar o nível de conhecimento sobre testes para todos os participantes;

- Promover o teste como profissão em mais países;
- Para permitir que os testadores obtenham uma qualificação reconhecida em seu idioma nativo
- Permitir o compartilhamento de conhecimento e recursos entre os países;
- Para fornecer reconhecimento internacional de testadores e esta qualificação devido à participação de muitos países.

## Pré-requisitos para esta certificação

O critério de entrada para o exame do nível de certificação do *ISTQB® CTFL Foundation Level* é que os candidatos têm interesse em testes de software. No entanto, é altamente recomendável que os candidatos também:

- Ter pelo menos uma experiência mínima em desenvolvimento de software ou teste de software, como seis meses de experiência como sistema ou testador de aceite do usuário ou como desenvolvedor de software;
- Faça um curso que tenha sido credenciado por um dos conselhos membros reconhecidos pelo *ISTQB®* para os padrões do *ISTQB®*.

## Antecedentes e histórico do CTFL.

A certificação independente de testadores de software começou no Reino Unido com *Information Systems Examination Board* da *British Computer Society* (ISEB), quando um Conselho Teste de Software foi criado em 1998 (www.bcs.org.uk/iseb). Em 2002, o ASQF na Alemanha começou a apoiar um esquema de qualificação de testador alemão (www.asqf.de). Este *syllabus* baseia-se nos *syllabi* ISEB e ASQF; inclui conteúdo reorganizado, atualizado e adicional, e a ênfase é direcionada a tópicos que fornecerão a ajuda mais prática para os testadores.

Um Certificado em Teste de Software de Nível Fundamental existente (p. ex., do ISEB, ASQF ou um quadro de membros reconhecido pelo *ISTQB®*) concedido antes do lançamento deste Certificado Internacional, será considerado equivalente ao Certificado Internacional. O certificado não expira e não precisa ser renovado. A data em que foi concedida é mostrada no Certificado.

Dentro de cada país participante, os aspectos locais são controlados por um Conselho de Teste de Software reconhecido nacional ou regionalmente pelo *ISTQB®*. Os deveres dos quadros membros são especificados pelo *ISTQB®*, mas são implementados dentro de cada país. Os deveres dos conselhos de país devem incluir o credenciamento de provedores de treinamento e a definição de exames.

# [Apendice B] Objetivos de aprendizagem e níveis cognitivos

Os seguintes objetivos de aprendizagem são definidos como aplicáveis a este *syllabus*. Cada tópico do *syllabus* será examinado de acordo com o objetivo de aprendizado para ele.

### *Nível 1: Lembrar (K1)*

O candidato reconhecerá, lembrará e recordará um termo ou conceito.

**Palavras-chave**: Identificar, lembrar, recuperar, recordar, reconhecer, saber

**Exemplo**: Pode-se reconhecer a definição de "falha" como: (1) a não entrega de serviço para um usuário final ou qualquer outra parte interessada; (2) desvio do componente ou sistema de sua entrega, serviço ou resultado esperado

### *Nível 2: Entender (K2)*

O candidato pode selecionar as razões ou explicações para as declarações relacionadas ao tópico e pode resumir, comparar, classificar, categorizar e dar exemplos para o conceito de teste.

**Palavras-chave**: Resumir, generalizar, abstrair, classificar, comparar, mapear, contrastar, exemplificar, interpretar, traduzir, representar, inferir, concluir, categorizar, construir modelos.

**Exemplos**: Explicar o motivo pelo qual a análise e a modelagem do teste devem ocorrer o mais cedo possível: (1) para encontrar defeitos enquanto são mais baratos de se remover; (2) para primeiramente encontrar os defeitos mais importantes.

Explicar as semelhanças e diferenças entre integração e teste de sistema:

- *Semelhanças*: os objetos de teste para testes de integração e de sistema incluem mais de um componente, e os testes de integração e de sistema podem incluir tipos de teste não funcionais.
- *Diferenças*: o teste de integração concentra-se em interfaces e interações, e o teste do sistema concentra-se em aspectos do sistema inteiro, como o processamento de ponta a ponta

### *Nível 3: Aplicar (K3)*

O candidato pode selecionar a aplicação correta de um conceito ou técnica e aplicá-lo em um contexto.

**Palavras-chave**: Implementar, executar, usar, seguir um procedimento, aplicar um procedimento

**Exemplos**:

- Identificar valores de limite para partições válidas e inválidas.
- Selecionar casos de teste de um determinado diagrama de transição de estado para cobrir todas as transições.

### Referência (para os níveis cognitivos dos objetivos de aprendizagem)

Anderson, L. W. e Krathwohl, D. R. (eds) (2001) Uma taxonomia para aprender, ensinar e avaliar: Uma revisão da taxonomia de objetivos educacionais de Bloom, Allyn & Bacon: Boston MA

# [Apêncide C] Notas de release

O *ISTQB® CTFL Foundation Level* Syllabus 2018 V3.1 é uma pequena atualização da versão 2018. Uma nota de versão separada 2018 V3.1 foi criada com uma visão geral por capítulo. Além disso, foi lançada uma versão do Foundation Syllabus 2018 V3.1 com as alterações.

O *ISTQB® CTFL Foundation Level* Syllabus 2018 é uma grande atualização, reescrita a partir do release 2011. Por esse motivo, não há notas de lançamento detalhadas a cada capítulo. No entanto, um resumo das principais alterações é fornecido aqui. Além disso, em um documento separado de notas de versão, o *ISTQB®* fornece rastreabilidade entre os objetivos de aprendizado do *CTFL Foundation Level Syllabus* versão 2011 e 2018, mostrando quais foram adicionados, atualizados ou removidos.

No início de 2017, mais de 550.000 pessoas em mais de 100 países fizeram o exame da fundação e mais de 500.000 são testadores certificados em todo o mundo. Com a expectativa de que todos tenham lido o Foundation Syllabus para passar no exame, isso faz com que o Syllabus CTFL seja o documento de teste de software mais lido de todos os tempos!

Essa grande atualização é feita em relação a este patrimônio e para melhorar o valor que o *ISTQB®* oferece às próximas 500.000 pessoas na comunidade global de testes.

Nesta versão, todos os objetivos de aprendizado foram editados para torná-los atômicos e para criar uma rastreabilidade clara de cada objetivo de aprendizado até o(s) capítulos(s) de conteúdo (e questões do exame) relacionadas a esse objetivo de aprendizado. as seções de conteúdo (e questões do exame) de volta ao objetivo de aprendizagem associado. Além disso, as alocações de tempo dos capítulos foram mais realistas do que aquelas na versão de 2011 do *syllabus*, usando heurísticas e fórmulas usadas com outros *syllabus* do *ISTQB®*, que são baseadas em uma análise dos objetivos de aprendizagem a serem abordados em cada capítulo.

Embora este seja um *syllabus* de nível fundamental, expressando as melhores práticas e técnicas que resistiram ao teste do tempo, fizemos mudanças para modernizar a apresentação do material, especialmente em termos de métodos de desenvolvimento de software (p. ex., Scrum e implantação contínua) e tecnologias (p. ex., a Internet das Coisas). Atualizamos os padrões referenciados para torná-los mais recentes da seguinte forma:

1. ISO / IEC / IEEE 29119 substitui o padrão IEEE 829.
2. ISO / IEC 25010 substitui o ISO 9126.
3. ISO / IEC 20246 substitui o IEEE 1028.

Além disso, uma vez que o portfólio do *ISTQB®* cresceu dramaticamente na última década, adicionamos referências cruzadas extensivas a material relacionado em outros syllabus do *ISTQB®*, quando relevante, bem como revisamos cuidadosamente o alinhamento com todos os *syllabi* e com o Glossário *ISTQB®*. O objetivo é tornar esta versão mais fácil de ler, entender, aprender e traduzir, concentrando-se em aumentar a utilidade prática e o equilíbrio entre conhecimento e habilidades.

Para uma análise detalhada das alterações feitas nesta versão, consulte o [ISTQB_FL_OVER].