# Certified Tester

# Foundation Level Syllabus

# CTFL

## versão 4.0

International Software Testing Qualifications Board


BSTQB

Versão traduzida, adaptada e disponibilizada na Língua Portuguesa pelo *Grupo de Traduções do BSTQB*

# Direitos autorais

# Histórico da Revisão

| Versão | Data | Observações |
|---|---|---|
| CTFL v4.0 | 21/04/2023 | Versão de lançamento |
| CTFL v3.1.1 | 01/07/2021 | Atualização de direitos autorais e logotipo |
| CTFL v3.1 | 11/11/2019 | Versão de manutenção de pequenas atualizações |
| ISTQB 2018 | 27/04/2018 | Versão candidata de lançamento |
| ISTQB 2011 | 01/04/2011 | Versão de manutenção |
| ISTQB 2010 | 30/03/2010 | Versão de manutenção |
| ISTQB 2007 | 01/05/2007 | Versão de manutenção |
| ISTQB 2005 | 01/07/2005 | Versão candidata de lançamento |
| ASQF V2.2 | 07/2003 | ASQF Syllabus Foundation Level versão v2.2 "*Lehrplan Grundlagen des Software-testens*" |
| ISEB V2.0 | 25/02/1999 | ISEB Software Testing Foundation Syllabus v2.0 |
| | | |

## *Histórico da versão BSTQB*

| Versão | Data | Observações |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Índice

# Agradecimentos

Este documento foi formalmente lançado pela General Assembly do ISTQB® em 21 de abril de 2023.

Foi produzido por uma equipe dos grupos de trabalho conjuntos do ISTQB Foundation Level e Agile: Laura Albert, Renzo Cerquozzi (vice chair), Wim Decoutere, Klaudia Dussa-Zieger, Chintaka Indikadahena, Arnika Hryszko, Martin Klonk, Kenji Onishi, Michaël Pilaeten (co-chair), Meile Posthuma, Gandhinee Rajkomar, Stuart Reid, Eric Riou du Cosquer (co-chair), Jean-François Riverin, Adam Roman, Lucjan Stapp, Stephanie Ulrich (vice chair), Eshraka Zakaria.

A equipe agradece a Stuart Reid, Patricia McQuaid and Leanne Howard pela revisão técnica e aos Conselhos Membros por suas sugestões e contribuições.

As seguintes pessoas participaram da revisão, dos comentários e da votação deste syllabus: Adam Roman, Adam Scierski, Ágota Horváth, Ainsley Rood, Ale Rebon Portillo, Alessandro Collino, Alexander Alexandrov, Amanda Logue, Ana Ochoa, André Baumann, André Verschelling, Andreas Spillner, Anna Miazek, Armin Born, Arnd Pehl, Arne Becher, Attila Gyúri, Attila Kovács, Beata Karpinska, Benjamin Timmermans, Blair Mo, Carsten Weise, Chinthaka Indikadahena, Chris Van Bael, Ciaran O'Leary, Claude Zhang, Cristina Sobrero, Dandan Zheng, Dani Almog, Daniel Säther, Daniel van der Zwan, Danilo Magli, Darvay Tamás Béla, Dawn Haynes, Dena Pauletti, Dénes Medzihradszky, Doris Dötzer, Dot Graham, Edward Weller, Erhardt Wunderlich, Eric Riou Du Cosquer, Florian Fieber, Fran O'Hara, François Vaillancourt, Frans Dijkman, Gabriele Haller, Gary Mogyorodi, Georg Sehl, Géza Bujdosó, Giancarlo Tomasig, Giorgio Pisani, Gustavo Márquez Sosa, Helmut Pichler, Hongbao Zhai, Horst Pohlmann, Ignacio Trejos, Ilia Kulakov, Ine Lutterman, Ingvar Nordström, Iosif Itkin, Jamie Mitchell, Jan Giesen, Jean-Francois Riverin, Joanna Kazun, Joanne Tremblay, Joëlle Genois, Johan Klintin, John Kurowski, Jörn Münzel, Judy McKay, Jürgen Beniermann, Karol Frühauf, Katalin Balla, Kevin Kooh, Klaudia Dussa-Zieger, Klaus Erlenbach, Klaus Olsen, Krisztián Miskó, Laura Albert, Liang Ren, Lijuan Wang, Lloyd Roden, Lucjan Stapp, Mahmoud Khalaili, Marek Majernik, Maria Clara Choucair, Mark Rutz, Markus Niehammer, Martin Klonk, Márton Siska, Matthew Gregg, Matthias Hamburg, Mattijs Kemmink, Maud Schlich, May Abu-Sbeit, Meile Posthuma, Mette Bruhn-Pedersen, Michal Tal, Michel Boies, Mike Smith, Miroslav Renda, Mohsen Ekssir, Monika Stocklein Olsen, Murian Song, Nicola De Rosa, Nikita Kalyani, Nishan Portoyan, Nitzan Goldenberg, Ole Chr. Hansen, Patricia McQuaid, Patricia Osorio, Paul Weymouth, Pawel Kwasik, Peter Zimmerer, Petr Neugebauer, Piet de Roo, Radoslaw Smilgin, Ralf Bongard, Ralf Reissing, Randall Rice, Rik Marselis, Rogier Ammerlaan, Sabine Gschwandtner, Sabine Uhde, Salinda Wickramasinghe, Salvatore Reale, Sammy Kolluru, Samuel Ouko, Stephanie Ulrich, Stuart Reid, Surabhi Bellani, Szilard Szell, Tamás Gergely, Tamás Horváth, Tatiana Sergeeva, Tauhida Parveen, Thaer Mustafa, Thomas Eisbrenner, Thomas Harms, Thomas Heller, Tobias Letzkus, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo and Zsolt Hargitai.

ISTQB Working Group Foundation Level (Edition 2018): Klaus Olsen (chair), Tauhida Parveen (vice chair), Rex Black (project manager), Eshraka Zakaria, Debra Friedenberg, Ebbe Munk, Hans Schaefer, Judy McKay, Marie Walsh, Meile Posthuma, Mike Smith, Radoslaw Smilgin, Stephanie Ulrich, Steve Toms, Corne Kruger, Dani Almog, Eric Riou du Cosquer, Igal Levi, Johan Klintin, Kenji Onishi, Rashed Karim, Stevan Zivanovic, Sunny Kwon, Thomas Müller, Vipul Kocher, Yaron Tsubery e a todos os Conselhos Membros por suas sugestões.

ISTQB Working Group Foundation Level (Edition 2011): Thomas Müller (presidente), Debra Friedenberg. A equipe A equipe principal agradece à 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 a todos os Conselhos Membros pelas sugestões para a versão atual do syllabus.

ISTQB Working Group Foundation Level (Edition 2010): Thomas Müller (presidente), Rahul Verma, Martin Klonk e Armin Beer. A equipe principal agradece à equipe de revisão (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 a todos os Conselhos Membros por suas sugestões.

ISTQB Working Group Foundation Level (Edition 2007): Thomas Müller (presidente), Dorothy Graham, Debra Friedenberg e Erik van Veenendaal. A equipe principal agradece à equipe de revisão (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson e Wonil Kwon) e a todos os Conselhos Membros por suas sugestões.

ISTQB Working Group Foundation Level (Edition 2005): Thomas Müller (presidente), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson e Erik van Veenendaal. A equipe principal agradece à equipe de revisão e a todos os Conselhos Membros pelas sugestões.

O BSTQB agradece aos voluntário do GT Traduções pelo empenho e esforço na tradução e revisão deste material: Eduardo Medeiros, George Fialkovitz, Irene Nagase, Osmar Higashi, Rogério Athaide, Stênio Viveiros.

# 0 Introdução

## 0.1 Objetivo deste syllabus

Esse syllabus forma a base para a *International Software Testing Qualification* no *Foundation Level*. O ISTQB® fornece esse syllabus da seguinte forma:

- Aos conselhos membros, para traduzir para seu idioma local e credenciar os provedores de treinamento. Os conselhos membros podem adaptar o syllabus às suas necessidades específicas de idioma e modificar as referências para adaptá-las às suas publicações locais.
- Aos órgãos de certificação, para que elaborem questões de exame em seu idioma local, adaptadas aos objetivos de aprendizagem deste programa.
- Aos provedores de treinamento, para produzir material didático e determinar métodos de ensino adequados.
- Para candidatos à certificação, para se preparar para o exame de certificação (como parte de um curso de treinamento ou de forma independente).

Para a comunidade internacional de Engenharia de Software e Sistemas, para promover a profissão de Teste de Software e Sistemas, e como base para livros e artigos.

## 0.2 O Certified Tester Foundation Level em Teste de Software

A certificação *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, Desenvolvedores de Software e membros da equipe de desenvolvimento. Essa certificação de nível fundamental também é apropriada para qualquer pessoa que queira ter uma compreensão básica dos testes de software, como Gerentes de Projeto, Gerentes de Qualidade, Product Owner, Gerentes de Desenvolvimento de Software, Analistas de Negócios, Diretores de TI e Consultores de Gestão. Os detentores do Certificado Foundation Level poderão obter certificações de nível superior em teste de software.

## 0.3 Carreira para Testadores

O esquema do ISTQB® oferece suporte para profissionais de teste em todos os estágios de suas carreiras, oferecendo amplitude e profundidade de conhecimento. As pessoas que obtiveram a certificação *ISTQB® Foundation Level* também podem se interessar pelos níveis *Core Advanced* (Analista de Testes, Analista de Testes Técnicos e Gerente de Testes) e, posteriormente, pelo *Expert Level* (Gerenciamento de Testes ou Melhoria do Processo de Testes). Qualquer pessoa que queira desenvolver habilidades em práticas de teste em um ambiente ágil pode considerar as certificações *Agile Technical Tester* ou *Agile Test Leadership at Scale*. O fluxo do *Specialist* oferece um mergulho profundo em áreas que têm abordagens e atividades de teste específicas (p. ex., em automação de testes, testes de AI, testes baseados em modelos, testes de aplicativos móveis), que estão relacionadas a áreas de teste específicas (p. ex., testes de performance, testes de usabilidade, testes de aceite, testes de segurança) ou que agrupam o conhecimento de testes para determinados

domínios (p. ex., automotivo ou de jogos). Acesse www.istqb.org para obter as informações mais recentes sobre o Esquema de Certificação em Teste do ISTQB®.

## 0.4 Resultados de Negócio

Esta seção lista os 14 Resultados de Negócio esperados de uma pessoa que tenha obtido a certificação *Foundation Level*.

Um Testador certificado no *Foundation Level* pode...

FL-BO1 Compreender o que é um teste e por que ele é benéfico;
FL-BO2 Compreender os conceitos fundamentais de teste de software;
FL-BO3 Identificar a abordagem e as atividades de teste a serem implementadas dependendo do contexto do teste;
FL-BO4 Avaliar e melhorar a qualidade da documentação;
FL-BO5 Aumentar a eficácia e a eficiência dos testes;
FL-BO6 Alinhar o processo de teste com o ciclo de vida de desenvolvimento de software;
FL-BO7 Compreender os princípios de gerenciamento de testes;
FL-BO8 Escreva e comunique relatórios de defeitos claros e compreensíveis;
FL-BO9 Compreender os fatores que influenciam as prioridades e os esforços relacionados aos testes;
FL-BO10 Trabalhar como parte de uma equipe multifuncional;
FL-BO11 Conhecer os riscos e benefícios relacionados à automação de testes;
FL-BO12 Identificar as habilidades essenciais necessárias para a realização de testes;
FL-BO13 Compreender o impacto do risco nos testes;
FL-BO14 Relatar com eficácia o progresso e a qualidade do teste.

## 0.5 Objetivos de Aprendizagem e Nível Cognitivo de Conhecimento

Os objetivos de aprendizagem examináveis (LO) apoiam os resultados de negócio e são usados para criar o exame *Certified Tester Foundation Level*. Em geral, todo o conteúdo dos capítulos de 1 a 6 deste syllabus pode ser examinado em um nível K1. Ou seja, o candidato pode ser solicitado a reconhecer, lembrar ou recordar uma palavra-chave ou um conceito mencionado em qualquer um dos seis capítulos. Os níveis específicos dos objetivos de aprendizagem são mostrados no início de cada capítulo e classificados da seguinte forma:

- **K1**: Lembrar
- **K2**: Compreender
- **K3**: Aplicar

Mais detalhes e exemplos de objetivos de aprendizagem são fornecidos no Apêndice A. Todos os termos listados como palavras-chave logo abaixo dos títulos dos capítulos devem ser lembrados (K1), mesmo que não sejam explicitamente mencionados nos Objetivos de aprendizagem.

## 0.6 Exame de certificação Foundation Level

O exame de certificação *Foundation Level* baseia-se neste syllabus. As respostas às questões do exame podem exigir o uso de material baseado em mais de uma seção deste syllabus. Todas as seções do syllabus são passíveis de exame, exceto a Introdução e os Apêndices. Normas e livros são incluídos como referências (Capítulo 7), mas seu conteúdo não é passível de exame, além do que está resumido no próprio syllabus a partir dessas normas e livros. Consulte o documento *Foundation Level Examination Structures and Rules*.

## 0.7 Credenciamento

Um Conselho Membro do ISTQB® pode credenciar provedores de treinamento cujo material do curso siga este syllabus. Os provedores de treinamento devem obter as diretrizes de credenciamento do Conselho Membro (BSTQB®) ou do órgão que realiza o credenciamento. Um curso credenciado é reconhecido como estando em conformidade com este syllabus e pode ter um exame ISTQB® como parte do curso. As diretrizes de credenciamento para este syllabus seguem as diretrizes gerais de credenciamento publicadas pelo *Processes Management and Compliance Working Group*.

## 0.8 Manuseio de Normas

Há normas referenciadas no *Foundation Syllabus* (p. ex., normas IEEE ou ISO). Essas referências fornecem uma estrutura (como nas referências à ISO25010 com relação às características de qualidade) ou fornecem uma fonte de informações adicionais, se o leitor desejar. Os documentos de normas não se destinam a exame. Consulte o capítulo 7 para obter mais informações sobre normas.

## 0.9 Mantendo-se atualizado

O setor de software muda rapidamente. Para lidar com essas mudanças e fornecer aos stakeholders acesso a informações relevantes e atuais, os grupos de trabalho do ISTQB criaram links no site www.istqb.org, que se referem à documentação de apoio e às mudanças nos padrões. Essas informações não são passíveis de exame no syllabus *Foundation Level*.

## 0.10 Nível de detalhe

O nível de detalhamento desse syllabus permite a realização de cursos e exames consistentes internacionalmente. Para atingir esse objetivo, o syllabus consiste em:

- Objetivos gerais de instrução que descrevem a intenção do *Foundation Level*;
- Uma lista de termos (palavras-chave) que os alunos devem ser capazes de lembrar;
- Objetivos de aprendizagem (LO) para cada área de conhecimento, descrevendo os resultados cognitivos de aprendizagem a serem alcançados;
- Uma descrição dos principais conceitos, incluindo referências a fontes reconhecidas.

O conteúdo do syllabus não é uma descrição de toda a área de conhecimento de testes de software; ele reflete o nível de detalhes a ser abordado nos cursos de treinamento de nível fundamental. Ele se concentra em conceitos e técnicas de teste que podem ser aplicados a todos os projetos de software, independentemente do SDLC empregado.

## 0.11 Como este syllabus está organizado

Há seis capítulos com conteúdo passível de exame. O título de nível superior de cada capítulo especifica o tempo de estudo para o capítulo. O tempo não é fornecido em subcapítulos. Para cursos de treinamento credenciados, o syllabus exige um mínimo de 1135 minutos (18 horas e 55 minutos) de instrução, distribuídos nos seis capítulos da seguinte forma:

**Capítulo 1**: Fundamentos dos Testes (180 minutos)

- O aluno aprende os princípios básicos relacionados a testes, os motivos pelos quais os testes são necessários e quais são os objetivos do teste;
- O aluno compreende o processo de teste, as principais atividades de teste e o testware;
- O aluno compreende as habilidades essenciais para a realização de testes.

**Capítulo 2**: Testes ao longo do ciclo de vida de desenvolvimento de software (130 minutos)

- O aluno aprende como os testes são incorporados a diferentes abordagens de desenvolvimento.
- O aluno aprende os conceitos de abordagens de teste primeiro, bem como de DevOps.
- O aluno aprende sobre os diferentes níveis de teste, tipos de teste e testes de manutenção.

**Capítulo 3**: Teste Estático (80 minutos)

- O aluno aprende sobre os fundamentos dos testes estáticos, o processo de feedback e revisão.

**Capítulo 4**: Análise e Projeto de Teste (390 minutos)

- O aluno aprende a aplicar técnicas de teste caixa-preta, caixa-branca e baseadas na experiência para derivar casos de teste de vários produtos de trabalho de software.
- O aluno aprende sobre a abordagem de teste baseada na colaboração.

**Capítulo 5**: Gerenciando as atividades de teste (335 minutos)

- O aluno aprende a planejar testes em geral e a estimar o esforço de teste.
- O aluno aprende como os riscos podem influenciar o escopo dos testes.
- O aluno aprende a monitorar e controlar as atividades de teste.
- O aluno aprende como o gerenciamento de configuração dá suporte aos testes.
- O aluno aprende a relatar defeitos de forma clara e compreensível.

**Capítulo 6**: Ferramentas de Teste (20 minutos)

- O aluno aprende a classificar as ferramentas e a entender os riscos e os benefícios da automação de testes.

# 1 Fundamentos de Teste (180 min)

## Palavras-chave

cobertura, depuração, defeito, erro, falha, qualidade, garantia de qualidade, causa raiz, análise de teste, base de teste, caso de teste, conclusão do teste, condição de teste, controle de teste, dados de teste, projeto de teste, execução de teste, implementação de teste, monitoramento de teste, objeto de teste, objetivo de teste, planejamento de teste, procedimento de teste, resultado de teste, teste, testware, validação, verificação

## Objetivos de aprendizagem

### 1.1 O que é teste?

FL-1.1.1 (K1) Identificar objetivos típicos de teste

FL-1.1.2 (K2) Diferenciar teste de depuração

### 1.2 Por que os testes são necessários?

FL-1.2.1 (K2) Exemplificar por que os testes são necessários

FL-1.2.2 (K1) Relembrar a relação entre testes e garantia de qualidade

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

### 1.3 Princípios de teste

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

### 1.4 Atividades de teste, Testware e Papéis de teste

FL-1.4.1 (K2) Resumir as diferentes atividades e tarefas de teste

FL-1.4.2 (K2) Explicar o impacto do contexto no processo de teste

FL-1.4.3 (K2) Diferenciar testware que dá suporte às atividades de teste

FL-1.4.4 (K2) Explicar o valor de manter a rastreabilidade

FL-1.4.5 (K2) Comparar os diferentes papéis no teste

### 1.5 Habilidades essenciais e boas práticas em testes

FL-1.5.1 (K2) Dar exemplos das habilidades genéricas necessárias para testar

FL-1.5.2 (K1) Relembrar as vantagens da abordagem de equipe completa

FL-1.5.3 (K2) Distinguir os benefícios e as desvantagens da independência dos testes

## 1.1 O que é teste?

Os sistemas de software são parte integrante de nossa vida cotidiana. A maioria das pessoas já teve experiência com software que não funcionou como esperado. Um software que não funciona corretamente pode causar muitos problemas, inclusive perda de dinheiro, tempo ou reputação comercial e, em casos extremos, até mesmo lesões ou morte. O teste de software avalia a qualidade do software e ajuda a reduzir o risco de falha do software em operação.

O teste de software é um conjunto de atividades para descobrir defeitos e avaliar a qualidade dos artefatos de software. Esses artefatos, quando estão sendo testados, são conhecidos como objetos de teste. Uma equívoco comum sobre testes é que eles consistem apenas na execução de testes (ou seja, executar o software e verificar os resultados dos testes). Entretanto, o teste de software também inclui outras atividades e deve estar alinhado com o ciclo de vida de desenvolvimento de software - SDLC (ver capítulo 2).

Outro equívoco comum sobre os testes é que eles se concentram inteiramente na verificação do objeto de teste. Embora o teste envolva verificação, ou seja, verificar se o sistema atende aos requisitos especificados, ele também envolve validação, o que significa checar se o sistema atende às necessidades dos usuários e de outros stakeholders em seu ambiente operacional.

Os testes podem ser dinâmicos ou estáticos. O teste dinâmico envolve a execução do software, enquanto o teste estático não. O teste estático inclui revisões (ver capítulo 3) e análise estática. Os testes dinâmicos usam diferentes tipos de técnicas de teste e abordagens de teste para derivar casos de teste (ver capítulo 4).

O teste não é apenas uma atividade técnica. Ele também precisa ser adequadamente planejado, gerenciado, estimado, monitorado e controlado (ver capítulo 5).

Os Testadores usam ferramentas (ver capítulo 6), mas é importante lembrar que o teste é, em grande parte, uma atividade intelectual, exigindo que os Testadores tenham conhecimento especializado, usem habilidades analíticas e apliquem o pensamento crítico e o pensamento sistêmico (Myers 2011, Roman 2018).

A norma ISO/IEC/IEEE 29119-1 fornece mais informações sobre os conceitos de teste de software.

### 1.1.1 Objetivos do teste

Os objetivos típicos do teste são:

- Avaliar produtos de trabalho, como requisitos, histórias de usuários, projetos e código;
- Detectar falhas e defeitos;
- Garantir a cobertura necessária de um objeto de teste;
- Reduzindo o nível de risco de qualidade de software inadequado;
- Verificar se os requisitos especificados foram atendidos;
- Verificar se um objeto de teste está em conformidade com os requisitos contratuais, legais e normativos;
- Fornecer informações aos stakeholders para que eles possam tomar decisões informadas;
- Criar confiança na qualidade do objeto de teste;

- Validar se o objeto de teste está completo e funciona conforme o esperado pelos stakeholders.

Os objetivos dos testes podem variar, dependendo do contexto, o que inclui o produto de trabalho que está sendo testado, o nível de teste, os riscos, o ciclo de vida de desenvolvimento de software (SDLC) que está sendo seguido e os fatores relacionados ao contexto do negócio, por exemplo, estrutura corporativa, considerações competitivas ou tempo de comercialização.

### 1.1.2 Teste e depuração

O teste e a depuração são atividades distintas. O teste pode desencadear falhas causadas por defeitos no software (teste dinâmico) ou pode encontrar defeitos diretamente no objeto de teste (teste estático).

Quando o teste dinâmico (ver capítulo 4) aciona uma falha, a depuração se preocupa em encontrar as causas dessa falha (defeitos), analisar essas causas e eliminá-las. O processo típico de depuração nesse caso envolve:

- Reproduzir uma falha
- Diagnosticar (encontrar a causa principal)
- Corrigir a causa

O teste de confirmação subsequente verifica se as correções resolveram o problema. De preferência, o teste de confirmação é feito pela mesma pessoa que realizou o teste inicial. Os testes de regressão subsequentes também podem ser realizados para verificar se as correções estão causando falhas em outras partes do objeto de teste (ver seção 2.2.3 para obter mais informações sobre testes de confirmação e testes de regressão).

Quando o teste estático identifica um defeito, a depuração se preocupa em removê-lo. Não há necessidade de reprodução ou diagnóstico, pois o teste estático encontra defeitos diretamente e não pode causar falhas (ver capítulo 3).

## 1.2 Por que os testes são necessários?

Os testes, como uma forma de controle de qualidade, ajudam a atingir os objetivos acordados dentro do escopo, do tempo, da qualidade e das restrições orçamentárias estabelecidas. A contribuição dos testes para o sucesso não deve se restringir às atividades da equipe de testes. Qualquer stakeholder pode usar suas habilidades de teste para trazer o projeto mais próximo do sucesso. O teste de componentes, sistemas e documentação associada ajuda a identificar defeitos no software,

### 1.2.1 Contribuições para o sucesso dos testes

O teste oferece um meio econômico de detectar defeitos. Esses defeitos podem então serem removidos (por depuração: uma atividade que não é de teste), de modo que o teste contribui indiretamente para objetos de teste de maior qualidade.

O teste fornece um meio de avaliar diretamente a qualidade de um objeto de teste em vários estágios do SDLC. Essas medidas são usadas como parte de uma atividade maior de gerenciamento de

projetos, contribuindo para as decisões de passar para o próximo estágio do SDLC, como a decisão de liberação.

Os testes oferecem aos usuários uma representação indireta no projeto de desenvolvimento. Os Testadores garantem que seu entendimento das necessidades dos usuários seja considerado em todo o ciclo de vida do desenvolvimento. A alternativa é envolver um conjunto representativo de usuários como parte do projeto de desenvolvimento, o que geralmente não é possível devido aos altos custos e à falta de disponibilidade de usuários adequados.

Os testes também podem ser necessários para atender a requisitos contratuais ou legais, ou para cumprir normas regulatórias.

### 1.2.2 Testes e Garantia da Qualidade (QA)

Embora as pessoas geralmente usem os termos "teste" e "garantia da qualidade" (QA) de forma intercambiável, teste e QA não são a mesma coisa. O teste é uma forma de controle de qualidade (QC).

O QC é uma abordagem corretiva e orientada para o produto que se concentra nas atividades que apoiam a obtenção de níveis adequados de qualidade. Os testes são uma das principais formas de controle de qualidade, enquanto outras incluem métodos formais (verificação de modelos e prova de exatidão), simulação e prototipagem.

A QA é uma abordagem preventiva e orientada para o processo que se concentra na implementação e no aprimoramento dos processos. Ela funciona com base no fato de que, se um bom processo for seguido corretamente, ele gerará um bom produto. A QA se aplica aos processos de desenvolvimento e teste e é responsabilidade de todos em um projeto.

Os resultados dos testes são usados por QA e QC. No QC, eles são usados para corrigir defeitos, enquanto no QA eles fornecem feedback sobre a performance dos processos de desenvolvimento e teste.

### 1.2.3 Erros, Defeitos, Falhas e Causas-raiz

Os seres humanos cometem erros (equívocos), que produzem defeitos (falta, bugs) que, por sua vez, podem resultar em falhas. Os seres humanos cometem erros por vários motivos, como pressão de tempo, complexidade dos produtos de trabalho, processos, infraestrutura ou interações, ou simplesmente porque estão cansados ou não têm treinamento adequado.

Os defeitos podem ser encontrados na documentação, como uma especificação de requisitos ou um script de teste, no código-fonte ou em um artefato de suporte, como um arquivo de compilação. Os defeitos em artefatos produzidos no início do SDLC, se não forem detectados, geralmente levam a artefatos defeituosos mais tarde no ciclo de vida. Se um defeito no código for executado, o sistema poderá deixar de fazer o que deveria fazer ou fazer algo que não deveria, causando uma falha. Alguns defeitos sempre resultarão em falha se forem executados, enquanto outros só resultarão em falhas em circunstâncias específicas, e alguns podem nunca resultar em falha.

Os erros e defeitos não são a única causa de falhas. As falhas também podem ser causadas por condições ambientais, como quando a radiação ou o campo eletromagnético causam defeitos no firmware.

Uma causa-raiz é um motivo fundamental para a ocorrência de um problema (p. ex., uma situação que leva a um erro). As causas-raiz são identificadas por meio da análise de causa-raiz, que normalmente é realizada quando ocorre uma falha ou quando um defeito é identificado. Acredita-se que outras falhas ou defeitos semelhantes possam ser evitados ou que sua frequência possa ser reduzida com a abordagem da causa-raiz, como, por exemplo, sua remoção.

## 1.3 Princípios de Teste

Ao longo dos anos, foram sugeridos vários princípios de teste que oferecem diretrizes gerais aplicáveis a todos os testes. Este syllabus descreve sete desses princípios.

1) **O teste mostra a presença, não a ausência de defeitos**. Os testes podem mostrar que os defeitos estão presentes no objeto de teste, mas não podem provar que não há defeitos (Buxton 1970). Os testes reduzem a probabilidade de defeitos não serem descobertos no objeto de teste, mas, mesmo que nenhum defeito seja encontrado, os testes não podem provar a corretude do objeto de teste.
2) **Testes exaustivos são impossíveis**. Testar tudo não é viável, exceto em casos triviais (Manna 1978). Em vez de tentar testar exaustivamente, as técnicas de teste (ver capítulo 4), a priorização de casos de teste (ver seção 5.1.5) e os testes baseados em riscos (ver seção 5.2) devem ser usados para concentrar os esforços de teste**.**
3) **Testes antecipados economizam tempo e dinheiro**. Os defeitos que são removidos no início do processo não causarão defeitos subsequentes nos produtos de trabalho derivados. O custo da qualidade será reduzido, pois menos falhas ocorrerão posteriormente no SDLC (Boehm, 1981). Para encontrar defeitos logo no início, os testes estáticos (ver capítulo 3) e os testes dinâmicos (ver capítulo 4) devem ser iniciados o mais cedo possível.
4) **Os defeitos se agrupam**. Um pequeno número de componentes do sistema geralmente contém a maioria dos defeitos descobertos ou é responsável pela maioria das falhas operacionais (Enders 1975). Esse fenômeno é uma ilustração do Princípio de Pareto. Os agrupamentos de defeitos previstos e os agrupamentos de defeitos reais observados durante o teste ou em operação são uma entrada importante para o teste baseado em risco (ver seção 5.2).
5) **Os testes se degradam**. Se os mesmos testes forem repetidos muitas vezes, eles se tornarão cada vez mais ineficazes na detecção de novos defeitos (Beizer 1990). Para superar esse efeito, talvez seja necessário modificar os testes e os dados de teste existentes, e talvez seja necessário escrever novos testes. Entretanto, em alguns casos, a repetição dos mesmos testes pode ter um resultado benéfico, por exemplo, em testes de regressão automatizados (ver seção 2.2.3)**.**
6) **Os testes dependem do contexto**. Não existe uma única abordagem universalmente aplicável aos testes. Os testes são feitos de forma diferente em contextos diferentes (Kaner 2011).

7) **Falácia da ausência de defeitos**. É uma falácia (ou seja, uma concepção errônea) esperar que a verificação do software garanta o sucesso de um sistema. Testar exaustivamente todos os requisitos especificados e corrigir todos os defeitos encontrados ainda pode produzir um sistema que não atenda às necessidades e expectativas dos usuários, que não ajude a atingir os objetivos de negócio do cliente e que seja inferior a outros sistemas concorrentes. Além da verificação, a validação também deve ser realizada (Boehm, 1981).

## 1.4 Atividades de teste, Testware e Papéis no teste

O teste depende do contexto, mas, em um nível elevado, há conjuntos comuns de atividades de teste sem os quais é menos provável que o teste atinja seus objetivos. Esses conjuntos de atividades de teste formam um processo de teste. O processo de teste pode ser adaptado a uma determinada situação com base em vários fatores. Quais atividades de teste estão incluídas nesse processo de teste, como são implementadas e quando ocorrem normalmente são decididas como parte do planejamento do teste para a situação específica (ver seção 5.1).

As seções a seguir descrevem os aspectos gerais desse processo de teste em termos de atividades e tarefas de teste, o impacto do contexto, o material de teste, a rastreabilidade entre a base e o material de teste e as funções de teste.

A norma ISO/IEC/IEEE 29119-2 fornece mais informações sobre os processos de teste.

### 1.4.1 Atividades e Tarefas de Teste

Um processo de teste geralmente consiste nos principais grupos de atividades descritos abaixo. Embora muitas dessas atividades possam parecer seguir uma sequência lógica, elas geralmente são implementadas de forma iterativa ou paralela. Essas atividades de teste geralmente precisam ser adaptadas ao sistema e ao projeto.

O **Planejamento do Teste** consiste em definir os objetivos do teste e, em seguida, selecionar uma abordagem que melhor atinja os objetivos dentro das restrições impostas pelo contexto geral. O planejamento de testes é explicado com mais detalhes na seção 5.1.

O **Monitoramento e Controle de Teste.** O monitoramento de teste envolve a verificação contínua de todas as atividades de teste e a comparação do progresso real com o plano. O controle do teste envolve a tomada das ações necessárias para atingir os objetivos do teste. O monitoramento e o controle dos testes são explicados com mais detalhes na seção 5.3**.**

A **Análise de Teste** inclui a análise da base de teste para identificar os recursos testáveis e definir e priorizar as condições de teste associadas, juntamente com os riscos e níveis de risco relacionados (ver seção 5.2). A base de teste e os objetos de teste também são avaliados para identificar defeitos que possam conter e para avaliar sua testabilidade. A análise do teste geralmente é apoiada pelo uso de técnicas de teste (ver capítulo 4). A análise de teste responde à pergunta "o que testar?" em termos de critérios de cobertura mensuráveis.

O **Modelagem de Teste** inclui a elaboração das condições de teste em casos de teste e outros materiais (p. ex., cartas de teste). Essa atividade geralmente envolve a identificação de itens de cobertura, que servem como guia para especificar as entradas do caso de teste. As técnicas de teste

(ver capítulo 4) podem ser usadas para apoiar essa atividade. A modelagem de teste também inclui a definição dos requisitos de dados de teste, o projeto do ambiente de teste e a identificação de qualquer outra infraestrutura e ferramenta necessária. A modelagem de teste responde à pergunta "como testar?".

A **Implementação do Teste** inclui a criação ou a aquisição do material de teste necessário para a execução (p. ex., dados de teste). Os casos de teste podem ser organizados em procedimentos de teste e geralmente são reunidos em conjuntos de testes. São criados scripts de teste manuais e automatizados. Os procedimentos de teste são priorizados e organizados em um cronograma de execução de teste para uma execução eficiente do teste (ver seção 5.1.5). O ambiente de teste é criado e verificado quanto à configuração correta.

A **Execução do Teste** inclui a execução dos testes de acordo com o cronograma de execução ("rodar" o teste). A execução do teste pode ser manual ou automatizada. A execução de testes pode assumir várias formas, inclusive testes contínuos ou sessões de testes em pares. Os resultados reais dos testes são comparados com os resultados esperados. Os resultados do teste são registrados. As anomalias são analisadas para identificar suas causas prováveis. Essa análise nos permite relatar as anomalias com base nas falhas observadas (ver seção 5.5).

As atividades de **Conclusão de Teste** geralmente ocorrem nos marcos do projeto (p. ex., lançamento, fim da iteração, conclusão do nível de teste) para quaisquer defeitos não resolvidos, solicitações de alteração ou itens do backlog do produto criados. Qualquer material de teste que possa ser útil no futuro é identificado e arquivado ou entregue às equipes apropriadas. O ambiente de teste é encerrado em um estado acordado. As atividades de teste são analisadas para identificar as lições aprendidas e as melhorias para iterações, versões ou projetos futuros (ver seção 2.1.6). Um relatório de conclusão do teste é criado e comunicado aos stakeholders**.**

### 1.4.2 Processo de Teste no Contexto

Os testes não são realizados de forma isolada. As atividades de teste são parte integrante dos processos de desenvolvimento executados em uma organização. Os testes são financiados pelos stakeholders e seu objetivo final é ajudar a atender às necessidades de negócio deles. Portanto, a forma como o teste é realizado dependerá de vários fatores contextuais, incluindo:

- Stakeholders (necessidades, expectativas, requisitos, disposição para cooperar etc.).
- Membros da equipe (habilidades, conhecimento, nível de experiência, disponibilidade, necessidades de treinamento etc.).
- Domínio do negócio (criticidade do objeto de teste, riscos identificados, necessidades do mercado, normas legais específicas etc.).
- Fatores técnicos (tipo de software, arquitetura do produto, tecnologia usada etc.).
- Restrições do projeto (escopo, tempo, orçamento, recursos etc.).
- Fatores organizacionais (estrutura organizacional, políticas existentes, práticas utilizadas etc.).
- Ciclo de vida do desenvolvimento de software (práticas de engenharia, métodos de desenvolvimento etc.).
- Ferramentas (disponibilidade, usabilidade, conformidade etc.).

Esses fatores terão um impacto em muitas questões relacionadas a testes, incluindo: estratégia de teste, técnicas de teste usadas, grau de automação de teste, nível de cobertura necessária, nível de detalhe da documentação de teste, relatórios etc.

### 1.4.3 Testware

O testware é criado como produto de trabalho de saída das atividades de teste descritas na seção 1.4.1. Há uma variação significativa na forma como as diferentes organizações produzem, moldam, nomeiam, organizam e gerenciam seus produtos de trabalho. O gerenciamento adequado da configuração (ver seção 5.4) garante a consistência e a integridade dos produtos de trabalho. A lista de produtos de trabalho a seguir não é completa:

- Os **produtos de trabalho** do **planejamento de testes** incluem: plano de testes, cronograma de testes, registro de riscos e critérios de entrada e saída (ver seção 5.1). O registro de riscos é uma lista de riscos juntamente com a probabilidade de risco, o impacto do risco e informações sobre a mitigação do risco (ver seção 5.2). O cronograma de testes, o registro de riscos e os critérios de entrada e saída geralmente fazem parte do plano de testes.
- Os **produtos de trabalho** do **monitoramento e controle de testes** incluem: relatórios de progresso de testes (ver seção 5.3.2), documentação de diretrizes de controle (ver seção 5.3) e informações sobre riscos (ver seção 5.2).
- Os **produtos de trabalho** da **análise de teste** incluem: Condições de teste (priorizadas) (p. ex., critérios de aceite, ver seção 4.5.2) e relatórios de defeitos referentes a defeitos na base de teste (se não forem corrigidos diretamente).
- Os **produtos de trabalho** da **modelagem de teste** incluem: Casos de teste (priorizados), cartas de teste, itens de cobertura, requisitos de dados de teste e requisitos de ambiente de teste.
- Os **produtos de trabalho** da **implementação de teste** incluem: procedimentos de teste, scripts de teste automatizados, conjuntos de teste, dados de teste, cronograma de execução de teste e elementos do ambiente de teste. Exemplos de elementos do ambiente de teste incluem: *stubs*, *drivers*, simuladores e virtualizações de serviço.
- Os **produtos de trabalho** da **execução de testes** incluem: registros de testes e relatórios de defeitos (ver seção 5.5).
- Os **produtos de trabalho** da **conclusão do teste** incluem: relatório de conclusão do teste (ver seção 5.3.2), itens de ação para melhoria de projetos ou iterações subsequentes, lições aprendidas documentadas e solicitações de alteração (p. ex., como itens do backlog do produto).

### 1.4.4 Rastreabilidade entre a Base de Teste e o Testware

Para implementar o monitoramento e o controle eficazes dos testes, é importante estabelecer e manter a rastreabilidade em todo o processo de teste entre os elementos da base de teste, o testware associado a esses elementos (p. ex., condições de teste, riscos, casos de teste), os resultados dos testes e os defeitos detectados.

A rastreabilidade precisa dá suporte à avaliação da cobertura, portanto, é muito útil que os critérios de cobertura mensuráveis sejam definidos na base do teste. Os critérios de cobertura podem

funcionar como indicadores-chave de performance para conduzir as atividades que mostram até que ponto os objetivos do teste foram alcançados (ver seção 1.1.1). Por exemplo:

- A rastreabilidade dos casos de teste aos requisitos pode verificar se os requisitos são cobertos pelos casos de teste.
- A rastreabilidade dos resultados dos testes aos riscos pode ser usada para avaliar o nível de risco residual em um objeto de teste.

Além de avaliar a cobertura, uma boa rastreabilidade permite determinar o impacto das mudanças, facilita as auditorias de teste e ajuda a atender os critérios de governança de TI. A boa rastreabilidade também torna o progresso do teste e os relatórios de conclusão mais compreensíveis, incluindo o status dos elementos da base de teste. Isso também pode ajudar a comunicar os aspectos técnicos dos testes aos stakeholders de uma maneira compreensível. A rastreabilidade fornece informações para avaliar a qualidade do produto, a capacidade do processo e o progresso do projeto em relação aos objetivos de negócio.

### 1.4.5 Papéis no Teste

Neste syllabus, são abordadas duas funções principais nos testes: um papel de gerenciamento de testes e um papel de testador. As atividades e tarefas atribuídas a esses dois papéis dependem de fatores como o contexto do projeto e do produto, as habilidades das pessoas que ocupam esses papéis e a organização.

O papel de gerenciamento de teste assume a responsabilidade geral pelo processo de teste, pela equipe de teste e pela liderança das atividades de teste. Ela concentra-se principalmente nas atividades de planejamento, monitoramento e controle, e conclusão de testes. A maneira pela qual o papel de gerenciamento de testes é realizado varia de acordo com o contexto. Por exemplo, no desenvolvimento ágil de software, algumas das tarefas de gerenciamento de testes podem ser realizadas pela equipe ágil. As tarefas que abrangem várias equipes ou toda a organização podem ser executadas por gerentes de teste fora da equipe de desenvolvimento.

O papel de testador assume a responsabilidade geral pelo aspecto de engenharia (técnico) do teste. Ela concentra-se principalmente nas atividades de análise, modelagem, implementação e execução de teste.

Pessoas diferentes podem assumir esses papéis em momentos diferentes. Por exemplo, o papel de gerenciamento de testes pode ser desempenhada por um Líder de Equipe, por um Gerente de Teste, por um Gerente de Desenvolvimento etc. Também é possível que uma pessoa assuma o papel de testador e gerenciamento de teste ao mesmo tempo.

## 1.5 Habilidades essenciais e boas práticas em testes

Habilidade é a capacidade de fazer algo bem-feito que vem do conhecimento, da prática e da aptidão de uma pessoa. Os bons Testadores devem ter algumas habilidades essenciais para fazer bem o seu trabalho. Os bons Testadores devem ser participantes eficazes de uma equipe e devem ser capaz de realizar testes em diferentes níveis de independência de teste.

### 1.5.1 Habilidades genéricas necessárias para testes

Embora sejam genéricas, as habilidades a seguir são particularmente relevantes para os Testadores:

- Conhecimento sobre testes (para aumentar a eficácia dos testes, por exemplo, usando técnicas de teste)
- Meticulosidade, cuidado, curiosidade, atenção aos detalhes, ser metódico (para identificar defeitos, especialmente aqueles que são difíceis de encontrar)
- Boas habilidades de comunicação, ser um bom ouvinte, e trabalhar em equipe (para interagir efetivamente com todos os stakeholders, transmitir informações a outras pessoas, ser compreendido, e relatar e discutir defeitos)
- Pensamento analítico, pensamento crítico, criatividade (para aumentar a eficácia dos testes)
- Conhecimento técnico (para aumentar a eficiência dos testes, por exemplo, usando ferramentas de teste adequadas)
- Conhecimento do domínio (para poder entender e se comunicar com usuários finais/representantes de negócio)

Os Testadores geralmente são os portadores de más notícias. É uma característica humana comum culpar o portador pelas más notícias. Isso torna as habilidades de comunicação cruciais para os Testadores. A comunicação dos resultados dos testes pode ser percebida como uma crítica ao produto e ao seu autor. O viés de confirmação pode dificultar o aceite de informações que discordem das crenças atuais. Algumas pessoas podem ver o teste como uma atividade destrutiva, embora ele contribua muito para o sucesso do projeto e a qualidade do produto. Para tentar melhorar essa visão, as informações sobre defeitos e falhas devem ser comunicadas de forma construtiva.

### 1.5.2 Abordagem de equipe completa

Uma das habilidades importantes para um Testador é a capacidade de trabalhar de forma eficaz em um contexto de equipe e de contribuir positivamente para as metas da equipe. A abordagem de equipe completa - uma prática proveniente da *Extreme Programming* (ver seção 2.1) - baseia-se nessa habilidade.

Na abordagem de equipe completa, qualquer membro da equipe com o conhecimento e as habilidades necessárias pode executar qualquer tarefa, e todos são responsáveis pela qualidade. Os membros da equipe compartilham o mesmo espaço de trabalho (físico ou virtual), pois a co-localização facilita a comunicação e a interação. A abordagem de equipe completa melhora a dinâmica da equipe, aprimora a comunicação e a colaboração dentro da equipe e cria sinergia ao permitir que os vários conjuntos de habilidades da equipe sejam aproveitados para o benefício do projeto.

Os Testadores trabalham em estreita colaboração com outros membros da equipe para garantir que os níveis de qualidade desejados sejam alcançados. Isso inclui colaborar com representantes do negócio para ajudá-los a criar testes de aceite adequados e trabalhar com desenvolvedores para chegar a um acordo sobre a estratégia de teste e decidir sobre abordagens de automação de teste. Assim, os Testadores podem transferir conhecimento sobre testes para outros membros da equipe e influenciar o desenvolvimento do produto.

Dependendo do contexto, a abordagem de equipe completa nem sempre pode ser adequada. Por exemplo, em algumas situações, como as críticas para a segurança, pode ser necessário um alto nível de independência dos testes.

### 1.5.3 Independência dos testes

Um certo grau de independência torna o Testador mais eficaz na localização de defeitos devido às diferenças entre os vieses cognitivos do autor e do Testador (cf. Salman 1995). No entanto, a independência não substitui a familiaridade, por exemplo, os desenvolvedores podem encontrar com eficiência muitos defeitos em seu próprio código.

Os produtos de trabalho podem ser testados por seu autor (sem independência), pelos colegas do autor da mesma equipe (alguma independência), por Testadores de fora da equipe do autor, mas dentro da organização (alta independência) ou por Testadores de fora da organização (independência muito alta). Na maioria dos projetos, geralmente é melhor realizar testes com vários níveis de independência (p. ex., desenvolvedores realizando testes de componentes e de integração de componentes, equipe de testes realizando testes de sistemas e de integração de sistemas e representantes do negócio realizando testes de aceite).

O principal benefício da independência dos testes é que os Testadores independentes provavelmente reconhecerão diferentes tipos de falhas e defeitos em comparação com os desenvolvedores, devido às suas diferentes formações, perspectivas técnicas e preconceitos. Além disso, um Testador independente pode verificar, contestar ou refutar as suposições feitas pelos stakeholders durante a especificação e a implementação do sistema.

Entretanto, há também algumas desvantagens. Os Testadores independentes podem ficar isolados da equipe de desenvolvimento, o que pode levar à falta de colaboração, a problemas de comunicação ou a uma relação adversa com a equipe de desenvolvimento. Os desenvolvedores podem perder o senso de responsabilidade pela qualidade. Os Testadores independentes podem ser vistos como um gargalo ou ser responsabilizados por atrasos no lançamento.

# 2 Testes ao longo do Ciclo de Vida de Desenvolvimento de Software (130 min)

## Palavras-chave

teste de aceite, teste caixa-preta, teste de integração de componentes, teste de componentes, teste de confirmação, teste funcional, teste de integração, teste de manutenção, teste não funcional, teste de regressão, shift-left, teste de integração de sistemas, teste de sistemas, nível de teste, objeto de teste, tipo de teste, teste caixa-branca

## Objetivos de aprendizagem

### 2.1 Testes no contexto de um Ciclo de Vida de Desenvolvimento de Software

FL-2.1.1 (K2) Explicar o impacto do ciclo de vida de desenvolvimento de software escolhido nos testes.

FL-2.1.2 (K1) Relembrar as boas práticas de teste que se aplicam a todos os ciclos de vida de desenvolvimento de software.

FL-2.1.3 (K1) Relembrar os exemplos de abordagens de desenvolvimento que priorizam o teste.

FL-2.1.4 (K2) Resumir como o DevOps pode ter um impacto nos testes.

FL-2.1.5 (K2) Explicar a abordagem shift-left.

FL-2.1.6 (K2) Explicar como as retrospectivas podem ser usadas como um mecanismo de melhoria de processos.

### 2.2 Níveis de Teste e Tipos de Teste

FL-2.2.1 (K2) Distinguir os diferentes níveis de teste.

FL-2.2.2 (K2) Distinguir os diferentes tipos de teste.

FL-2.2.3 (K2) Distinguir o teste de confirmação do teste de regressão.

### 2.3 Teste de Manutenção

FL-2.3.1 (K2) Resumir os testes de manutenção e seus acionadores.

## 2.1 Testes no contexto de um Ciclo de Vida de Desenvolvimento de Software

Um modelo de ciclo de vida de desenvolvimento de software (SDLC) é uma representação abstrata e de alto nível do processo de desenvolvimento de software. Um modelo SDLC define como as diferentes fases de desenvolvimento e os tipos de atividades realizadas nesse processo se relacionam entre si, tanto lógica quanto cronologicamente. Exemplos de modelos de SDLC incluem: modelos de desenvolvimento sequencial (p. ex., modelo em cascata, modelo em V), modelos de desenvolvimento iterativo (p. ex., modelo em espiral, prototipagem) e modelos de desenvolvimento incremental (p. ex., Processo Unificado).

Algumas atividades nos processos de desenvolvimento de software também podem ser descritas por métodos de desenvolvimento de software mais detalhados e práticas ágeis. Os exemplos incluem: desenvolvimento orientado por testes de aceite (ATDD), desenvolvimento orientado pelo comportamento (BDD), *domain-driven design* (DDD), *extreme programming* (XP), *feature-driven development* (FDD), Kanban, Lean IT, Scrum e desenvolvimento orientado por teste (TDD).

### 2.1.1 Impacto do Ciclo de Vida de Desenvolvimento de Software nos Testes

Para ser bem-sucedido, o teste deve ser adaptado ao SDLC. A escolha do SDLC tem impacto sobre:

- O escopo e cronograma das atividades de teste (p. ex., níveis de teste e tipos de teste);
- O nível de detalhamento da documentação de teste;
- A escolha das técnicas de teste e da abordagem de teste;
- A extensão da automação de testes;
- O papel e responsabilidades de um Testador;

Nos modelos de desenvolvimento sequencial, nas fases iniciais, os Testadores normalmente participam das revisões de requisitos, da análise de testes e do projeto de testes. O código executável geralmente é criado nas fases posteriores, portanto, os testes dinâmicos não podem ser realizados no início do SDLC.

Em alguns modelos de desenvolvimento iterativos e incrementais, supõe-se que cada iteração entregue um protótipo funcional ou um incremento de produto. Isso implica que, em cada iteração, os testes estáticos e dinâmicos podem ser realizados em todos os níveis de teste. A entrega frequente de incrementos exige feedback rápido e testes de regressão extensivos.

O Desenvolvimento Ágil de Software pressupõe que podem ocorrer mudanças ao longo do projeto. Portanto, a documentação leve do produto de trabalho e a ampla automação de testes para facilitar os testes de regressão são favorecidas em projetos Ágeis. Além disso, a maior parte dos testes manuais tende a ser feita usando técnicas de teste baseadas na experiência (ver seção 4.4) que não exigem análise e projeto de teste prévio extensivo.

### 2.1.2 Ciclo de Vida de Desenvolvimento de Software e boas práticas de Teste

As boas práticas de teste, independentemente do modelo de SDLC escolhido, incluem:

- Para cada atividade de desenvolvimento de software, há uma atividade de teste correspondente, de modo que todas as atividades de desenvolvimento estejam sujeitas ao controle de qualidade;
- Diferentes níveis de teste (ver seção 2.2.1) têm objetivos de teste específicos e diferentes, o que permite que os testes sejam adequadamente abrangentes, evitando redundância;
- A análise e a modelagem do teste para um determinado nível de teste começam durante a fase de desenvolvimento correspondente do SDLC, para que o teste possa aderir ao princípio do teste antecipado (ver seção 1.3);
- Os Testadores estão envolvidos na revisão dos produtos de trabalho assim que os rascunhos dessa documentação estiverem disponíveis, de modo que esse teste antecipado e a detecção de defeitos possam apoiar a estratégia shift-left (ver seção 2.1.5).

### 2.1.3 Teste como um motivador para o desenvolvimento de software

O TDD, ATDD e BDD são abordagens de desenvolvimento semelhantes, em que os testes são definidos como um meio de direcionar o desenvolvimento. Cada uma dessas abordagens implementa o princípio do teste antecipado (ver seção 1.3) e segue uma abordagem shift-left (ver seção 2.1.5), pois os testes são definidos antes de o código ser escrito. Elas dão suporte a um modelo de desenvolvimento iterativo. Essas abordagens são caracterizadas da seguinte forma:

Desenvolvimento Orientado por Testes (TDD):

- Direciona a codificação por meio de casos de teste (em vez de um projeto de software extenso) (Beck 2003);
- Os testes são escritos primeiro, depois o código é escrito para satisfazer os testes e, em seguida, os testes e o código são refatorados;

Desenvolvimento Orientado por Teste de Aceite (ATDD) (ver seção 4.5.3):

- Deriva testes de critérios de aceite como parte do processo de desenho do sistema (Gärtner 2011);
- Os testes são escritos antes que a parte do aplicativo relacionada seja desenvolvida para atender aos testes.

Desenvolvimento Orientado pelo Comportamento (BDD):

- Expressa o comportamento desejado de um aplicativo com casos de teste escritos em uma forma simples de linguagem natural, que é fácil de entender pelos stakeholders - geralmente usando o formato Dado/Quando/Então. (Chelimsky 2010);
- Os casos de teste são então traduzidos automaticamente em testes executáveis.

Para todas as abordagens acima, os testes podem persistir como testes automatizados para garantir a qualidade do código em futuras adaptações/refatoração.

### 2.1.4 DevOps e Testes

DevOps é uma abordagem organizacional que visa a criar sinergia, fazendo com que o desenvolvimento (incluindo os testes) e as operações trabalhem juntos para atingir um conjunto de objetivos comuns. O DevOps exige uma mudança cultural em uma organização para preencher as

lacunas entre o desenvolvimento (incluindo testes) e as operações, tratando suas funções com o mesmo valor. O DevOps promove a autonomia da equipe, feedback rápido, cadeias de ferramentas integradas e práticas técnicas como Integração Contínua (CI *Continuous Integration*) e Entrega Contínua (CD *Continuous Delivery*). Isso permite que as equipes criem, testem e liberem códigos de alta qualidade mais rapidamente por meio de um pipeline de entrega de DevOps (Kim 2016).

Do ponto de vista dos testes, alguns dos benefícios do DevOps são:

- Feedback rápido sobre a qualidade do código e se as alterações afetam negativamente o código existente;
- O CI promove uma abordagem shift-left nos testes (ver seção 2.1.5), incentivando os desenvolvedores a enviar códigos de alta qualidade acompanhados de testes de componentes e análise estática;
- Promove processos automatizados, como CI/CD, que facilitam o estabelecimento de ambientes de teste estáveis;
- Aumenta a visão das características de qualidade não funcionais (p. ex., performance, confiabilidade);
- A automação por meio de um pipeline de entrega reduz a necessidade de testes manuais repetitivos;
- O risco na regressão é minimizado devido à escala e ao alcance dos testes de regressão automatizados;

O DevOps tem seus riscos e desafios, que incluem:

- O pipeline de entrega de DevOps deve ser definido e estabelecido;
- As ferramentas de CI/CD devem ser introduzidas e mantidas;
- A automação de testes requer recursos adicionais e pode ser difícil de estabelecer e manter.

Embora o DevOps venha com um alto nível de testes automatizados, os testes manuais - especialmente da perspectiva do usuário - ainda serão necessários.

### 2.1.5 Abordagem Shift-Left

O princípio do teste antecipado (ver seção 1.3) às vezes é chamado de shift-left porque é uma abordagem em que o teste é realizado mais cedo no SDLC. Normalmente, o shift-left sugere que os testes devem ser feitos mais cedo (p. ex., não esperar que o código seja implementado ou que os componentes sejam integrados), mas isso não significa que os testes posteriores no SDLC devam ser negligenciados.

Existem algumas boas práticas que ilustram como obter um "shift-left" nos testes, que incluem:

- Revisão da especificação sob a perspectiva de testes. Essas atividades de revisão das especificações geralmente encontram possíveis defeitos, como ambiguidades, incompletude e inconsistências;
- Escrever casos de teste antes de o código ser escrito e fazer com que o código seja executado em um conjunto de testes durante a sua implementação;
- Usar a CI e, melhor ainda, a CD, pois ela vem com feedback rápido e testes de componentes automatizados para acompanhar o código-fonte quando ele é enviado ao repositório de código;

- Concluir a análise estática do código-fonte antes do teste dinâmico ou como parte de um processo automatizado;
- Realizar testes não funcionais começando no nível de teste do componente, sempre que possível. Essa é uma forma de shift-left, pois esses tipos de testes não funcionais tendem a ser realizados mais tarde no SDLC, quando um sistema completo e um ambiente de teste representativo estão disponíveis.

Uma abordagem shift-left pode resultar em treinamento, esforço e/ou custos adicionais no início do processo, mas espera-se que economize esforços e/ou custos no final do processo.

Para a abordagem shift-left, é importante que os stakeholders sejam convencidos a aceitarem esse conceito.

### 2.1.6 Retrospectivas e melhoria de processos

As retrospectivas (também conhecidas como "reuniões pós-projeto" e retrospectivas de projeto) geralmente são realizadas no final de um projeto ou de uma iteração, em um marco de lançamento, ou podem ser realizadas quando necessário. O momento e a organização das retrospectivas dependem do modelo específico de SDLC que está sendo seguido. Nessas reuniões, os participantes (não apenas os Testadores, mas também, por exemplo, Desenvolvedores, Arquitetos, Product Owner, Analistas de Negócios) discutem:

- O que foi bem-sucedido e deve ser mantido?
- O que não foi bem-sucedido e poderia ser melhorado?
- Como incorporar as melhorias e manter os sucessos no futuro?

Os resultados devem ser registrados e normalmente fazem parte do relatório de conclusão do teste (ver seção 5.3.2). As retrospectivas são essenciais para a implementações bem-sucedidas da melhoria contínua e é importante que todas as melhorias recomendadas sejam acompanhadas.

Os típicos benefícios dos testes incluem:

- Aumento da eficácia/eficiência do teste (p. ex., implementando sugestões de aprimoramento do processo);
- Aumento da qualidade do material de teste (p. ex., por meio da revisão conjunta dos processos de teste);
- Vínculo e aprendizado da equipe (p. ex., como resultado da oportunidade de levantar questões e propor pontos de melhoria);
- Melhoria da qualidade da base de teste (p. ex., como as deficiências na extensão e na qualidade dos requisitos podem ser abordadas e resolvidas);
- Melhor cooperação entre desenvolvimento e testes (p. ex., como a colaboração é revisada e otimizada regularmente);

## 2.2 Níveis de Teste e Tipos de Teste

Os níveis de teste são grupos de atividades de teste que são organizadas e gerenciadas em conjunto. Cada nível de teste é uma instância do processo de teste, realizado em relação ao software em um

determinado estágio de desenvolvimento, desde componentes individuais até sistemas completos ou, quando aplicável, sistemas de sistemas.

Os níveis de teste estão relacionados a outras atividades dentro do SDLC. Nos modelos sequenciais do SDLC, os níveis de teste são geralmente definidos de forma que os critérios de saída de um nível façam parte dos critérios de entrada do próximo nível. Em alguns modelos iterativos, isso pode não se aplicar. As atividades de desenvolvimento podem se estender por vários níveis de teste. Os níveis de teste podem se sobrepor no tempo.

Os tipos de teste são grupos de atividades de teste relacionadas a características de qualidade específicas e a maioria dessas atividades de teste pode ser realizada em todos os níveis de teste.

### 2.2.1 Níveis de Teste

Neste syllabus, são descritos os cinco níveis de teste a seguir:

- O **Teste de Componente** (também conhecido como Teste de Unidade) concentra-se em testar componentes isoladamente. Geralmente, requer suporte específico, como estruturas de teste ou frameworks de teste de unidade. Normalmente, os testes de componentes são realizados por desenvolvedores em seus ambientes de desenvolvimento.
- O **Teste de Integração de Componentes** (também conhecido como Teste de Integração de Unidades) concentra-se no teste das interfaces e interações entre os componentes. O teste de integração de componentes depende muito das abordagens da estratégia de integração, como *bottom-up*, *top-down* ou *big-bang*.
- O **Teste de Sistema** concentra-se no comportamento geral e nos recursos de todo um sistema ou produto, geralmente incluindo o teste funcional de tarefas de ponta a ponta e o teste não funcional de características de qualidade. Para algumas características de qualidade não funcionais, é preferível testá-las em um sistema completo em um ambiente de teste representativo (p. ex., usabilidade). Também é possível usar simulações de subsistemas. O teste do sistema pode ser realizado por uma equipe de teste independente e está relacionado às especificações do sistema.
- O **Teste de Integração de Sistema** concentra-se no teste das interfaces do sistema e de outros sistemas e serviços externos. O teste de integração do sistema requer ambientes de teste adequados, de preferência semelhantes ao ambiente operacional.
- O **Teste de Aceite** concentra-se na validação e na demonstração da disposição para a implantação, o que significa que o sistema atende às necessidades (requisitos) de negócio do usuário. Preferencialmente, o teste de aceite deve ser realizado pelos usuários previstos. As principais formas de teste de aceite são: Teste de Aceite do Usuário (UAT), Teste de Aceite Operacional, Teste de Aceite Contratual e Normativo, Teste Alfa e Teste Beta.

Os níveis de teste são diferenciados pela seguinte lista incompleta de atributos, para evitar a sobreposição de atividades de teste:

- Objeto de teste;
- Objetivos do teste;
- Base de teste;
- Defeitos e falhas;
- Abordagem e responsabilidades.

## 2.2.2 Tipos de Teste

Existem muitos tipos de testes que podem ser aplicados em projetos. Neste programa, são abordados os quatro tipos de teste a seguir:

O **Teste Funcional** avalia as funções que um componente ou sistema deve executar. As funções são "o que" o objeto de teste deve fazer. O principal objetivo do teste funcional é verificar a integridade funcional, a correção funcional e a adequação funcional**.**

O **Teste Não Funcional** avalia atributos que não sejam características funcionais de um componente ou sistema. O teste não funcional é o teste de "quão bem o sistema se comporta". O principal objetivo do teste não funcional é verificar as características não funcionais da qualidade do software. A norma ISO/IEC 25010 fornece a seguinte classificação das características não funcionais de qualidade do software:

- Eficiência de Performance;
- Compatibilidade;
- Usabilidade;
- Confiabilidade;
- Segurança;
- Capacidade de manutenção;
- Portabilidade.

Às vezes, é apropriado que os testes não funcionais comecem no início do ciclo de vida (p. ex., como parte de revisões e testes de componentes ou testes de sistema). Muitos testes não funcionais são derivados de testes funcionais, pois usam os mesmos testes funcionais, mas verificam se, ao executar a função, uma restrição não funcional é atendida (p. ex., verificar se uma função é executada dentro de um tempo especificado ou se uma função pode ser portada para uma nova plataforma). A descoberta tardia de defeitos não funcionais pode representar uma séria ameaça ao sucesso de um projeto. Às vezes, os testes não funcionais precisam de um ambiente de teste muito específico, como um laboratório de usabilidade para testes de usabilidade.

O **Teste Caixa-Preta** (ver seção 4.2) é baseado em especificações e deriva testes da documentação externa ao objeto de teste. O principal objetivo do teste caixa-preta é verificar o comportamento do sistema em relação às suas especificações.

O **Teste Caixa-Branca** (ver seção 4.3) é baseado na estrutura e deriva testes da implementação ou da estrutura interna do sistema (p. ex., código, arquitetura, fluxos de trabalho e fluxos de dados). O principal objetivo do teste caixa-branca é cobrir a estrutura subjacente pelos testes até o nível aceitável.

Todos os quatro tipos de teste mencionados acima podem ser aplicados a todos os níveis de teste, embora o foco seja diferente em cada nível. Diferentes técnicas de teste podem ser usadas para derivar condições de teste e casos de teste para todos os tipos de teste mencionados.

### 2.2.3 Testes de Confirmação e Teste de Regressão

Normalmente, as alterações são feitas em um componente ou sistema para aprimorá-lo, adicionando um novo recurso, ou para corrigi-lo, removendo um defeito. Os testes também devem incluir testes de confirmação e testes de regressão.

O **Teste de Confirmação** confirma que um defeito original foi corrigido com sucesso. Dependendo do risco, é possível testar a versão corrigida do software de várias maneiras, inclusive:

- executando todos os casos de teste que falharam anteriormente devido ao defeito, ou, também,
- adicionar novos testes para cobrir quaisquer alterações necessárias para corrigir o defeito

No entanto, quando há pouco tempo ou dinheiro para corrigir defeitos, o teste de confirmação pode se restringir a simplesmente executar as etapas que devem reproduzir a falha causada pelo defeito e verificar se a falha não ocorre.

Os **Teste de Regressão** confirmam que nenhuma consequência adversa foi causada por uma alteração, inclusive uma correção que já tenha sido confirmada após o teste. Essas consequências adversas podem afetar o mesmo componente em que a alteração foi feita, outros componentes do mesmo sistema ou até mesmo outros sistemas conectados. O teste de regressão pode não se restringir ao objeto de teste em si, mas também pode estar relacionado ao ambiente. É aconselhável realizar primeiro uma análise de impacto para otimizar a extensão do teste de regressão. A análise de impacto mostra quais partes do software podem ser afetadas.

Os conjuntos de testes de regressão são executados muitas vezes e, em geral, o número de casos de teste de regressão aumentará a cada iteração ou versão, portanto, os testes de regressão são fortes candidatos à automação. A automação desses testes deve começar no início do projeto. Quando a CI é usada, como no DevOps (ver seção 2.1.4), é uma boa prática incluir também testes de regressão automatizados. Dependendo da situação, isso pode incluir testes de regressão em diferentes níveis.

Testes de confirmação e/ou testes de regressão para o objeto de teste são necessários em todos os níveis de teste se os defeitos forem corrigidos e/ou se forem feitas alterações nesses níveis de teste.

## 2.3 Teste de Manutenção

Há diferentes categorias de manutenção, que podem ser corretivas, adaptáveis a mudanças no ambiente ou melhorar a performance ou a capacidade de manutenção (ver ISO/IEC 14764 para mais detalhes), portanto, a manutenção pode envolver versões/implantações planejadas e versões/implantações não planejadas (*hot fixes*). A análise de impacto pode ser feita antes de uma alteração, para ajudar a decidir se a alteração deve ser feita, com base nas possíveis consequências em outras áreas do sistema. O teste das alterações em um sistema em produção inclui a avaliação do sucesso da implementação da alteração e a verificação de possíveis regressões em partes do sistema que permanecem inalteradas (que geralmente é a sua maior parte).

O escopo dos testes de manutenção geralmente depende de:

- O grau de risco da mudança;
- O tamanho do sistema existente;

- O tamanho da mudança.

Os fatores para manutenção e teste de manutenção podem ser classificados:

- Modificações, como aprimoramentos planejados (ou seja, baseados em versões), alterações corretivas ou *hot fixes*;
- Atualizações ou migrações do ambiente operacional, como de uma plataforma para outra, o que pode exigir testes associados ao novo ambiente, bem como ao software alterado, ou testes de conversão de dados quando os dados de outro aplicativo são migrados para o sistema que está sendo mantido.
- Aposentadoria, por exemplo, quando um aplicativo chega ao fim de sua vida útil. Quando um sistema é desativado, isso pode exigir testes de arquivamento de dados se forem necessários longos períodos de retenção de dados. O teste dos procedimentos de restauração e recuperação após o arquivamento também pode ser necessário caso determinados dados sejam necessários durante o período de arquivamento.

# 3 Teste Estático (80 min)

## Palavras-chave

anomalia, teste dinâmico, revisão formal, revisão informal, inspeção, revisão, análise estática, teste estático, revisão técnica, walkthrough

## Objetivos de aprendizagem

### 3.1 Noções básicas de Teste Estático

FL-3.1.1 (K1) Reconhecer os tipos de produtos que podem ser examinados pelas diferentes técnicas de teste estático.

FL-3.1.2 (K2) Explicar o valor dos testes estáticos.

FL-3.1.3 (K2) Comparar e contrastar testes estáticos e dinâmicos.

### 3.2 Processo de feedback e revisão

FL-3.2.1 (K1) Identificar os benefícios do feedback antecipado e frequente dos stakeholders.

FL-3.2.2 (K2) Resumir as atividades do processo de revisão.

FL-3.2.3 (K1) Relembrar quais responsabilidades são atribuídas às funções principais ao realizar revisões.

FL-3.2.4 (K2) Comparar e contrastar os diferentes tipos de revisão.

FL-3.2.5 (K1) Relembrar os fatores que contribuem para uma revisão bem-sucedida.

## 3.1 Noções básicas de Teste Estático

Ao contrário dos testes dinâmicos, nos testes estáticos o software em teste não precisa ser executado. O código, a especificação do processo, a especificação da arquitetura do sistema ou outros produtos de trabalho são avaliados por meio de exame manual (p. ex., revisões) ou com a ajuda de uma ferramenta (p. ex., análise estática). Os objetivos do teste incluem a melhoria da qualidade, a detecção de defeitos e a avaliação de características como legibilidade, integridade, correção, testabilidade e consistência. O teste estático pode ser aplicado tanto para verificação quanto para validação.

Testadores, representantes do negócio e desenvolvedores trabalham juntos durante o mapeamento de exemplos, escrita colaborativa de histórias de usuários e sessões de refinamento do backlog para garantir que as histórias de usuários e os produtos de trabalho relacionados atendam aos critérios definidos, por exemplo, a Definição de Pronto (*DoR - Definition of Ready*) (ver seção 5.1.3). Técnicas de revisão podem ser aplicadas para garantir que as histórias de usuários sejam completas e compreensíveis e incluam critérios de aceite testáveis. Ao fazer as perguntas certas, os Testadores exploram, desafiam e ajudam a melhorar as histórias de usuário propostas.

A análise estática pode identificar problemas antes dos testes dinâmicos e, muitas vezes, exige menos esforço, já que não são necessários casos de teste e, normalmente, são usadas ferramentas (ver capítulo 6). A análise estática é frequentemente incorporada às estruturas de CI (ver seção 2.1.4). Embora seja amplamente usada para detectar defeitos específicos no código, a análise estática também é usada para avaliar a capacidade de manutenção e a segurança. Verificadores ortográficos e ferramentas de legibilidade são outros exemplos de ferramentas de análise estática.

### 3.1.1 Produtos de trabalho examináveis por testes estáticos

Quase todos os produtos de trabalho podem ser examinados por meio de testes estáticos. Os exemplos incluem documentos de especificação de requisitos, código-fonte, planos de teste, casos de teste, itens da lista de pendências do produto, cartas de teste, documentação do projeto, contratos e modelos.

Qualquer produto de trabalho que possa ser lido e compreendido pode ser objeto de uma revisão. No entanto, para a análise estática, os produtos de trabalho precisam de uma estrutura em relação à qual possam ser verificados (p. ex., modelos, código ou texto com uma sintaxe formal).

Os produtos de trabalho que não são apropriados para testes estáticos incluem aqueles que são difíceis de serem interpretados por seres humanos e que não podem ser analisados nem por ferramentas (p. ex., código executável de terceiros por motivos legais).

### 3.1.2 Valor do teste estático

O teste estático pode detectar defeitos nas primeiras fases do SDLC, cumprindo o princípio do teste antecipado (ver seção 1.3). Ele também pode identificar defeitos que não podem ser detectados por testes dinâmicos (p. ex., código inacessível, padrões de projeto não implementados conforme desejado, defeitos em produtos de trabalho não executáveis).

Os testes estáticos permitem avaliar a qualidade e criar confiança nos produtos de trabalho. Ao verificar os requisitos documentados, os stakeholders também podem se certificar de que esses requisitos descrevem suas necessidades reais. Como os testes estáticos podem ser realizados no início do SDLC, é possível criar um entendimento compartilhado entre os stakeholders envolvidos. A comunicação entre os stakeholders também será aprimorada. Por esse motivo, é recomendável envolver uma grande variedade de participantes nos testes estáticos.

Embora a implementação das revisões possa ser dispendiosa, os custos gerais do projeto geralmente são muito menores do que quando não são realizadas revisões, pois é necessário gastar menos tempo e esforço para corrigir defeitos posteriormente no projeto.

Os defeitos de código podem ser detectados usando a análise estática de forma mais eficiente do que em testes dinâmicos, geralmente resultando em menos defeitos de código e em um menor esforço geral de desenvolvimento.

### 3.1.3 Diferenças entre testes estáticos e testes dinâmicos

As práticas de teste estático e de teste dinâmico se complementam. Elas têm objetivos semelhantes, como apoiar a detecção de defeitos em produtos de trabalho (ver seção 1.1.1), mas também existem algumas diferenças, como:

- Os testes estáticos e dinâmicos (com análise de falhas) podem levar à detecção de defeitos, mas há alguns tipos de defeitos que só podem ser encontrados por meio de testes estáticos ou dinâmicos;
- Os testes estáticos encontram defeitos diretamente, enquanto os testes dinâmicos causam falhas a partir das quais os defeitos associados são determinados por meio de análises subsequentes;
- Os testes estáticos podem detectar com mais facilidade os defeitos que se encontram nos caminhos do código que raramente são executados ou que são difíceis de alcançar usando testes dinâmicos;
- O teste estático pode ser aplicado a produtos de trabalho não executáveis, enquanto o teste dinâmico só pode ser aplicado a produtos de trabalho executáveis;
- Os testes estáticos podem ser usados para medir as características de qualidade que não dependem da execução do código (p. ex., capacidade de manutenção), enquanto os testes dinâmicos podem ser usados para medir as características de qualidade que dependem da execução do código (p. ex., eficiência de performance).

Os defeitos típicos que são mais fáceis e/ou mais baratos de encontrar por meio de testes estáticos incluem:

- Defeitos nos requisitos (p. ex., inconsistências, ambiguidades, contradições, omissões, imprecisões, duplicações)
- Defeitos de projeto (p. ex., estruturas de banco de dados ineficientes, modularização deficiente)
- Certos tipos de defeitos de codificação (p. ex., variáveis com valores indefinidos, variáveis não declaradas, código inacessível ou duplicado, complexidade excessiva do código)

- Desvios dos padrões (p. ex., falta de adesão às convenções de nomenclatura nos padrões de codificação)
- Especificações incorretas da interface (p. ex., número, tipo ou ordem de parâmetros incompatíveis)
- Tipos específicos de vulnerabilidades de segurança (p. ex., estouro de buffer)
- Lacunas ou imprecisões na cobertura da base de testes (p. ex., testes ausentes para um critério de aceite)

## 3.2 Processo de feedback e revisão

### 3.2.1 Benefícios do feedback antecipado e frequente dos stakeholders

O feedback antecipado e frequente permite a comunicação precoce de possíveis problemas de qualidade. Se houver pouco envolvimento dos stakeholders durante o SDLC, o produto que está sendo desenvolvido pode não atender à visão original ou atual dos stakeholders. A incapacidade de entregar o que o stakeholder deseja pode resultar em retrabalho dispendioso, perda de prazos, jogos de culpa e até mesmo levar ao fracasso total do projeto.

O feedback frequente dos stakeholders durante todo o SDLC pode evitar mal-entendidos sobre os requisitos e garantir que as alterações nos requisitos sejam compreendidas e implementadas mais cedo. Isso ajuda a equipe de desenvolvimento a melhorar a compreensão do que está sendo desenvolvido. Isso permite que a equipe se concentre nos recursos que agregam mais valor aos stakeholders e que têm o maior impacto positivo sobre os riscos identificados.

### 3.2.2 Atividades do processo de revisão

A norma ISO/IEC 20246 define um processo de revisão genérico que fornece um framework estruturado, porém flexível, a partir da qual um processo específico de revisão pode ser adaptado a uma situação particular. Se a revisão exigida for mais formal, serão necessárias mais tarefas descritas para as diferentes atividades.

O tamanho de muitos produtos de trabalho os torna grandes demais para serem cobertos por uma única revisão. O processo de revisão pode ser invocado algumas vezes para concluir a revisão de todo o produto de trabalho.

As atividades no processo de revisão são:

- **Planejamento.** Durante a fase de planejamento, deve ser definido o escopo da revisão, que inclui o objetivo, o produto de trabalho a ser revisado, as características de qualidade a serem avaliadas, as áreas a serem enfocadas, os critérios de saída, as informações de apoio, como padrões, o esforço e os prazos para a revisão.
- **Início da revisão.** Durante o início da revisão, o objetivo é garantir que todos os envolvidos estejam preparados para começar a revisão. Isso inclui garantir que todos os participantes tenham acesso ao produto de trabalho, entendam suas funções e responsabilidades e recebam tudo o que for necessário para realizar a análise.
- **Revisão individual.** Cada revisor realiza uma revisão individual para avaliar a qualidade do produto de trabalho sob revisão e para identificar anomalias, recomendações e perguntas,

aplicando uma ou mais técnicas de revisão (p. ex., revisão baseada em lista de verificação, revisão baseada em cenário). A norma ISO/IEC 20246 fornece mais detalhes sobre as diferentes técnicas de revisão. Os revisores registram todas as anomalias, recomendações e perguntas identificadas.

- **Comunicação e Análise de Problemas.** Como as anomalias identificadas durante uma revisão não são necessariamente defeitos, todas essas anomalias precisam ser analisadas e discutidas. Para cada uma encontrada, deve ser tomada uma decisão sobre seu status, propriedade e ações necessárias. Normalmente, isso é feito em uma reunião de revisão, durante a qual os participantes também decidem qual é o nível de qualidade do produto de trabalho revisado e quais ações de acompanhamento são necessárias. Pode ser necessária uma revisão de acompanhamento para concluir as ações.
- **Correção e Relatório.** Para cada defeito, deve ser criado um relatório de defeitos para que as ações corretivas possam ser acompanhadas. Quando os critérios de saída forem atingidos, o produto de trabalho poderá ser aceito. Os resultados da revisão são relatados.

### 3.2.3 Funções e responsabilidades nas revisões

As revisões envolvem vários stakeholders, que podem assumir diversas funções. As principais, e suas responsabilidades são:

- **Gerente**: decide o que deve ser revisado e fornece recursos, como equipe e tempo para a revisão
- **Autor**: cria e corrige o produto de trabalho em análise
- **Moderador** (também conhecido como facilitador): garante o andamento eficaz das reuniões de revisão, incluindo mediação, gerenciamento de tempo e um ambiente de revisão seguro no qual todos possam falar livremente
- **Relator** (também conhecido como registrador): reúne as anomalias dos revisores e registra as informações da revisão, como decisões e novas anomalias encontradas durante a reunião de revisão
- **Revisor**: realiza revisões. Um revisor pode ser alguém que esteja trabalhando no projeto, um especialista no assunto ou qualquer outra parte interessada
- **Líder da revisão**: assume a responsabilidade geral pela revisão, como decidir quem estará envolvido e organizar quando e onde a revisão será realizada

Outras funções mais detalhadas são possíveis, conforme descrito na norma ISO/IEC 20246.

### 3.2.4 Tipos de revisão

Existem muitos tipos de revisão, desde revisões informais até revisões formais. O nível necessário de formalidade depende de fatores como o SDLC que está sendo seguido, a maturidade do processo de desenvolvimento, a importância e a complexidade do produto de trabalho que está sendo revisado, os requisitos legais ou regulamentares e a necessidade de uma trilha de auditoria. O mesmo produto de trabalho pode ser revisado com diferentes tipos de revisão, por exemplo, primeiro uma informal e depois uma mais formal.

A seleção do tipo certo de revisão é fundamental para atingir os objetivos de revisão exigidos (ver seção 3.2.5). A seleção não se baseia apenas nos objetivos, mas também em fatores como as necessidades do projeto, os recursos disponíveis, o tipo e os riscos do produto de trabalho, o domínio do negócio e a cultura da empresa.

Alguns tipos de revisão comumente usados são:

- **Revisão informal.** As revisões informais não seguem um processo definido e não exigem um resultado formal documentado. O principal objetivo é detectar anomalias.
- **Walkthrough.** Um passo a passo, conduzido pelo autor, pode servir a muitos objetivos, como avaliar a qualidade e criar confiança no produto do trabalho, instruir os revisores, obter consenso, gerar novas ideias, motivar e permitir que os autores melhorem e detectar anomalias. Os revisores podem realizar uma revisão individual antes do passo a passo, mas isso não é obrigatório.
- **Revisão técnica.** Uma revisão técnica é realizada por revisores tecnicamente qualificados e liderada por um moderador. Os objetivos de uma revisão técnica são obter consenso e tomar decisões em relação a um problema técnico, mas também detectar anomalias, avaliar a qualidade e criar confiança no produto do trabalho, gerar novas ideias e motivar e capacitar os autores a melhorar.
- **Inspeção.** Como as inspeções são o tipo mais formal de revisão, elas seguem o processo genérico completo (ver seção 3.2.2). O objetivo principal é encontrar o número máximo de anomalias. Outros objetivos são avaliar a qualidade, criar confiança no produto do trabalho e motivar e permitir que os autores melhorem. As métricas são coletadas e usadas para aprimorar o SDLC, inclusive o processo de inspeção. Nas inspeções, o autor não pode atuar como líder ou relator da revisão.

### 3.2.5 Fatores de sucesso para revisões

Há vários fatores que determinam o sucesso das revisões, que incluem:

- Definir objetivos claros e critérios de saída mensuráveis. A avaliação dos participantes nunca deve ser um objetivo;
- Escolher o tipo de revisão apropriado para atingir os objetivos determinados e se adequar ao tipo de produto de trabalho, aos participantes da revisão, às necessidades e ao contexto do projeto;
- Realizar revisões em pequenas partes, de modo que os revisores não percam a concentração durante uma revisão individual e/ou a reunião de revisão (quando realizada);
- Fornecer feedback das revisões aos stakeholders e aos autores para que eles possam melhorar o produto e suas atividades (ver seção 3.2.1);
- Fornecer tempo suficiente para que os participantes se preparem para a revisão;
- Apoio da gerência para o processo de revisão;
- Tornar as revisões parte da cultura da organização, para promover o aprendizado e o aprimoramento dos processos;
- Fornecer treinamento adequado a todos os participantes para que eles saibam como desempenhar suas funções;
- Facilitação de reuniões;

# 4 Análise e Modelagem de Teste (390 min)

## Palavras-chave

critérios de aceite, desenvolvimento orientado por teste de aceite, técnica de teste caixa-preta, análise de valor de limite, cobertura de ramificação, teste baseado em lista de verificação, abordagem de teste baseada na colaboração, cobertura, item de cobertura, teste de tabela de decisão, particionamento de equivalência, suposição de erro, técnica de teste baseada na experiência, teste exploratório, teste de transição de estado, cobertura de instrução, técnica de teste, técnica de teste caixa-branca

## Objetivos de aprendizagem

### 4.1 Visão geral das técnicas de teste

FL-4.1.1 (K2) Distinguir técnicas de teste baseadas em caixa-preta, caixa-branca e experiência.

### 4.2 Técnicas de Teste Caixa-Preta

FL-4.2.1 (K3) Usar o particionamento de equivalência para derivar casos de teste.

FL-4.2.2 (K3) Usar a análise de valor limite para derivar casos de teste.

FL-4.2.3 (K3) Usar testes de tabela de decisão para derivar casos de teste.

FL-4.2.4 (K3) Usar o teste de transição de estado para derivar casos de teste.

### 4.3 Técnicas de Teste Caixa-Branca

FL-4.3.1 (K2) Explicar o teste de instrução.

FL-4.3.2 (K2) Explicar o teste de ramificação.

FL-4.3.3 (K2) Explicar o valor dos testes caixa-branca.

### 4.4 Técnicas de Teste Baseadas na Experiência

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

FL-4.4.2 (K2) Explicar o teste exploratório.

FL-4.4.3 (K2) Explicar os testes baseados em listas de verificação.

### 4.5. Abordagens de Teste Baseadas na colaboração

FL-4.5.1 (K2) Explicar como escrever histórias de usuários em colaboração com desenvolvedores e representantes de negócio.

FL-4.5.2 (K2) Classificar as diferentes opções para escrever critérios de aceite.

FL-4.5.3 (K3) Usar o desenvolvimento orientado por testes de aceite (ATDD) para derivar casos de teste.

## 4.1 Visão geral das técnicas de teste

As técnicas de teste dão suporte ao Testador na análise do teste (o que testar) e no projeto do teste (como testar). As técnicas de teste ajudam a desenvolver um conjunto relativamente pequeno, mas suficiente, de casos de teste de forma sistemática. As técnicas de teste também ajudam o Testador a definir as condições de teste, identificar os itens de cobertura e identificar os dados de teste durante a análise e o projeto do teste. Mais informações sobre técnicas de teste e suas medidas correspondentes podem ser encontradas no padrão ISO/IEC/IEEE 29119-4 e em (Beizer 1990, Craig 2002, Copeland 2004, Koomen 2006, Jorgensen 2014, Ammann 2016, Forgács 2019).

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

As **Técnicas de Teste Caixa-Preta** (também conhecidas como técnicas baseadas em especificações) são baseadas em uma análise do comportamento específico do objeto de teste sem referência à sua estrutura interna. Portanto, os casos de teste são independentes de como o software é implementado. Consequentemente, se a implementação mudar, mas o comportamento exigido permanecer o mesmo, os casos de teste ainda serão úteis.

As **Técnicas de Teste Caixa-Branca** (também conhecidas como técnicas baseadas na estrutura) baseiam-se em uma análise da estrutura interna e do processamento do objeto de teste. Como os casos de teste dependem de como o software é projetado, eles só podem ser criados após o projeto ou a implementação do objeto de teste.

As **Técnicas de Teste Baseada na Experiência** usam efetivamente o conhecimento e a experiência dos Testadores para o projeto e a implementação dos casos de teste. A eficácia dessas técnicas depende muito das habilidades do Testador. As técnicas de teste baseadas na experiência podem detectar defeitos que podem não ser detectados usando as técnicas de teste caixa-preta e de caixa-branca. Portanto, as técnicas de teste baseadas na experiência são complementares às técnicas de teste caixa-preta e de caixa-branca.

## 4.2 Técnicas de Teste Caixa-Preta

As técnicas de teste caixa-preta comumente usadas e discutidas nas seções a seguir são:

- Particionamento de equivalência;
- Análise de valor de limite;
- Teste de tabela de decisão;
- Teste de transição de estado;

### 4.2.1 Particionamento de Equivalência (EP)

O Particionamento de Equivalência (EP) divide os dados em partições (conhecidas como partições de equivalência) com base na expectativa de que todos os elementos de uma determinada partição sejam processados da mesma forma pelo objeto de teste. A teoria por trás dessa técnica é que, se um caso de teste, que testa um valor de uma partição de equivalência, detectar um defeito, esse

defeito também deverá ser detectado por casos de teste que testem qualquer outro valor da mesma partição. Portanto, um teste para cada partição é suficiente.

As partições de equivalência podem ser identificadas para qualquer elemento de dados relacionado ao objeto de teste, incluindo entradas, saídas, itens de configuração, valores internos, valores relacionados ao tempo e parâmetros de interface. As partições podem ser contínuas ou discretas, ordenadas ou não ordenadas, finitas ou infinitas. As partições não devem se sobrepor e devem ser conjuntos não vazios.

No caso de objetos de teste simples, o EP pode ser fácil, mas, na prática, entender como o objeto de teste tratará valores diferentes costuma ser complicado. Portanto, o particionamento deve ser feito com cuidado.

Uma partição que contém valores válidos é chamada de partição válida. Uma partição que contém valores inválidos é chamada de partição inválida. As definições de valores válidos e inválidos podem variar entre equipes e organizações. Por exemplo, os valores válidos podem ser interpretados como aqueles que devem ser processados pelo objeto de teste ou como aqueles para os quais a especificação define seu processamento. Os valores inválidos podem ser interpretados como aqueles que devem ser ignorados ou rejeitados pelo objeto de teste ou como aqueles para os quais nenhum processamento é definido na especificação do objeto de teste.

No EP, os itens de cobertura são as partições de equivalência. Para atingir 100% de cobertura com essa técnica, os casos de teste devem executar todas as partições identificadas (inclusive as inválidas), cobrindo cada partição pelo menos uma vez. A cobertura é medida como o número de partições executadas por pelo menos um caso de teste, dividido pelo número total de partições identificadas, e é expressa como uma porcentagem.

Muitos objetos de teste incluem vários conjuntos de partições (p. ex., objetos de teste com mais de um parâmetro de entrada), o que significa que um caso de teste abrangerá partições de diferentes conjuntos de partições. O critério de cobertura mais simples no caso de vários conjuntos de partições é chamado de *Each Choice Coverage* (ECC) (Ammann 2016). A cobertura de cada escolha exige que os casos de teste executem cada partição de cada conjunto de partições pelo menos uma vez. A cobertura de cada escolha não leva em conta as combinações de partições.

### 4.2.2 Análise de Valor de Limite (BVA)

A BVA (Boundary Value Analysis) é uma técnica baseada na execução dos limites das partições de equivalência. Portanto, a BVA só pode ser usada para partições ordenadas. Os valores mínimo e máximo de uma partição são seus valores de limite. No caso da BVA, se dois elementos pertencem à mesma partição, todos os elementos entre eles também devem pertencer a essa partição.

O BVA concentra-se nos valores de limite das partições porque os desenvolvedores têm maior probabilidade de cometer erros com esses valores de limite. Os defeitos típicos encontrados pelo BVA estão localizados onde os limites implementados estão mal posicionados em posições acima ou abaixo das posições pretendidas ou são totalmente omitidos.

Este syllabus abrange duas versões do BVA: BVA de 2 valores e BVA de 3 valores. Elas diferem em termos de itens de cobertura por limite que precisam ser executados para atingir 100% de cobertura.

No BVA de 2 valores (Craig 2002, Myers 2011), para cada valor de limite há dois itens de cobertura: esse valor de limite e seu vizinho mais próximo pertencente à partição adjacente. Para atingir 100% de cobertura com o BVA de 2 valores, os casos de teste devem executar todos os itens de cobertura, ou seja, todos os valores de limite identificados. A cobertura é medida como o número de valores de limite que foram executados, dividido pelo número total de valores de limite identificados, e é expressa como uma porcentagem.

No BVA de 3 valores (Koomen 2006, O'Regan 2019), para cada valor de limite há três itens de cobertura: esse valor de limite e seus dois vizinhos. Portanto, no BVA de 3 valores, alguns dos itens de cobertura podem não ser valores de limite. Para atingir 100% de cobertura com o BVA de 3 valores, os casos de teste devem executar todos os itens de cobertura, ou seja, os valores de limite identificados e seus vizinhos. A cobertura é medida como o número de valores de limite e seus vizinhos executados, dividido pelo número total de valores de limite identificados e seus vizinhos, e é expressa como uma porcentagem.

O BVA de 3 valores é mais rigoroso do que o BVA de 2 valores, pois pode detectar defeitos ignorados pelo BVA de 2 valores. Por exemplo, se a decisão "se ($x \leq 10$) ..." for implementada incorretamente como "se (x = 10) ...", nenhum dado de teste derivado do BVA de 2 valores (x = 10, x = 11) poderá detectar o defeito. Entretanto, x = 9, derivado do BVA de 3 valores, provavelmente detectará o defeito.

### 4.2.3 Teste de Tabela de Decisão

As tabelas de decisão são usadas para testar a implementação dos requisitos do sistema que especificam como diferentes combinações de condições resultam em diferentes resultados. As tabelas de decisão são uma forma eficaz de registrar lógicas complexas, como regras de negócios.

Ao criar tabelas de decisão, são definidas as condições e as ações resultantes do sistema. Elas formam as linhas da tabela. Cada coluna corresponde a uma regra de decisão que define uma combinação única de condições, juntamente com as ações associadas. Nas tabelas de decisão de entrada limitada, todos os valores das condições e ações (exceto os irrelevantes ou inviáveis; veja abaixo) são mostrados como valores booleanos (verdadeiro ou falso). Como alternativa, em tabelas de decisão de entrada estendida, algumas ou todas as condições e ações também podem assumir valores múltiplos (p. ex., intervalos de números, partições de equivalência, valores discretos).

A notação para condições é a seguinte: "V" (verdadeiro) significa que a condição foi satisfeita. "F" (falso) significa que a condição não foi satisfeita. "-" significa que o valor da condição é irrelevante para o resultado da ação. "N/A" significa que a condição é inviável para uma determinada regra. Para ações: "X" significa que a ação deve ocorrer. Em branco significa que a ação não deve ocorrer. Outras notações também podem ser usadas.

Uma tabela de decisão completa tem colunas suficientes para cobrir todas as combinações de condições. A tabela pode ser simplificada com a exclusão de colunas que contenham combinações inviáveis de condições. A tabela também pode ser minimizada pela fusão de colunas, nas quais algumas condições não afetam o resultado, em uma única coluna. Os algoritmos de minimização da tabela de decisão estão fora do escopo deste programa.

No teste de tabela de decisão, os itens de cobertura são as colunas que contêm combinações viáveis de condições. Para atingir 100% de cobertura com essa técnica, os casos de teste devem executar

todas essas colunas. A cobertura é medida como o número de colunas executadas, dividido pelo número total de colunas viáveis, e é expressa como uma porcentagem.

O ponto forte do teste de tabela de decisão é que ele oferece uma abordagem sistemática para identificar todas as combinações de condições, algumas das quais poderiam ser negligenciadas. Ele também ajuda a encontrar quaisquer lacunas ou contradições nos requisitos. Se houver muitas condições, a execução de todas as regras de decisão pode consumir muito tempo, pois o número de regras cresce exponencialmente com o número de condições. Nesse caso, para reduzir o número de regras que precisam ser executadas, pode ser usada uma tabela de decisão minimizada ou uma abordagem baseada em risco.

### 4.2.4 Teste de Transição de Estado

Um diagrama de transição de estado modela o comportamento de um sistema, mostrando seus possíveis estados e transições de estado válidas. Uma transição é iniciada por um evento, que pode ser qualificado adicionalmente por uma condição de proteção. Presume-se que as transições sejam instantâneas e, às vezes, podem resultar em ações do software. A sintaxe comum de rotulagem de transição é a seguinte: "evento [condição de proteção] / ação". As condições de proteção e as ações podem ser omitidas se não existirem ou se forem irrelevantes para o Testador.

Uma tabela de estados é um modelo equivalente a um diagrama de transição de estados. Suas linhas representam estados e suas colunas representam eventos (juntamente com condições de proteção, se existirem). As entradas (células) da tabela representam transições e contêm o estado de destino, bem como as ações resultantes, se definidas. Ao contrário do diagrama de transição de estados, a tabela de estados mostra explicitamente as transições inválidas, que são representadas por células vazias.

Um caso de teste baseado em um diagrama de transição de estado ou em uma tabela de estados geralmente é representado como uma sequência de eventos, que resulta em uma sequência de alterações de estado (e ações, se necessário). Um caso de teste pode abranger, e geralmente abrangerá, várias transições entre estados.

Existem muitos critérios de cobertura para o teste de transição de estado. Este syllabus aborda três deles.

Na **cobertura de todos os estados**, os itens de cobertura são os estados. Para atingir 100% de cobertura de todos os estados, os casos de teste devem garantir que todos os estados sejam visitados. A cobertura é medida como o número de estados visitados dividido pelo número total de estados e é expressa como uma porcentagem.

Na **cobertura de transições válidas** (também chamada de cobertura de 0-switch), os itens de cobertura são transições válidas únicas. Para atingir 100% de cobertura de transições válidas, os casos de teste devem executar todas as transições válidas. A cobertura é medida como o número de transições válidas executadas dividido pelo número total de transições válidas e é expressa como uma porcentagem.

Na **cobertura de todas as transições**, os itens de cobertura são todas as transições mostradas em uma tabela de estados. Para atingir 100% de cobertura, os casos de teste devem executar todas as transições válidas e tentar executar as transições inválidas. O teste de apenas uma transição inválida

em um único caso de teste ajuda a evitar o mascaramento de falhas, ou seja, uma situação em que um defeito impede a detecção de outro. A cobertura é medida como o número de transições válidas e inválidas executadas ou que tentaram ser cobertas pelos casos de teste executados, dividido pelo número total de transições válidas e inválidas, e é expressa como uma porcentagem.

A cobertura de todos os estados é mais fraca do que a cobertura de transições válidas, porque normalmente pode ser obtida sem a execução de todas as transições. A cobertura de transições válidas é o critério de cobertura mais amplamente utilizado. Alcançar a cobertura total de todas as transições garante tanto a cobertura total de todos os estados quanto a cobertura total de transições válidas e deve ser um requisito mínimo para softwares de missão e segurança crítica.

## 4.3 Técnicas de Teste Caixa-Branca

Devido à sua popularidade e simplicidade, esta seção se concentra em duas técnicas de teste caixa-branca relacionadas ao código:

- Teste de instruções
- Teste de ramificação

Há técnicas mais rigorosas que são usadas em alguns ambientes de segurança crítica, de missão crítica ou de alta integridade para obter uma cobertura de código mais completa. Há também técnicas de teste caixa-branca usadas em níveis de teste mais altos (p. ex., teste de API) ou usando cobertura não relacionada ao código (p. ex., cobertura de neurônios em testes de redes neurais). Essas técnicas não são discutidas neste syllabus.

### 4.3.1 Teste de Instrução e Cobertura de Instrução

No teste de instrução, os itens de cobertura são as instruções executáveis. O objetivo é projetar casos de teste que exercitem as instruções no código até que um nível aceitável de cobertura seja alcançado. A cobertura é medida como o número de instruções exercidas pelos casos de teste dividido pelo número total de instruções executáveis no código, e é expressa como uma porcentagem.

Quando a cobertura de 100% das instruções é alcançada, está garantido que todas as instruções executáveis no código tenham sido executadas pelo menos uma vez. Em particular, isso significa que cada instrução com um defeito será executada, o que pode causar uma falha que demonstre a presença do defeito. No entanto, a execução de uma instrução com um caso de teste não detectará defeitos em todos os casos. Por exemplo, pode não detectar defeitos que dependem de dados (p. ex., uma divisão por zero que só falha quando um denominador é definido como zero). Além disso, a cobertura de 100% de instrução não garante que toda a lógica de decisão tenha sido testada, pois, por exemplo, ela pode não executar todas as decisões do código (ver capítulo 4.3.2).

### 4.3.2 Teste de Ramificação e Cobertura de Ramificação

Uma ramificação é uma transferência de controle entre dois nós no gráfico do fluxo de controle, que mostra as possíveis sequências nas quais as instruções do código-fonte são executadas no objeto de

teste. Cada transferência de controle pode ser incondicional (ou seja, código em linha reta) ou condicional (ou seja, um resultado de decisão).

No teste de ramificação, os itens de cobertura são ramificações e o objetivo é modelar casos de teste para executar ramificações no código até que um nível aceitável de cobertura seja alcançado. A cobertura é medida como o número de ramificações exercidas pelos casos de teste dividido pelo número total de ramificações, e é expressa como uma porcentagem.

Quando se atinge 100% de cobertura de ramificação, todas as ramificações do código, incondicionais e condicionais, são exercidas por casos de teste. As ramificações condicionais normalmente correspondem a um resultado verdadeiro ou falso de uma decisão "*if...then*", um resultado de uma instrução "*switch/case*" ou uma decisão de sair ou continuar em um "*loop*". No entanto, e execução de uma ramificação com um caso de teste não detectará defeitos em todos os casos. Por exemplo, ele pode não detectar defeitos que exijam a execução de um caminho específico em um código.

A cobertura de ramificação substitui a cobertura de instrução. Isso significa que qualquer conjunto de casos de teste que atinja 100% de cobertura de ramificação também atinge 100% de cobertura de instrução (mas não vice-versa).

### 4.3.3 O valor do Teste Caixa-Branca

Um ponto forte fundamental que todas as técnicas de caixa-branca compartilham é que toda a implementação do software é levada em conta durante o teste, o que facilita a detecção de defeitos mesmo quando a especificação do software é vaga, desatualizada ou incompleta. Um ponto fraco correspondente é que, se o software não implementar um ou mais requisitos, o teste caixa-branca pode não detectar os defeitos resultantes ausentes (Watson 1996).

As técnicas de caixa-branca podem ser usadas em testes estáticos (p. ex., durante execuções secas de código). Elas são adequadas para revisar códigos que ainda não estão prontos para execução (Hetzel 1988), bem como pseudocódigo e outras lógicas de alto nível ou top-down que podem ser modeladas com um gráfico de fluxo de controle.

Realizar somente testes de caixa-preta não fornece uma medida da cobertura real do código. As medidas de cobertura de caixa-branca fornecem uma medição objetiva da cobertura e fornecem as informações necessárias para permitir que testes adicionais sejam gerados para aumentar essa cobertura e, consequentemente, aumentar a confiança no código.

## 4.4 Técnicas de Teste Baseadas na Experiência

As técnicas de teste baseadas na experiência comumente usadas e discutidas nas seções a seguir são:

- Suposição de erro
- Teste exploratório
- Teste baseado em lista de verificação

### 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;
- Os tipos de erros que os desenvolvedores tendem a cometer e os tipos de defeitos que resultam desses erros;
- Os tipos de falhas que ocorreram em outros aplicativos semelhantes.

Em geral, os erros, defeitos e falhas podem estar relacionados a: entrada (p. ex., entrada correta não aceita, parâmetros errados ou ausentes), saída (p. ex., formato errado, resultado errado), lógica (p. ex., casos ausentes, operador errado), cálculo (p. ex., operando incorreto, cálculo errado), interfaces (p. ex., incompatibilidade de parâmetros, tipos incompatíveis) ou dados (p. ex., inicialização incorreta, tipo errado).

Os ataques a falhas são uma abordagem metódica para a implementação da suposição de erro. Essa técnica exige que o Testador crie ou adquira uma lista de possíveis erros, defeitos e falhas e modele testes que identifiquem os defeitos associados aos erros, exponham os defeitos ou causem as falhas. Essas listas podem ser criadas com base na experiência, nos dados de defeitos e falhas ou no conhecimento comum sobre o motivo pelo qual o software falha.

Consulte (Whittaker 2002, Whittaker 2003, Andrews 2006) para obter mais informações sobre suposição de erro e ataques de falhas.

### 4.4.2 Testes Exploratórios

No teste exploratório, os testes são modelados, executados e avaliados simultaneamente enquanto o Testador aprende sobre o objeto de teste. O teste é usado para aprender mais sobre o objeto de teste, para explorá-lo mais profundamente com testes focados e para criar testes para áreas não testadas.

Às vezes, o teste exploratório é estruturado e realizado usando testes baseados em sessões. Em uma abordagem baseada em sessões, o teste exploratório é realizado dentro de um período definido. O Testador usa uma carta de teste contendo os objetivos do teste para orientar o teste. A sessão de teste geralmente é seguida de uma reunião de informação que envolve uma discussão entre o Testador e stakeholders interessados nos resultados da sessão de teste. Nessa abordagem, os objetivos do teste podem ser tratados como condições de teste de alto nível. Os itens de cobertura são identificados e executados durante a sessão de teste. O Testador pode usar fichas de sessão de teste para documentar as etapas seguidas e as descobertas feitas.

Os testes exploratórios são úteis quando há poucas especificações ou especificações inadequadas, ou quando há uma pressão significativa de tempo sobre os testes. O teste exploratório também é útil para complementar outras técnicas de teste mais formais. O teste exploratório será mais eficaz se o Testador for experiente, tiver conhecimento do domínio e possuir um nível elevado de habilidades essenciais, como habilidades analíticas, curiosidade e criatividade (ver seção 1.5.1).

O teste exploratório pode incorporar o uso de outras técnicas de teste (p. ex., particionamento de equivalência). Mais informações sobre testes exploratórios podem ser encontradas em (Kaner 1999, Whittaker 2009, Hendrickson 2013).

### 4.4.3 Testes baseados em Lista de Verificação

Nos testes baseados em lista de verificação, um Testador projeta, implementa e executa testes para cobrir as condições de teste de uma lista de verificação. As listas de verificação podem ser criadas com base na experiência, no conhecimento sobre o que é importante para o usuário ou na compreensão de por que e como o software falha. As listas de verificação não devem conter itens que possam ser verificados automaticamente, itens mais adequados como critérios de entrada/saída ou itens muito gerais (Brykczynski 1999).

Os itens da lista de verificação geralmente são formulados na forma de uma pergunta. Deve ser possível verificar cada item separadamente e diretamente. Esses itens podem se referir a requisitos, propriedades da interface gráfica, características de qualidade ou outras formas de condições de teste. As listas de verificação podem ser criadas para dar suporte a vários tipos de teste, incluindo testes funcionais e não funcionais (p. ex., 10 heurísticas para testes de usabilidade (Nielsen 1994)).

Algumas entradas da lista de verificação podem se tornar gradualmente menos eficazes com o tempo, pois os desenvolvedores aprenderão a evitar os mesmos erros. Também pode ser necessário adicionar novas entradas para refletir defeitos de alta gravidade encontrados recentemente. Portanto, as listas de verificação devem ser atualizadas regularmente com base na análise de defeitos. Entretanto, deve-se tomar cuidado para evitar que a lista de verificação fique muito longa (Gawande 2009).

Na ausência de casos de teste detalhados, os testes baseados em listas de verificação podem fornecer diretrizes e algum grau de consistência para os testes. Se as listas de verificação forem de alto nível, é provável que ocorra alguma variabilidade no teste real, resultando em uma cobertura potencialmente maior, mas com menos repetibilidade.

## 4.5 Abordagens de Teste Baseadas na Colaboração

Cada uma das técnicas mencionadas acima (ver seções 4.2, 4.3, 4.4) tem um objetivo específico com relação à detecção de defeitos. As abordagens baseadas na colaboração, por outro lado, também se concentram em evitar defeitos por meio de colaboração e comunicação.

### 4.5.1 Escrita colaborativa de histórias de usuários

Uma história de usuário representa um recurso que será valioso para um usuário ou comprador de um sistema ou software. As histórias de usuários têm três aspectos críticos (Jeffries 2000), chamados em conjunto de "3C":

- *Cartão*: o meio onde se descreve uma história de usuário (p. ex., um cartão de registro, uma entrada em um quadro eletrônico);
- *Conversação*: a explicação de como o software será usado (pode ser documentado ou verbal)
- *Confirmação*: os critérios de aceite (ver seção 4.5.2)

O formato mais comum de uma história de usuário é "Como [ator ou persona], quero [meta a ser cumprida], para que eu possa [valor de negócio resultante para a função]", seguido dos critérios de aceite.

A autoria colaborativa da história do usuário pode usar técnicas como brainstorming e mapeamento mental. A colaboração permite que a equipe obtenha uma visão compartilhada do que deve ser entregue, levando em conta três perspectivas: negócios, desenvolvimento e testes.

Boas histórias de usuários devem ser: Independentes, Negociáveis, Valiosas, Estimáveis, pequenas (Small) e Testáveis (INVEST). Se um stakeholder não souber como testar uma história de usuário, isso pode indicar que a história de usuário não está suficientemente clara, ou que não reflete algo valioso para ele, ou que o stakeholder precisa apenas de ajuda para testar (Wake 2003).

### 4.5.2 Critérios de Aceite

Os critérios de aceite de uma história de usuário são as condições que a implementação dela deve atender para ser aceita pelos stakeholders. Sob essa perspectiva, os critérios de aceite podem ser vistos como as condições de teste que devem ser executadas pelos testes. Os critérios de aceite geralmente são resultado da Conversação (ver seção 4.5.1).

Os critérios de aceite são usados para:

- Definir o escopo da história do usuário;
- Chegar a um consenso entre os stakeholders;
- Descrever os cenários positivos e negativos;
- Servir como base para o teste de aceite da história do usuário (ver seção 4.5.3);
- Permitir planejamento e estimativa precisos.

Há várias maneiras de escrever critérios de aceite para uma história de usuário. Os dois formatos mais comuns são:

- Orientado a cenários (p. ex., formato Given/When/Then usado no BDD, ver seção 2.1.3);
- Orientado por regras (p. ex., lista de pontos de verificação ou forma tabulada de mapeamento de entrada e saída);

A maioria dos critérios de aceite podem ser documentadas em um desses dois formatos. No entanto, a equipe pode usar outro formato personalizado, desde que os critérios de aceite sejam bem definidos e não ambíguos.

### 4.5.3 Desenvolvimento Orientado por Teste de Aceite (ATDD)

O ATDD é uma abordagem que prioriza o teste (ver seção 2.1.3). Os casos de teste são criados antes da implementação da história do usuário. Eles são criados por membros da equipe com diferentes perspectivas, por exemplo, clientes, desenvolvedores e Testadores (Adzic 2009), podendo ser executados de forma manual ou automatizada.

A primeira etapa é um workshop de especificação em que a história do usuário e, se ainda não estiver definida, seus critérios de aceite são analisados, discutidos e escritos pelos membros da equipe. Lacunas, ambiguidades ou defeitos na história do usuário são resolvidos durante esse processo. A

próxima etapa é a criação dos casos de teste. Isso pode ser feito pela equipe como um todo ou individualmente pelo Testador. Os casos de teste são baseados nos critérios de aceite e podem ser vistos como exemplos de como o software funciona. Isso ajudará a equipe a implementar corretamente a história do usuário.

Como exemplos e testes são a mesma coisa, esses termos são frequentemente usados de forma intercambiável. Durante o projeto do teste, as técnicas de teste descritas nas seções 4.2, 4.3 e 4.4 podem ser aplicadas.

Normalmente, os primeiros casos de teste são positivos, confirmando o comportamento correto, sem exceções ou condições de erro, e compreendendo a sequência de atividades executadas se tudo ocorrer conforme o esperado. Depois que os casos de teste positivos forem concluídos, a equipe deverá realizar testes negativos. Por fim, a equipe deve cobrir também as características de qualidade não funcionais (p. ex., eficiência de performance, usabilidade). Os casos de teste devem ser expressos de forma que sejam compreensíveis para os stakeholders. Normalmente, os casos de teste contêm frases em linguagem natural que envolvem as pré-condições necessárias (se houver), as entradas e as pós-condições.

Os casos de teste devem abranger todas as características da história do usuário e não devem ir além dela. Entretanto, os critérios de aceite podem detalhar alguns dos problemas descritos na história do usuário. Além disso, dois casos de teste não devem descrever as mesmas características da história do usuário.

Quando capturados em um formato compatível com uma estrutura de automação de testes, os desenvolvedores podem automatizar os casos de teste escrevendo o código de suporte à medida que implementam o recurso descrito por uma história de usuário. Os testes de aceite tornam-se, então, requisitos executáveis.

# 5 Gerenciamento das Atividades de Teste (335 min)

## Palavras-chave

gerenciamento de defeitos, relatório de defeitos, critérios de entrada, critérios de saída, risco de produto, risco de projeto, risco, análise de risco, avaliação de risco, controle de risco, identificação de risco, nível de risco, gerenciamento de risco, mitigação de risco, monitoramento de risco, teste baseado em risco, abordagem de teste, relatório de conclusão de teste, controle de teste, monitoramento de teste, plano de teste, planejamento de teste, relatório de progresso de teste, pirâmide de teste, quadrantes de teste

## Objetivos de aprendizagem

### 5.1 Planejamento de Teste

FL-5.1.1 (K2) Exemplificar a finalidade e o conteúdo de um plano de teste.

FL-5.1.2 (K1) Reconhecer como um Testador agrega valor ao planejamento de iteração e lançamento.

FL-5.1.3 (K2) Comparar e contrastar critérios de entrada e critérios de saída.

FL-5.1.4 (K3) Usar técnicas de estimativa para calcular o esforço necessário de teste.

FL-5.1.5 (K3) Aplicar a priorização de casos de teste.

FL-5.1.6 (K1) Relembrar os conceitos da pirâmide de teste.

FL-5.1.7 (K2) Resumir os quadrantes de teste e suas relações com os níveis e tipos de teste.

### 5.2 Gerenciamento de Risco

FL-5.2.1 (K1) Identificar o nível de risco usando a probabilidade de risco e o impacto do risco.

FL-5.2.2 (K2) Distinguir entre riscos de projeto e riscos de produto.

FL-5.2.3 (K2) Explicar como a análise de risco do produto pode influenciar o rigor e o escopo dos testes.

FL-5.2.4 (K2) Explicar quais medidas podem ser tomadas em resposta aos riscos analisados do produto.

### 5.3 Monitoramento, Controle e Conclusão do Teste

FL-5.3.1 (K1) Métricas de recuperação usadas para testes.

FL-5.3.2 (K2) Resumir as finalidades, o conteúdo e o público dos relatórios de teste.

FL-5.3.3 (K2) Exemplificar como comunicar o status do teste.

### 5.4 Gerenciamento de Configuração (CM)

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

### 5.5 Gerenciamento de Defeitos

FL-5.5.1 (K3) Preparar um relatório de defeitos.

# 5.1 Planejamento de Teste

## 5.1.1 Objetivo e conteúdo de um Plano de Teste

Um plano de teste descreve os objetivos, os recursos e os processos de um projeto de teste. Um plano de teste:

- Documenta os meios e o cronograma para atingir os objetivos do teste;
- Ajuda a garantir que as atividades de teste realizadas atenderão aos critérios estabelecidos;
- Serve como um meio de comunicação com os membros da equipe e outros stakeholders;
- Demonstra que os testes seguirão a política e a estratégia de testes existentes (ou explica por que os testes se desviarão delas);

O planejamento de testes orienta o pensamento dos Testadores e os força a enfrentar os desafios futuros relacionados a riscos, cronogramas, pessoas, ferramentas, custos, esforços etc. O processo de preparação de um plano de teste é uma maneira útil de pensar nos esforços necessários para atingir os objetivos do projeto de teste.

O conteúdo típico de um plano de teste inclui:

- Contexto do teste (p. ex., escopo, objetivos do teste, restrições, base do teste);
- Premissas e restrições do projeto de teste;
- Stakeholders (p. ex., funções, responsabilidades, relevância para os testes, necessidades de contratação e treinamento);
- Comunicação (p. ex., formas e frequência de comunicação, modelos de documentação);
- Registro de riscos (p. ex., riscos do produto, riscos do projeto);
- Abordagem de teste (p. ex., níveis de teste, tipos de teste, técnicas de teste, produtos de teste, critérios de entrada e de saída, independência do teste, métricas a serem coletadas, requisitos de dados de teste, requisitos de ambiente de teste, desvios da política de teste organizacional e da estratégia de teste);
- Orçamento e cronograma.

Mais detalhes sobre o plano de teste e seu conteúdo podem ser encontrados na norma Isso/IEC/IEEE 29119-3.

## 5.1.2 Contribuição do Testador para o planejamento de iteração e liberação

Nos SDLCs iterativos, normalmente ocorrem dois tipos de planejamento: planejamento de liberação e planejamento de iteração.

O planejamento da liberação prevê o lançamento de um produto, define e redefine o backlog do produto e pode envolver o refinamento de histórias de usuários maiores em um conjunto de histórias de usuários menores. Ele também serve como base para a abordagem e o plano de teste em todas as iterações. Os Testadores envolvidos no planejamento da liberação participam da escrita das histórias de usuário e critérios de aceite testáveis (ver seção 4.5), participam das análises de risco do projeto e da qualidade (ver seção 5.2), estimam o esforço de teste associado às histórias de usuário (ver seção 5.1.4), determinam a abordagem de teste e planejam o teste para a versão.

O planejamento da iteração prevê o fim de uma única iteração e se preocupa com o backlog da iteração. Os Testadores envolvidos participam da análise de risco detalhada das histórias de usuários, determinam a testabilidade das histórias, dividem-nas em tarefas (especialmente tarefas de teste), estimam o esforço de teste para todas as tarefas de teste e identificam e refinam os aspectos funcionais e não funcionais do objeto de teste.

### 5.1.3 Critérios de Entrada e Critérios de Saída

Os critérios de entrada definem as condições prévias para a realização de uma determinada atividade. Se os critérios de entrada não forem atendidos, é provável que a atividade seja mais difícil, demorada, cara e arriscada. Os critérios de saída definem o que deve ser alcançado para que uma atividade seja declarada concluída. Os critérios de entrada e os critérios de saída devem ser definidos para cada nível de teste e serão diferentes com base nos objetivos do teste.

Os critérios de entrada típicos incluem: disponibilidade de recursos (p. ex., pessoas, ferramentas, ambientes, dados de teste, orçamento, tempo), disponibilidade de material de teste (p. ex., base de teste, requisitos testáveis, histórias de usuários, casos de teste) e nível de qualidade inicial de um objeto de teste (p. ex., todos os testes de fumaça foram aprovados).

Os critérios de saída típicos incluem: medidas de precisão (p. ex., nível de cobertura alcançado, número de defeitos não resolvidos, densidade de defeitos, número de casos de teste com falha) e critérios de conclusão (p. ex., testes planejados foram executados, testes estáticos foram realizados, todos os defeitos encontrados foram relatados, todos os testes de regressão foram automatizados).

O esgotamento do tempo ou do orçamento também pode ser visto como um critério de saída válido. Mesmo sem que outros critérios de saída sejam atendidos, pode ser aceitável encerrar os testes nessas circunstâncias, se os stakeholders tiverem analisado e aceitado o risco de entrar em operação sem mais testes.

No desenvolvimento ágil de software, os critérios de saída são geralmente chamados de Definição de Feito (DoD - Definition of Done), definindo as métricas objetivas da equipe para um item liberável. Os critérios de entrada que uma história de usuário deve cumprir para iniciar as atividades de desenvolvimento e/ou teste são chamados de Definição de Pronto (DoR - Definition of Ready).

### 5.1.4 Técnicas de estimativa

A estimativa do esforço de teste envolve a previsão da quantidade de trabalho relacionado ao teste necessária para atender aos objetivos de um projeto de teste. É importante deixar claro para os stakeholders que a estimativa é baseada em várias suposições e está sempre sujeita a erros. A estimativa para tarefas pequenas geralmente é mais precisa do que para as grandes. Portanto, ao estimar uma tarefa grande, ela pode ser decomposta em um conjunto de tarefas menores que, por sua vez, podem ser estimadas.

Neste syllabus, são descritas as quatro técnicas de estimativa a seguir.

**Estimativa baseada em índices**. Nessa técnica baseada em métricas, os números são coletados de projetos anteriores dentro da organização, o que torna possível derivar “indicadores padrão" para projetos semelhantes. As proporções dos próprios projetos de uma organização (p. ex., extraídas de dados históricos) geralmente são a melhor fonte a ser usada no processo de estimativa. Esses

indicadores padrão podem então ser usados para estimar o esforço de teste para o novo projeto. Por exemplo, se no projeto anterior a proporção de esforço de desenvolvimento para teste foi de 3:2 e no projeto atual espera-se que o esforço de desenvolvimento seja de 600 homens/hora, o esforço de teste pode ser estimado em 400 homens/hora.

**Extrapolação**. Nessa técnica baseada em métricas, as medições são feitas o mais cedo possível no projeto atual para coletar os dados. Com observações suficientes, o esforço necessário para o trabalho restante pode ser aproximado por meio da extrapolação desses dados (geralmente aplicando um modelo matemático). Esse método é muito adequado em SDLCs iterativos. Por exemplo, a equipe pode extrapolar o esforço de teste na próxima iteração como o esforço médio das três últimas iterações.

**Wideband Delphi**. Essa técnica iterativa é baseada em especialistas que fazem estimativas baseadas na experiência. Cada especialista, isoladamente, estima o esforço. Os resultados são coletados e, se houver desvios que estejam fora da faixa dos limites acordados, os especialistas discutem suas estimativas atuais. Em seguida, pede-se a cada especialista que faça uma nova estimativa com base nesse feedback, novamente de forma isolada. Esse processo é repetido até que se chegue a um consenso. O Planning Poker é uma variante do Wideband Delphi, comumente utilizado no desenvolvimento ágil de software. No Planning Poker, as estimativas geralmente são feitas usando cartões com números que representam o tamanho do esforço.

**Estimativa de três pontos**. Nessa técnica baseada em especialistas, três estimativas são feitas pelos especialistas: a estimativa mais otimista (o), a estimativa mais provável (m) e a estimativa mais pessimista (p). A estimativa final (E) é a média aritmética ponderada dessas estimativas. Na versão mais popular dessa técnica, a estimativa é calculada como:

$$E = \frac{o + 4m + p}{6}$$

A vantagem dessa técnica é que ela permite que os especialistas calculem o erro de medição:

$$SD = \frac{(p - o)}{6}$$

Por exemplo, se as estimativas (em homens/hora) forem: o=6, m=9 e p=18, então a estimativa final é de 10±2 homens/hora (ou seja, entre 8 e 12 homens/hora), porque:

$$E = \frac{6+36+18}{6} = 10 \quad \text{e} \quad SD = \frac{(18-6)}{6} = 2$$

Consulte (Kan 2003, Koomen 2006, Westfall 2009) para conhecer essas e muitas outras técnicas de estimativa de teste.

### 5.1.5 Priorização de casos de teste

Depois que os casos de teste e os procedimentos de teste são especificados e reunidos em conjuntos de testes, esses conjuntos de testes podem ser organizados em um cronograma de execução de testes que define a ordem em que devem ser executados. Ao priorizar os casos de teste, diferentes

fatores podem ser levados em conta. As estratégias de priorização de casos de teste mais comumente usadas são as seguintes:

- Priorização baseada em risco, em que a ordem de execução do teste é baseada nos resultados da análise de risco (ver seção 5.2.3). Os casos de teste que abrangem os riscos mais importantes são executados primeiro.
- Priorização baseada em cobertura, em que a ordem de execução do teste é baseada na cobertura (p. ex., cobertura de instrução). Os casos de teste que atingem a maior cobertura são executados primeiro. Em outra variante, chamada de priorização de cobertura adicional, o caso de teste que atinge a maior cobertura é executado primeiro; cada caso de teste subsequente é aquele que atinge a maior cobertura adicional.
- Priorização baseada em requisitos, em que a ordem de execução do teste é baseada nas prioridades dos requisitos rastreados até os casos de teste correspondentes. As prioridades dos requisitos são definidas pelos stakeholders. Os casos de teste relacionados aos requisitos mais importantes são executados primeiro.

Preferencialmente, os casos de teste seriam ordenados para execução com base em seus níveis de prioridade, usando, por exemplo, uma das estratégias de priorização mencionadas anteriormente. Entretanto, essa prática pode não funcionar se os casos de teste ou os recursos que estão sendo testados tiverem dependências. Se um caso de teste com prioridade mais alta for dependente de um caso de teste com prioridade mais baixa, o caso de teste de prioridade mais baixa deverá ser executado primeiro.

A ordem de execução do teste também deve levar em conta a disponibilidade de recursos. Por exemplo, as ferramentas de teste necessárias, os ambientes de teste ou as pessoas que podem estar disponíveis apenas em um período específico.

### 5.1.6 Pirâmide de Teste

A pirâmide de teste é um modelo que mostra que diferentes testes podem ter diferentes granularidades. O modelo da pirâmide de testes apoia a equipe na automação de testes e na alocação de esforço de teste, mostrando que diferentes objetivos são apoiados por diferentes níveis de automação de testes. As camadas da pirâmide representam grupos de testes. Quanto mais alta a camada, menor a granularidade do teste, o isolamento do teste e o tempo de execução do teste. Os testes na camada inferior são pequenos, isolados, rápidos e verificam uma pequena parte da funcionalidade, portanto, geralmente são necessários muitos deles para obter uma cobertura razoável. A camada superior representa testes complexos, de alto nível e de ponta a ponta. Esses testes de alto nível geralmente são mais lentos do que os testes das camadas inferiores e, normalmente, verificam uma grande parte da funcionalidade, de modo que, em geral, são necessários apenas alguns deles para obter uma cobertura razoável. O número e a nomenclatura das camadas podem ser diferentes. Por exemplo, o modelo original da pirâmide de testes (Cohn 2009) define três camada“: "testes de unidade“, "testes de serviço" e "testes de interface do usuário". Outro modelo popular define testes de unidade (componente), testes de integração (integração de componente) e testes de ponta a ponta. Outros níveis de teste (ver seção 2.2.1) também podem ser usados.

### 5.1.7 Quadrantes de Teste

Os quadrantes de teste, definidos por Brian Marick (Marick 2003, Crispin 2008), agrupam os níveis de teste com os tipos de teste, as atividades, as técnicas de teste e os produtos de trabalho apropriados no desenvolvimento ágil de software. O modelo ajuda o gerenciamento de testes a visualizá-los para garantir que todos os tipos e níveis de teste apropriados sejam incluídos no SDLC e a entender que alguns tipos de teste são mais relevantes para determinados níveis de teste do que outros. Esse modelo também fornece uma maneira de diferenciar e descrever os tipos de testes para todos os stakeholders, incluindo desenvolvedores, Testadores e representantes do negócio.

Nesse modelo, os testes podem ser voltados para os negócios ou para a tecnologia. Os testes também podem apoiar a equipe (ou seja, orientar o desenvolvimento) ou criticar o produto (ou seja, medir seu comportamento em relação às expectativas). A combinação desses dois pontos de vista determina os quatro quadrantes:

- **Quadrante Q1** (voltado para a tecnologia, suporte à equipe). Esse quadrante contém testes de componentes e de integração de componentes. Esses testes devem ser automatizados e incluídos no processo de CI.
- **Quadrante Q2** (voltado para os negócios, suporte à equipe). Esse quadrante contém testes funcionais, exemplos, testes de histórias de usuários, protótipos de experiência do usuário, testes de API e simulações. Esses testes verificam os critérios de aceite e podem ser manuais ou automatizados.
- **Quadrante Q3** (voltado para os negócios, crítica do produto). Esse quadrante contém testes exploratórios, testes de usabilidade e testes de aceite do usuário. Esses testes são orientados para o usuário e geralmente manuais.
- **Quadrante Q4** (voltado para a tecnologia, crítica do produto). Esse quadrante contém *smoke tests* e testes não funcionais (exceto testes de usabilidade). Esses testes geralmente são automatizados.

## 5.2 Gerenciamento de Risco

As organizações enfrentam muitos fatores internos e externos que tornam incerto se e quando elas atingirão seus objetivos (ISO31000). O gerenciamento de riscos permite que as organizações aumentem a probabilidade de atingir seus objetivos, melhorem a qualidade de seus produtos e aumentem a confiança dos stakeholders.

As principais atividades de gerenciamento de riscos são:

- Análise de risco (que consiste na identificação e avaliação de risco. (ver seção 5.2.3)
- Controle de riscos (que consiste na mitigação e monitoramento de riscos. (ver seção 5.2.4)

A abordagem de teste, na qual as atividades de teste são selecionadas, priorizadas e gerenciadas com base na análise e no controle de riscos, é chamada de teste baseado em riscos.

### 5.2.1 Definição de Risco e Atributos do Risco

Risco é um possível evento, perigo, ameaça ou situação cuja ocorrência causa um efeito adverso. Um risco pode ser caracterizado por dois fatores:

- **Probabilidade**: a probabilidade de ocorrência do risco (maior que zero e menor que um)
- **Impacto** (dano): as consequências dessa ocorrência

Esses dois fatores expressam o nível de risco, que é uma medida do risco. Quanto maior o nível de risco, mais importante é o seu tratamento.

### 5.2.2 Riscos do projeto e riscos do produto

No teste de software, geralmente nos preocupamos com dois tipos de riscos: riscos de projeto e riscos de produto.

Os **Riscos do Projeto** estão relacionados ao gerenciamento e ao controle do projeto. Os riscos do projeto incluem:

- *Problemas organizacionais* (p. ex., atrasos na entrega de produtos de trabalho, estimativas imprecisas, redução de custos)
- *Problemas de pessoal* (p. ex., habilidades insuficientes, conflitos, problemas de comunicação, falta de pessoal)
- *Problemas técnicos* (p. ex., desvios de escopo, suporte insuficiente a ferramentas)
- *Problemas com fornecedores* (p. ex., falha na entrega de terceiros, falência da empresa de apoio)

Os riscos do projeto, quando ocorrem, podem ter um impacto no cronograma, no orçamento ou no escopo do projeto, o que afeta a capacidade do projeto de atingir seus objetivos.

Os **Riscos do Produto** estão relacionados às características de qualidade do produto (p. ex., descritas no modelo de qualidade ISO 25010). Exemplos de riscos do produto incluem: funcionalidade ausente ou errada, cálculos incorretos, erros de tempo de execução, arquitetura ruim, algoritmos ineficientes, tempo de resposta inadequado, experiência ruim do usuário, vulnerabilidades de segurança. Os riscos do produto, quando ocorrem, podem resultar em várias consequências negativas, incluindo:

- Insatisfação do usuário;
- Perda de receita, confiança e reputação;
- Danos a terceiros;
- Altos custos de manutenção, e sobrecarga do helpdesk;
- Penalidades criminais;
- Em casos extremos, danos físicos, lesões ou até mesmo a morte.

### 5.2.3 Análise de Risco do Produto

Do ponto de vista do teste, o objetivo da análise de risco do produto é proporcionar uma conscientização do risco do produto para concentrar o esforço de teste de forma a minimizar o nível residual de risco do produto. O ideal é que a análise comece no início do SDLC.

A análise de risco do produto consiste na identificação e na avaliação de riscos. A identificação de riscos consiste em gerar uma lista abrangente de riscos. Os stakeholders podem identificá-los usando várias técnicas e ferramentas, como, por exemplo, brainstorming, workshops, entrevistas ou diagramas de causa e efeito. A avaliação de riscos envolve: a categorização dos riscos identificados, a determinação da probabilidade, do impacto e do nível dos riscos, a priorização e a proposta de maneiras de lidar com eles. A categorização ajuda a atribuir ações de mitigação, porque geralmente os riscos que se enquadram na mesma categoria podem ser mitigados usando uma abordagem semelhante.

A avaliação de riscos pode usar uma abordagem quantitativa ou qualitativa, ou uma combinação delas. Na abordagem quantitativa, o nível de risco é calculado como a multiplicação da probabilidade de risco e do impacto do risco. Na abordagem qualitativa, o nível de risco pode ser determinado usando uma matriz de risco.

A análise de risco do produto pode influenciar a minúcia e o escopo dos testes. Seus resultados são usados para:

- Determinar o escopo dos testes a serem realizados;
- Determinar os níveis de teste específicos e propor os tipos de teste a serem realizados;
- Determinar as técnicas de teste a serem empregadas e a cobertura a ser alcançada;
- Estimar o esforço de teste necessário para cada tarefa;
- Priorizar os testes em uma tentativa de encontrar os defeitos críticos o mais cedo possível;
- Determinar se outras atividades, além dos testes, podem ser empregadas para reduzir o risco.

### 5.2.4 Controle de Risco do Produto

O controle de riscos do produto compreende todas as medidas tomadas em resposta aos riscos identificados e avaliados do produto. O controle de riscos do produto consiste na mitigação e no monitoramento dos riscos. Esta mitigação envolve a implementação das ações propostas na avaliação de riscos para reduzir o nível de risco. O objetivo do monitoramento de riscos é garantir que as ações de mitigação sejam eficazes, obtenham mais informações para melhorar a avaliação de riscos e identificar riscos emergentes.

Com relação ao controle de riscos do produto, uma vez que um risco tenha sido analisado, várias opções de resposta ao risco são possíveis, por exemplo, mitigação do risco por meio de testes, aceite do risco, transferência do risco ou plano de contingência (Veenendaal 2012). As ações que podem ser tomadas para mitigar os riscos do produto por meio de testes são as seguintes:

- Selecionar os Testadores com o nível de experiência e habilidade adequado para um determinado tipo de risco;
- Aplicar um nível adequado de independência de testes;
- Conduzir revisões e realizar análises estáticas;

- Aplicar as técnicas de teste e os níveis de cobertura adequados;
- Aplicar os tipos de teste apropriados que abordam as características de qualidade afetadas;
- Realizar testes dinâmicos, incluindo testes de regressão.

## 5.3 Monitoramento, Controle e Conclusão do Teste

O monitoramento de testes está relacionado à coleta de informações sobre os testes. Essas informações são usadas para avaliar o progresso do teste e para medir se os critérios de saída do teste ou as tarefas de teste associadas aos critérios de saída foram atendidos, como o cumprimento das metas de cobertura dos riscos, requisitos ou critérios de aceite do produto.

O controle de testes usa as informações do monitoramento de testes para fornecer, na forma de diretrizes de controle, orientação e as ações corretivas necessárias para obter testes mais eficazes e eficientes. Exemplos de diretrizes de controle incluem:

- Repriorizar os testes quando um risco identificado se torna um problema;
- Reavaliar se um item de teste atende aos critérios de entrada ou saída devido ao retrabalho;
- Ajustar o cronograma de testes para lidar com um atraso na entrega do ambiente de teste;
- Adicionar novos recursos quando e onde forem necessário.

A conclusão do teste coleta dados de atividades de teste concluídas para consolidar a experiência, o material de teste e qualquer outra informação relevante. As atividades de conclusão do teste ocorrem em marcos do projeto, como quando um nível de teste é concluído, uma iteração Ágil é finalizada, um projeto de teste é concluído (ou cancelado), um sistema de software é lançado ou uma versão de manutenção é concluída.

### 5.3.1 Métricas usadas em testes

As métricas de teste são reunidas para mostrar o progresso em relação ao cronograma e orçamento planejados, a qualidade atual do objeto de teste e a eficácia das atividades de teste em relação aos objetivos ou a uma meta de iteração. O monitoramento do teste reúne uma variedade de métricas para apoiar o controle e a conclusão do teste.

As métricas de teste comuns incluem:

- *Métricas de progresso do projeto* (p. ex., conclusão de tarefas, uso de recursos, esforço de teste);
- *Métricas de progresso do teste* (p. ex., progresso da implementação do caso de teste, progresso da preparação do ambiente de teste, número de casos de teste executados/não executados, aprovados/fracassados, tempo de execução do teste);
- *Métricas de qualidade do produto* (p. ex., disponibilidade, tempo de resposta, tempo médio até a falha);
- *Métricas de defeitos* (p. ex., número e prioridades de defeitos encontrados/corrigidos, densidade de defeitos, porcentagem de detecção de defeitos);
- *Métricas de risco* (p. ex., nível de risco residual);
- *Métricas de cobertura* (p. ex., cobertura de requisitos, cobertura de código);
- *Métricas de custo* (p. ex., custo de teste, custo organizacional de qualidade).

## 5.3.2 Relatórios de Teste: objetivo, conteúdo e público-alvo

Os relatórios de teste resumem e comunicam as informações do teste durante e após o teste. Os relatórios de progresso do teste dão suporte ao controle contínuo do teste e devem fornecer informações suficientes para fazer modificações no cronograma, nos recursos ou no plano de teste, quando essas alterações forem necessárias devido a desvios do plano ou mudanças nas circunstâncias. Os relatórios de conclusão do teste resumem um estágio específico do teste (p. ex., nível de teste, ciclo de teste, iteração) e podem fornecer informações para testes subsequentes.

Durante o monitoramento e o controle do teste, a equipe de teste gera relatórios de progresso do teste para os stakeholders, a fim de mantê-los informados. Os relatórios de progresso do teste geralmente são gerados regularmente (p. ex., diariamente, semanalmente etc.) e incluem:

- Período de teste;
- Progresso do teste (p. ex., adiantado ou atrasado), incluindo quaisquer desvios notáveis;
- Impedimentos para testes e suas soluções;
- Métricas de teste (ver seção 5.3.1 para obter exemplos);
- Riscos novos e alterados durante o período de teste;
- Testes planejados para o próximo período.

Um relatório de conclusão de teste é preparado durante a conclusão do teste, quando um projeto, nível de teste ou tipo de teste está concluído e quando, idealmente, seus critérios de saída foram atendidos. Esse relatório usa relatórios de progresso do teste e outros dados. Os relatórios típicos de conclusão de teste incluem:

- Resumo do teste;
- Testes e avaliação da qualidade do produto com base no plano de teste original (ou seja, objetivos de teste e critérios de saída);
- Desvios do plano de teste (p. ex., diferenças em relação ao cronograma, à duração e ao esforço planejados);
- Impedimentos e soluções alternativas para o teste;
- Métricas de teste baseadas em relatórios de progresso de teste;
- Riscos não mitigados, defeitos não corrigidos;
- Lições aprendidas que são relevantes para o teste;

Públicos diferentes exigem informações diferentes nos relatórios e influenciam o grau de formalidade e a frequência dos relatórios. Os relatórios sobre o progresso dos testes para outras pessoas da mesma equipe costumam ser frequentes e informais, enquanto os relatórios sobre os testes de um projeto concluído seguem um modelo definido e ocorrem apenas uma vez.

A norma ISO/IEC/IEEE 29119-3 inclui modelos e exemplos de relatórios de progresso de teste (chamados de relatórios de status de teste) e relatórios de conclusão de teste.

## 5.3.3 Comunicação do status dos testes

O melhor meio de comunicar o status do teste varia, dependendo das preocupações com o gerenciamento de testes, das estratégias de teste da organização, dos padrões normativos ou, no caso de equipes auto-organizadas (ver seção 1.5.2), da própria equipe. As opções incluem:

- Comunicação verbal com membros da equipe e outros stakeholders;
- Painéis (p. ex., painéis de CI/CD, painéis de tarefas e gráficos de burn-down);
- Canais de comunicação eletrônica (p. ex., e-mail, bate-papo);
- Documentação on-line;
- Relatórios de testes formais (ver seção 5.3.2).

Uma ou mais dessas opções podem ser usadas. Uma comunicação mais formal pode ser mais apropriada para equipes distribuídas, em que a comunicação direta, face a face, nem sempre é possível devido à distância geográfica ou às diferenças de horário. Normalmente, diferentes stakeholders estão interessados em diferentes tipos de informações, portanto, a comunicação deve ser adaptada de acordo.

## 5.4 Gerenciamento de Configuração (CM)

Nos testes, o gerenciamento de configuração (CM - *Configuration Management*) oferece uma disciplina para identificar, controlar e rastrear produtos de trabalho, como planos de teste, estratégias de teste, condições de teste, casos de teste, scripts de teste, resultados de teste, registros de teste e relatórios de teste como itens de configuração.

Para um item de configuração complexo (p. ex., um ambiente de teste), o CM registra os itens que o compõem, seus relacionamentos e versões. Se o item de configuração for aprovado para teste, ele se tornará uma linha de base e só poderá ser alterado por meio de um processo formal de controle de alterações.

O gerenciamento de configuração mantém um registro dos itens de configuração alterados quando uma nova *baseline* é criada. É possível reverter para uma *baseline* anterior para reproduzir resultados de testes anteriores.

Para dar suporte adequado aos testes, o CM garante o seguinte:

- Todos os itens de configuração, incluindo itens de teste (partes individuais do objeto de teste), são identificados de forma exclusiva, controlados por versão, rastreados quanto a alterações e relacionados a outros itens de configuração para que a rastreabilidade possa ser mantida durante todo o processo de teste
- Todos os itens de documentação e software identificados são referenciados sem ambiguidade na documentação de teste

A integração contínua, a entrega contínua, a implantação contínua e os testes associados normalmente são implementados como parte de um pipeline de DevOps automatizado (ver seção 2.1.4), no qual a CM automatizada normalmente está incluída.

## 5.5 Gerenciamento de Defeitos

Como um dos principais objetivos do teste é encontrar defeitos, um processo de gerenciamento de defeitos estabelecido é essencial. Embora digamos "defeitos", aqui as anomalias relatadas podem se tornar defeitos reais ou outra coisa (p. ex., falso positivo, solicitação de–alteração) - isso é resolvido durante o processo de tratamento dos relatórios de defeitos. As anomalias podem ser relatadas

durante qualquer fase do SDLC e a forma depende do SDLC. No mínimo, o processo de gerenciamento de defeitos inclui um fluxo de trabalho para lidar com anomalias individuais, desde sua descoberta até seu fechamento, e regras para sua classificação. O fluxo de trabalho normalmente inclui atividades para registrar as anomalias, analisá-las, classificá-las, decidir sobre uma resposta adequada, como consertar ou manter como está e, finalmente, fechar o relatório de defeitos. O processo deve ser seguido por todas os stakeholders envolvidos. É aconselhável lidar com os defeitos dos testes estáticos (especialmente a análise estática) de maneira semelhante.

Os típicos relatórios de defeitos possuem os seguintes objetivos:

- Fornecer aos responsáveis pelo tratamento e solução dos defeitos relatados informações suficientes para resolver o problema;
- Fornecer os meios de rastrear a qualidade do produto do trabalho;
- Fornecer ideias para aprimoramento do processo de desenvolvimento e teste.

Um relatório de defeitos registrado durante um teste dinâmico normalmente possui:

- Um identificador exclusivo;
- Um título com um breve resumo da anomalia que está sendo relatada;
- A data em que a anomalia foi observada, a organização emissora e o autor, incluindo seu cargo;
- A identificação do objeto de teste e do ambiente de teste;
- O contexto do defeito (p. ex., caso de teste que está sendo executado, atividade de teste que está sendo realizada, fase do SDLC e outras informações relevantes, como a técnica de teste, a lista de verificação ou os dados de teste que estão sendo usados);
- A descrição da falha para permitir a reprodução e a resolução, incluindo as etapas que detectaram a anomalia e todos os registros de teste relevantes, despejos de banco de dados, capturas de tela ou gravações;
- Os resultados esperados e os resultados reais;
- A severidade do defeito (grau de impacto) nos interesses dos stakeholders ou nos requisitos;
- A prioridade de correção;
- O status do defeito (p. ex., aberto, adiado, duplicado, aguardando correção, aguardando teste de confirmação, reaberto, fechado, rejeitado);
- As referências (p. ex., ao caso de teste).

Alguns desses dados podem ser incluídos automaticamente ao usar ferramentas de gerenciamento de defeitos (p. ex., identificador, data, autor e status inicial). Modelos de documentos para um relatório de defeitos e exemplos de relatórios de defeitos podem ser encontrados na norma ISO/IEC/IEEE 29119-3, que se refere a relatórios de defeitos como relatórios de incidentes.

# 6 Ferramentas de Teste (20 min)

## Palavras-chave

automação de testes

## Objetivos de aprendizagem

### 6.1 Suporte de Ferramentas para Testes

FL-6.1.1 (K2) Explicar como os diferentes tipos de ferramentas de teste dão suporte aos testes.

### 6.2 Benefícios e Riscos da Automação de Teste

FL-6.2.1 (K1) Relembrar os benefícios e os riscos da automação de testes.

## 6.1 Suporte de Ferramentas para Testes

As ferramentas de teste dão suporte e facilitam muitas atividades de teste. Os exemplos incluem, mas não se limitam a:

- Ferramentas de gerenciamento: aumentam a eficiência do processo de teste, facilitando o gerenciamento do SDLC, dos requisitos, dos testes, dos defeitos e da configuração;
- Ferramentas de teste estático: dão suporte ao Testador na realização de revisões e análises estáticas;
- Ferramentas de projeto e implementação de testes: facilitam a geração de casos de teste, dados de teste e procedimentos de teste;
- Ferramentas de execução e cobertura de testes: facilitam a execução de testes automatizados e a medição da cobertura;
- Ferramentas de teste não funcional: permitem que o Testador realize testes não funcionais que são difíceis ou impossíveis de serem realizados manualmente;
- Ferramentas de DevOps: suporte ao pipeline de entrega de DevOps, rastreamento de fluxo de trabalho, processo(s) de construção automatizado(s), CI/CD;
- Ferramentas de colaboração: facilitam a comunicação;
- Ferramentas que oferecem suporte à escalabilidade e à padronização da implantação (p. ex., máquinas virtuais, ferramentas de conteinerização);
- Qualquer outra ferramenta que auxilie no teste (p. ex., uma planilha é uma ferramenta de teste no contexto do teste).

## 6.2 Benefícios e Riscos da Automação de Teste

A simples aquisição de uma ferramenta não garante o sucesso. Cada nova ferramenta exigirá esforço para obter benefícios reais e duradouros (p. ex., para a introdução, manutenção e treinamento da ferramenta). Há também alguns riscos que precisam ser analisados e atenuados.

Os possíveis benefícios do uso da automação de testes incluem:

- Economia de tempo com a redução do trabalho manual repetitivo (p. ex., executar testes de regressão, reinserir os mesmos dados de teste, comparar os resultados esperados com os resultados reais e verificar os padrões de codificação);
- Prevenção de erros humanos simples por meio de maior consistência e repetibilidade (p. ex., os testes são consistentemente derivados dos requisitos, os dados de teste são criados de maneira sistemática e os testes são executados por uma ferramenta na mesma ordem e com a mesma frequência);
- Avaliação mais objetiva (p. ex., cobertura) e fornecimento de medidas que são muito complicadas para serem derivadas por humanos;
- Acesso mais fácil a informações sobre testes para dar suporte ao gerenciamento de testes e aos relatórios de testes (p. ex., estatísticas, gráficos e dados agregados sobre o progresso dos testes, taxas de defeitos e duração da execução dos testes);
- Redução dos tempos de execução de testes para proporcionar detecção antecipada de defeitos, feedback mais rápido e menor tempo de lançamento no mercado;

- Mais tempo para os Testadores criarem testes novos, mais profundos e mais eficazes.

Os possíveis riscos do uso da automação de testes incluem:

- Expectativas irreais sobre os benefícios de uma ferramenta (incluindo funcionalidade e facilidade de uso);
- Estimativas imprecisas de tempo, custos e esforço necessários para introduzir uma ferramenta, manter scripts de teste e alterar o processo de teste manual existente;
- Usar uma ferramenta de teste quando o teste manual é mais apropriado;
- Confiar demais em uma ferramenta, por exemplo, ignorando a necessidade do pensamento crítico humano;
- A dependência do fornecedor da ferramenta, que pode fechar as portas, aposentar a ferramenta, vender a ferramenta para outro fornecedor ou fornecer um suporte deficiente (p. ex., respostas a consultas, atualizações e correções de defeitos);
- Usar um software de código aberto que pode ser abandonado, o que significa que não há mais atualizações disponíveis, ou que seus componentes internos podem exigir atualizações bastante frequentes como um desenvolvimento adicional;
- A ferramenta de automação não é compatível com a plataforma de desenvolvimento;
- Escolha de uma ferramenta inadequada que não cumpra os requisitos normativos e/ou as normas de segurança.

# Referências

## Normas

**ISO/IEC/IEEE 29119-1 (2022)**

Software and systems engineering – Software testing – Part 1: General Concepts

**ISO/IEC/IEEE 29119-2 (2021)**

Software and systems engineering – Software testing – Part 2: Test processes

**ISO/IEC/IEEE 29119-3 (2021)**

Software and systems engineering – Software testing – Part 3: Test documentation

**ISO/IEC/IEEE 29119-4 (2021)**

Software and systems engineering – Software testing – Part 4: Test techniques

**ISO/IEC 25010, (2011)**

Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models

**ISO/IEC 20246 (2017)**

Software and systems engineering – Work product reviews

**ISO/IEC/IEEE 14764:2022**

Software engineering – Software life cycle processes – Maintenance ISO 31000 (2018) Risk management – Principles and guidelines

## Literatura

Adzic, G. (2009) Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing, Neuri Limited

Ammann, P. and Offutt, J. (2016) Introduction to Software Testing (2e), Cambridge University Press

Andrews, M. and Whittaker, J. (2006) How to Break Web Software: Functional and Security Testing of

Web Applications and Web Services, Addison-Wesley Professional

Beck, K. (2003) Test Driven Development: By Example, Addison-Wesley

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

Boehm, B. (1981) Software Engineering Economics, Prentice Hall, Englewood Cliffs, NJ

Buxton, J.N. and Randell B., eds (1970), Software Engineering Techniques. Report on a conference sponsored by the NATO Science Committee, Rome, Italy, 27–31 October 1969, p. 16

Chelimsky, D. et al. (2010) The Rspec Book: Behaviour Driven Development with Rspec, Cucumber, and Friends, The Pragmatic Bookshelf: Raleigh, NC

Cohn, M. (2009) Succeeding with Agile: Software Development Using Scrum, Addison-Wesley

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

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

Crispin, L. and Gregory, J. (2008) Agile Testing: A Practical Guide for Testers and Agile Teams, Pearson Education: Boston MA

Forgács, I., and Kovács, A. (2019) Practical Test Design: Selection of traditional and automated test design techniques, BCS, The Chartered Institute for IT

Gawande A. (2009) The Checklist Manifesto: How to Get Things Right, New York, NY: Metropolitan Books

Gärtner, M. (2011), ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, Pearson Education: Boston MA

Gilb, T., Graham, D. (1993) Software Inspection, Addison Wesley

Hendrickson, E. (2013) Explore It!: Reduce Risk and Increase Confidence with Exploratory Testing, The Pragmatic Programmers

Hetzel, B. (1988) The Complete Guide to Software Testing, 2nd ed., John Wiley and Sons

Jeffries, R., Anderson, A., Hendrickson, C. (2000) Extreme Programming Installed, Addison-Wesley Professional

Jorgensen, P. (2014) Software Testing, A Craftsman's Approach (4e), CRC Press: Boca Raton FL

Kan, S. (2003) Metrics and Models in Software Quality Engineering, 2nd ed., Addison-Wesley

Kaner, C., Falk, J., and Nguyen, H.Q. (1999) Testing Computer Software, 2nd ed., Wiley

Kaner, C., Bach, J., and Pettichord, B. (2011) Lessons Learned in Software Testing: A Context-Driven Approach, 1st ed., Wiley

Kim, G., Humble, J., Debois, P. and Willis, J. (2016) The DevOps Handbook, Portland, OR

Koomen, T., van der Aalst, L., Broekman, B. and Vroon, M. (2006) TMap Next for result-driven testing, UTN Publishers, The Netherlands

Myers, G. (2011) The Art of Software Testing, (3e), John Wiley & Sons: New York NY

O'Regan, G. (2019) Concise Guide to Software Testing, Springer Nature Switzerland

Pressman, R.S. (2019) Software Engineering. A Practitioner's Approach, 9th ed., McGraw Hill

Roman, A. (2018) Thinking-Driven Testing. The Most Reasonable Approach to Quality Control, Springer Nature Switzerland

Van Veenendaal, E (ed.) (2012) Practical Risk-Based Testing, The PRISMA Approach, UTN Publishers: The Netherlands

Watson, A.H., Wallace, D.R. and McCabe, T.J. (1996) Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, U.S. Dept. of Commerce, Technology Administration, NIST

Westfall, L. (2009) The Certified Software Quality Engineer Handbook, ASQ Quality Press

Whittaker, J. (2002) How to Break Software: A Practical Guide to Testing, Pearson

Whittaker, J. (2009) Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design, Addison Wesley

Whittaker, J. and Thompson, H. (2003) How to Break Software Security, Addison Wesley

Wiegers, K. (2001) Peer Reviews in Software: A Practical Guide, Addison-Wesley Professional

Articles and Web Pages

Brykczynski, B. (1999) "A survey of software inspection checklists," *ACM SIGSOFT Software Engineering Notes, 24(1), pp. 82-89*

Enders, A. (1975) "An Analysis of Errors and Their Causes in System Programs," *IEEE Transactions on Software Engineering 1(2), pp. 140-149*

Manna, Z., Waldinger, R. (1978) "The logic of computer programming," *IEEE Transactions on Software Engineering 4(3), pp. 199-229*

Marick, B. (2003) Exploration through Example, http://www.exampler.com/oldblog/2003/08/21.1.html#agile-testing-project-1

Nielsen, J. (1994) "Enhancing the explanatory power of usability heuristics," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Celebrating Interdependence, ACM Press, pp. 152–158*

Salman. I. (2016) "Cognitive biases in software quality and testing," *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16), ACM, pp. 823-826.*

Wake, B. (2003) "INVEST in Good Stories, and SMART Tasks," https://xp123.com/articles/invest-in-goodstories-and-smart-tasks/

# Apêndice A: Objetivos de Aprendizagem e Nível Cognitivo

Os seguintes Objetivos de Aprendizagem são definidos como aplicáveis a este syllabus. Cada tópico do syllabus será examinado de acordo com seu objetivo de aprendizado. Os Objetivos de Aprendizagem começam com um verbo de ação correspondente ao seu Nível Cognitivo de conhecimento, conforme listado abaixo.

**Nível 1: Lembrar (K1)** - o candidato lembrará, reconhecerá e recordará um termo ou conceito.

**Verbos de ação:** identificar, recordar, lembrar, reconhecer.

**Exemplos:**

- "Identifique os objetivos típicos do teste."
- "Relembre os conceitos da pirâmide de teste."
- "Reconhecer como um Testador agrega valor ao planejamento de iteração e lançamento"

**Nível 2: Compreensão (K2)** - o candidato pode selecionar as razões ou explicações para declarações relacionadas ao tópico e pode resumir, comparar, classificar e dar exemplos para o conceito de teste.

**Verbos de ação**: classificar, comparar, contrastar, diferenciar, distinguir, exemplificar, explicar, dar exemplos, interpretar, resumir.

**Exemplos:**

- "Classifique as diferentes opções para escrever critérios de aceite."
- "Compare as diferentes funções nos testes" (procure semelhanças, diferenças ou ambos).
- "Distinguir entre riscos de projeto e riscos de produto" (permite que os conceitos sejam diferenciados).
- "Exemplificar a finalidade e o conteúdo de um plano de teste."
- "Explique o impacto do contexto no processo de teste."
- "Faça um resumo das atividades do processo de revisão."

**Nível 3: Aplicar (K3)** - o candidato pode executar um procedimento quando confrontado com uma tarefa familiar, ou selecionar o procedimento correto e aplicá-lo a um determinado contexto.

**Verbos de ação**: aplicar, implementar, preparar, usar.

**Exemplos:**

- "Aplicar a priorização de casos de teste" (deve se referir a um procedimento, técnica, processo, algoritmo etc.).
- "Prepare um relatório de defeitos".
- "Use a análise de valor de limite para derivar casos de teste."

**Referências** para os níveis cognitivos dos objetivos de aprendizagem:

*Anderson, L. W. e Krathwohl, D. R. (eds) (2001) A Taxonomy for Learning, Teaching*

*Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon*

# Apêndice B: Matriz de rastreabilidade entre resultados de negócio x objetivos de aprendizagem

Esta seção lista o número de Objetivos de Aprendizagem do Nível Fundamental relacionados aos Resultados de Negócios e a Rastreabilidade entre os Resultados de Negócios do Nível Fundamental, e os Objetivos de Aprendizagem de Nível Fundamental.

| Resultados de Negócio: Foundation Level | | FL-B01 | FL-B02 | FL-B03 | FL-B04 | FL-B05 | FL-B06 | FL-B07 | FL-B08 | FL-B09 | FL-B10 | FL-B11 | FL-B12 | FL-B13 | FL-B14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **BO1** | Entender o que é teste e por que ele é benéfico | 6 | | | | | | | | | | | | | |
| **BO2** | Compreender os conceitos fundamentais dos testes de software | | 22 | | | | | | | | | | | | |
| **BO3** | Identificar a abordagem de teste e as atividades a serem implementadas, dependendo do contexto do teste | | | 6 | | | | | | | | | | | |
| **BO4** | Avaliar e melhorar a qualidade da documentação | | | | 9 | | | | | | | | | | |
| **BO5** | Aumentar a eficácia e a eficiência dos testes | | | | | 20 | | | | | | | | | |
| **BO6** | Alinhar o processo de teste com o ciclo de vida de desenvolvimento de software | | | | | | 6 | | | | | | | | |
| **BO7** | Compreender os princípios de gerenciamento de testes | | | | | | | 6 | | | | | | | |
| **BO8** | Escrever e comunicar relatórios de defeitos claros e compreensíveis | | | | | | | | 1 | | | | | | |
| **BO9** | Compreender os fatores que influenciam as prioridades e os esforços relacionados aos testes | | | | | | | | | 7 | | | | | |
| **BO10** | Trabalhar como parte de uma equipe multifuncional | | | | | | | | | | 8 | | | | |
| **BO11** | Conhecer os riscos e benefícios relacionados à automação de testes. | | | | | | | | | | | 1 | | | |
| **BO12** | Identificar as habilidades essenciais necessárias para os testes | | | | | | | | | | | | 5 | | |
| **BO13** | Compreender o impacto do risco nos testes | | | | | | | | | | | | | 4 | |
| **BO14** | Relatar com eficácia o progresso e a qualidade dos testes | | | | | | | | | | | | | | 4 |

## Capítulo 1 – Fundamentos de Teste

| Seção | Objetivos de Aprendizagem | Nível K | Resultados de Negócio | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | FL-B01 | FL-B02 | FL-B03 | FL-B04 | FL-B05 | FL-B06 | FL-B07 | FL-B08 | FL-B09 | FL-B10 | FL-B11 | FL-B12 | FL-B13 | FL-B14 |
| **1.1** | **O que é teste?** | | | | | | | | | | | | | | | |
| 1.1.1 | Objetivos do teste | K1 | X | | | | | | | | | | | | | |
| 1.1.2 | Teste e depuração | K2 | | X | | | | | | | | | | | | |
| **1.2** | **Por que os testes são necessários?** | | | | | | | | | | | | | | | |
| 1.2.1 | Contribuições para o sucesso dos testes | K2 | X | | | | | | | | | | | | | |
| 1.2.2 | Testes e Garantia da Qualidade (QA) | K1 | | X | | | | | | | | | | | | |
| 1.2.3 | Erros, Defeitos, Falhas e Causas-raiz | K2 | | X | | | | | | | | | | | | |
| **1.3** | **Princípios de Teste** | | | | | | | | | | | | | | | |
| 1.3.1 | Os 7 Princípios do Teste | K2 | | X | | | | | | | | | | | | |
| **1.4** | **Atividades de teste, Testware e Funções no teste** | | | | | | | | | | | | | | | |
| 1.4.1 | Atividades e Tarefas de Teste | K2 | | | X | | | | | | | | | | | |
| 1.4.2 | Contexto do Processo de Teste | K2 | | | X | | | X | | | | | | | | |
| 1.4.3 | Testware | K2 | | | X | | | | | | | | | | | |
| 1.4.4 | Rastreabilidade entre a Base de Teste e o Testware | K2 | | | | X | X | | | | | | | | | |
| 1.4.5 | Papéis no teste | K2 | | | | | | | | | | X | | | | |
| **1.5** | **Habilidades essenciais e boas práticas em testes** | | | | | | | | | | | | | | | |
| 1.5.1 | Habilidades genéricas necessárias para testes | K2 | | | | | | | | | | | | X | | |
| 1.5.2 | Abordagem de toda a equipe | K1 | | | | | | | | | | X | | | | |
| 1.5.3 | Independência dos testes | K2 | | | X | | | | | | | | | | | |

## Capítulo 2 – Teste ao longo do Ciclo de Vida de Desenvolvimento de Software

| Seção | Objetivos de Aprendizagem | Nível K | Resultados de Negócio | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | FL-B01 | FL-B02 | FL-B03 | FL-B04 | FL-B05 | FL-B06 | FL-B07 | FL-B08 | FL-B09 | FL-B10 | FL-B11 | FL-B12 | FL-B13 | FL-B14 |
| **2.1** | **Testes no contexto do Ciclo de Vida de Desenvolv. de Software** | | | | | | | | | | | | | | | |
| 2.1.1 | Impacto no Ciclo de Vida de Desenvolvimento de Software | K2 | | | | | | X | | | | | | | | |
| 2.1.2 | Ciclo de Vida de Desenvolvimento de Software e boas práticas | K1 | | | | | | X | | | | | | | | |
| 2.1.3 | Teste como um motivador para o desenvolvimento de software | K1 | | | | | X | | | | | | | | | |
| 2.1.4 | DevOps e Testes | K2 | | | | | X | X | | | X | X | | | | |
| 2.1.5 | Abordagem Shift-Left | K2 | | | | | X | X | | | | | | | | |
| 2.1.6 | Retrospectivas e melhorias de processo | K2 | | | | | x | | | | | X | | | | |
| **2.2** | **Níveis de Teste e Tipos de Teste** | | | | | | | | | | | | | | | |
| 2.2.1 | Níveis de Teste | K2 | | X | X | | | | | | | | | | | |
| 2.2.2 | Tipos de Teste | K2 | | X | | | | | | | | | | | | |
| 2.2.3 | Teste de Confirmação e Teste de Regressão | K2 | | X | | | | | | | | | | | | |
| **2.3** | **Teste de Manutenção** | | | | | | | | | | | | | | | |
| 2.3.1 | Resumir os testes de manutenção e seus disparadores | K2 | | X | | | | | X | | | | | | | |

## Capítulo 3 – Teste Estático

| Seção | Objetivos de Aprendizagem | Nível K | Resultados de Negócio | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | FL-B01 | FL-B02 | FL-B03 | FL-B04 | FL-B05 | FL-B06 | FL-B07 | FL-B08 | FL-B09 | FL-B10 | FL-B11 | FL-B12 | FL-B13 | FL-B14 |
| **3.1** | **Noções básicas de Teste Estático** | | | | | | | | | | | | | | | |
| 3.1.1 | Produtos de trabalho examináveis por testes estáticos | K1 | | | | X | X | | | | | | | | | |
| 3.1.2 | Valor do teste estático | K2 | X | | | X | X | | | | | | | | | |
| 3.1.3 | Diferenças entre testes estáticos e testes dinâmicos | K2 | | | | X | X | | | | | | | | | |
| **3.2** | **Processo de Feedback e revisão** | | | | | | | | | | | | | | | |
| 3.2.1 | Benefícios do feedback antecipado e frequente dos stakeholders | K1 | X | | | X | | | | | | X | | | | |
| 3.2.2 | Atividades do processo de revisão | K2 | | | X | X | | | | | | | | | | |
| 3.2.3 | Funções e responsabilidades nas revisões | K1 | | | | X | | | | | | | | X | | |
| 3.2.4 | Tipos de revisão | K2 | | X | | | | | | | | | | | | |
| 3.2.5 | Fatores de sucesso para revisões | K1 | | | | | X | | | | | | | X | | |

## Capítulo 4 – Análise e Modelagem de Teste

| Seção | Objetivos de Aprendizagem | Nível K | Resultados de Negócio | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | FL-B01 | FL-B02 | FL-B03 | FL-B04 | FL-B05 | FL-B06 | FL-B07 | FL-B08 | FL-B09 | FL-B10 | FL-B11 | FL-B12 | FL-B13 | FL-B14 |
| **4.1** | **Visão geral das técnicas de teste** | | | | | | | | | | | | | | | |
| 4.1.1 | Distinguir as técnicas de teste caixa-preta, caixa-branca e experiência | K2 | | X | | | | | | | | | | | | |
| **4.2** | **Técnicas de Teste Caixa-Preta** | | | | | | | | | | | | | | | |
| 4.2.1 | Particionamento de Equivalência (EP) | K3 | | | | | X | | | | | | | | | |
| 4.2.2 | Análise de Valor de Limite (BVA) | K3 | | | | | X | | | | | | | | | |
| 4.2.3 | Teste de Tabela de Decisão | K3 | | | | | X | | | | | | | | | |
| 4.2.4 | Teste de Transição de Estado | K3 | | | | | X | | | | | | | | | |
| **4.3** | **Técnicas de Teste Caixa-Branca** | | | | | | | | | | | | | | | |
| 4.3.1 | Teste de Instrução e Cobertura de Instrução | K2 | | X | | | | | | | | | | | | |
| 4.3.2 | Teste de Ramificação e Cobertura de Ramificação | K2 | | X | | | | | | | | | | | | |
| 4.3.3 | O valor do Teste Caixa-Branca | K2 | X | X | | | | | | | | | | | | |
| **4.4** | **Técnicas de Teste Baseadas na Experiência** | | | | | | | | | | | | | | | |
| 4.4.1 | Suposição de Erro | K2 | | X | | | | | | | | | | | | |
| 4.4.2 | Testes Exploratórios | K2 | | X | | | | | | | | | | | | |
| 4.4.3 | Testes baseados em Lista de Verificação | K2 | | X | | | | | | | | | | | | |
| **4.5** | **Abordagens de Teste Baseadas na Colaboração** | | | | | | | | | | | | | | | |
| 4.5.1 | Escrita colaborativa de histórias de usuários | K2 | | | | X | | | | | | X | | | | |
| 4.5.2 | Critérios de Aceite | K2 | | | | | | | | | | X | | | | |
| 4.5.3 | Desenvolvimento Orientado por Teste de Aceite (ATDD) | K3 | | | | | X | | | | | | | | | |

## Capítulo 5 – Gerenciamento das Atividades de Teste

| Seção | Objetivos de Aprendizagem | Nível K | Resultados de Negócio | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | FL-B01 | FL-B02 | FL-B03 | FL-B04 | FL-B05 | FL-B06 | FL-B07 | FL-B08 | FL-B09 | FL-B10 | FL-B11 | FL-B12 | FL-B13 | FL-B14 |
| **5.1** | **Planejamento de Teste** | | | | | | | | | | | | | | | |
| 5.1.1 | Objetivo e conteúdo de um plano de teste | K2 | | X | | | | | X | | | | | | | |
| 5.1.2 | Contribuição do Testador para o planejamento de iteração e liberação | K1 | X | | | | | | | | | X | | X | | |
| 5.1.3 | Critérios de Entrada e Critérios de Saída | K2 | | | | X | | X | | | | | | | | X |
| 5.1.4 | Técnicas de estimativa | K3 | | | | | | | X | | X | | | | | |
| 5.1.5 | Priorização de casos de teste | K3 | | | | | | | X | | X | | | | | |
| 5.1.6 | Pirâmide de Teste | K1 | | X | | | | | | | | | | | | |
| 5.1.7 | Quadrantes de Teste | K2 | | X | | | | | | | X | | | | | |
| **5.2** | **Gerenciamento de Risco** | | | | | | | | | | | | | | | |
| 5.2.1 | Definição de Risco e Atributos do Risco | K1 | | | | | | | X | | | | | | X | |
| 5.2.2 | Riscos do projeto e riscos do produto | K2 | | X | | | | | | | | | | | X | |
| 5.2.3 | Análise de Risco do Produto | K2 | | | | | X | | | | X | | | | X | |
| 5.2.4 | Controle de Risco do Produto | K2 | | X | | | X | | | | | | | | X | |
| **5.3** | **Monitoramento, Controle e Conclusão do Teste** | | | | | | | | | | | | | | | |
| 5.3.1 | Métricas usadas em testes | K1 | | | | | | | | | X | | | | | X |
| 5.3.2 | Relatórios de Teste: objetivo, conteúdo e público-alvo | K2 | | | | | X | | | | X | | | | | X |
| 5.3.3 | Comunicação do status dos testes | K2 | | | | | | | | | | | | X | | X |
| **5.4** | **Gerenciamento de Configuração (CM)** | | | | | | | | | | | | | | | |
| 5.4.1 | Resumir como o gerenciamento de configuração apoia os testes | K2 | | | | | X | | X | | | | | | | |
| **5.5** | **Gerenciamento de Defeitos** | | | | | | | | | | | | | | | |
| 5.5.1 | Preparar um relatório de defeitos | K3 | | X | | | | | | X | | | | | | |

## Capítulo 6 – Ferramentas de Teste

| Seção | Objetivos de Aprendizagem | Nível K | Resultados de Negócio | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | FL-B01 | FL-B02 | FL-B03 | FL-B04 | FL-B05 | FL-B06 | FL-B07 | FL-B08 | FL-B09 | FL-B10 | FL-B11 | FL-B12 | FL-B13 | FL-B14 |
| **6.1** | **Suporte de ferramentas para testes** | | | | | | | | | | | | | | | |
| 6.1.1 | Explicar como os diferentes tipos de ferramentas apoiam os testes | K2 | | | | | X | | | | | | | | | |
| **6.2** | **Benefícios e riscos da automação de testes** | | | | | | | | | | | | | | | |
| 6.2.1 | Relembrar os benefícios e riscos da automação de testes | K1 | | | | | X | | | | | | X | | | |

# Apêndice C: Notas de versão

O syllabus ISTQB® Foundation v4.0 é uma grande atualização baseada no syllabus Foundation Level (v3.1.1) e no syllabus Agile Tester 2014. Por esse motivo, não há notas de versão detalhadas por capítulo e seção. No entanto, um resumo das principais alterações é fornecido abaixo. Além disso, em um documento separado de Notas de Versão, o ISTQB® fornece a rastreabilidade entre os objetivos de aprendizagem (LO) na versão 3.1.1 do Foundation Level Syllabus, versão 2014 do Agile Tester Syllabus e os objetivos de aprendizagem no novo Foundation Level v4.0 Syllabus, mostrando quais LOs foram adicionados, atualizados ou removidos.

Na época em que o syllabus foi escrito (2022-2023), mais de um milhão de pessoas em mais de 100 países fizeram o exame de nível básico e mais de 800.000 são Testadores certificados em todo o mundo. Com a expectativa de que todos eles tenham lido o Foundation Syllabus para poderem passar no exame, isso faz com que o Foundation Syllabus seja provavelmente o documento de teste de software mais lido de todos os tempos! Essa grande atualização é feita em respeito a esse patrimônio e para melhorar a visão de centenas de milhares de pessoas sobre o nível de qualidade que o ISTQB® oferece à comunidade global de testes.

Nesta versão, todos os LOs foram editados para torná-los atômicos e para criar rastreabilidade um a um entre os LOs e as seções do syllabus, de modo que não haja conteúdo sem que haja também um LO. 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. Esta versão principal fez as seguintes alterações:

- Redução do tamanho do syllabus geral. O syllabus não é um livro, mas um documento que serve para delinear os elementos básicos de um curso introdutório sobre teste de software, incluindo os tópicos que devem ser abordados e em que nível. Portanto, em particular:
    - Na maioria dos casos, os exemplos são excluídos do texto. É tarefa do provedor de treinamento fornecer os exemplos, bem como os exercícios, durante o treinamento
    - Foi seguida a "Syllabus writing checklist", que sugere o tamanho máximo do texto para os LOs em cada nível K (K1 = máx. 10 linhas, K2 = máx. 15 linhas, K3 = máx. 25 linhas)
- Redução do número de LOs em comparação com os programas de estudos Foundation v3.1.1 e Agile v2014
    - 14 K1 LOs em comparação com 21 LOs no FL v3.1.1 (15) e AT 2014 (6)
    - 42 K2 LOs em comparação com 53 LOs no FL v3.1.1 (40) e AT 2014 (13)
    - 8 K3 LOs em comparação com 15 LOs no FL v3.1.1 (7) e AT 2014 (8)
- São fornecidas referências mais abrangentes a livros e artigos clássicos e/ou respeitados sobre testes de software e tópicos relacionados
- Principais alterações no capítulo 1 (Fundamentos de testes)
    - Seção sobre habilidades de teste ampliada e aprimorada
    - Seção sobre a abordagem de toda a equipe (K1) adicionada
    - A seção sobre a independência dos testes foi transferida do Capítulo 5 para o Capítulo 1
- Principais alterações no capítulo 2 (Testes ao longo dos SDLCs)
    - As seções 2.1.1 e 2.1.2 foram reescritas e aprimoradas, e os LOs correspondentes foram modificados

    - Mais foco em práticas como: abordagem de teste antecipado (K1), shift-left (K2), retrospectivas (K2)
    - Nova seção sobre testes no contexto de DevOps (K2)
    - Nível de teste de integração dividido em dois níveis de teste separados: teste de integração de componentes e teste de integração de sistemas
- Principais alterações no capítulo 3 (Teste Estático)
    - Seção sobre técnicas de revisão, juntamente com o K3 LO (aplicar uma técnica de revisão) removido
- Principais alterações no capítulo 4 (Análise e Modelagem de Teste)
    - Teste de caso de uso removido (mas ainda presente no syllabus do Advanced Test Analyst)
    - Mais foco na abordagem baseada na colaboração para testes: novo K3 LO sobre o uso do ATDD para derivar casos de teste e dois novos K2 LOs sobre histórias de usuários e critérios de aceite
    - Teste e cobertura de decisão substituídos por teste e cobertura de ramificação (primeiro, a cobertura de ramificação é mais comumente usada na prática; segundo padrões diferentes definem a decisão de forma diferente, em oposição à "ramificação"; terceiro, isso resolve uma falha sutil, mas grave, do antigo FL2018, que afirma que "100% de cobertura de decisão implica 100% de cobertura de instrução" - essa frase não é verdadeira no caso de programas sem decisões)
    - Seção sobre o valor do teste caixa-branca aprimorado
- Principais alterações no capítulo 5 (Gerenciamento das atividades de teste)
    - Removida a seção sobre estratégias/abordagens de teste
    - Novo K3 LO sobre técnicas de estimativa para estimar o esforço de teste
    - Mais foco nos conhecidos conceitos e ferramentas relacionados ao Ágil no gerenciamento de testes: iteração e planejamento de liberação (K1), pirâmide de testes (K1) e quadrantes de testes (K2)
    - A seção sobre gerenciamento de riscos é mais bem estruturada, descrevendo quatro atividades principais: identificação de riscos, avaliação de riscos, mitigação de riscos e monitoramento de riscos
- Principais alterações no capítulo 6 (Ferramentas de teste)
    - Conteúdo sobre alguns problemas de automação de testes reduzido por ser muito avançado para o nível básico - seção sobre seleção de ferramentas, execução de projetos-piloto e introdução de ferramentas na organização removida