Syllabus de Certificação de

Testador (*Tester*) de Nível *Foundation*

v4.0

International Software Testing Qualifications Board

Anúncio sobre Direitos de Autor

Histórico de Revisões

Versão	Data	Comentários
CTFL v4.0	21.04.2023	CTFL v4.0 – Versão de lançamento geral
CTFL v3.1.1	01.07.2021	CTFL v3.1.1 – Atualização dos direitos de autor e do logótipo
CTFL v3.1	11.11.2019	CTFL v3.1 – Lançamento de versão de manutenção com pequenas atualizações
ISTQB 2018	27.04.2018	CTFL v3.0 – Versão de lançamento geral para candidatos
ISTQB 2011	1.04.2011	CTFL – Lançamento de versão de manutenção
ISTQB 2010	30.03.2010	CTFL – Lançamento de versão de manutenção
ISTQB 2007	01.05.2007	CTFL – Lançamento de versão de manutenção
ISTQB 2005	01.07.2005	Syllabus de Certificação de Testador (*Tester*) de Nível *Foundation* v1.0
ASQF V2.2	07.2003	Syllabus ASQF Nível *Foundation* Versão v2.2 "Lehrplan Grundlagen des Software-testens"
ISEB V2.0	25.02.1999	ISEB *Software Testing Foundation Syllabus* v2.0

Índice

Agradecimentos

Este documento foi formalmente lançado pela Assembleia Geral do ISTQB®, no dia 21 de abril de 2023.

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

A equipa agradece a Stuart Reid, Patricia McQuaid e Leanne Howard pela sua revisão técnica e à equipa de revisão e aos Conselhos Nacionais pelas suas sugestões e contribuições.

As seguintes pessoas participaram na revisão, nos comentários e nas votações 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, 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, Tomas Rosenqvist, Werner Lieblang, Yaron Tsubery, Zhenlei Zuo e Zsolt Hargitai.

Grupo de trabalho do ISTQB de Nível *Foundation* (Edição 2018): Klaus Olsen (presidente), Tauhida Parveen (vice-presidente), Rex Black (gestor de projeto), 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 todos os Conselhos Nacionais pelas suas sugestões.

Grupo de trabalho do ISTQB de Nível *Foundation* (Edição 2011): Thomas Müller (presidente), Debra Friedenberg. A equipa nuclear agradece à equipa 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), assim como a todos os Conselhos Nacionais pelas sugestões para a versão atual deste *Syllabus*.

Grupo de trabalho do ISTQB de Nível *Foundation* (Edição 2010): Thomas Müller (presidente), Rahul Verma, Martin Klonk e Armin Beer. A equipa nuclear agradece à equipa de revisão (Rex Black, Mette-Bruhn Pederson, Debra Friedenberg, Klaus Olsen, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veendendaal), assim como a todos os Conselhos Nacionais pelas suas sugestões.

Grupo de trabalho do ISTQB de Nível *Foundation* (Edição 2007): Thomas Müller (presidente), Dorothy Graham, Debra Friedenberg e Erik van Veenendaal. A equipa nuclear agradece à equipa de revisão (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson e Wonil Kwon) e a todos os Conselhos Nacionais pelas suas sugestões.

Grupo de trabalho do ISTQB de Nível *Foundation* (Edição 2005): Thomas Müller (presidente), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson e Erik van Veenendaal. A equipa nuclear agradece à equipa de revisão e aos Conselhos Nacionais pelas suas sugestões.

0. Introdução

0.1. Finalidade deste *Syllabus*

Este documento consiste no *syllabus* base para a Qualificação Internacional em Testes de *Software* no Nível *Foundation*. O ISTQB® fornece este programa aos seguintes destinatários:

1. Aos Conselhos Nacionais, para tradução no seu idioma local e para acreditação de entidades formadoras. Os Conselhos Nacionais podem adaptar o programa às necessidades específicas do seu idioma local e modificar as referências de acordo com as publicações locais.
2. Aos organismos de certificação, para que possam criar a partir deste programa as questões para os exames na língua local adaptadas aos objetivos de aprendizagem para este programa.
3. Às entidades formadoras, para que possam produzir o material didático e determinar os métodos de ensino mais adequados.
4. Aos candidatos à certificação para preparação para o exame de certificação (como parte de um curso de formação ou de forma independente).
5. À comunidade internacional de engenheiros de *software* e sistemas, para alavancar a profissão de testador (*tester*) de software e sistemas, e como base para livros e artigos.

0.2. O Testador Certificado de Nível *Foundation* em Testes de Software

A qualificação de Nível *Foundation* destina-se a qualquer pessoa envolvida em testes de software, tais como: testadores, analistas de testes, engenheiros de testes, consultores de testes, gestores de testes, programadores de software e membros da equipa de desenvolvimento. A qualificação de Nível *Foundation* é também sugerida para quem quer obter conhecimentos básicos em testes de software, tais como gestores de projeto, gestores de qualidade, proprietários de produtos (product owners), gestores de desenvolvimento de software, analistas de negócio, diretores de TI e consultores de gestão. Os detentores da certificação de Nível *Foundation* poderão evoluir para certificações de nível superior em testes de *software*.

0.3. Carreira para Testadores

O programa do ISTQB® fornece suporte para profissionais de testes em todas as fases das carreiras, proporcionando conhecimentos abrangentes e profundos. Os indivíduos que obtiveram a Certificação de Nível *Foundation* do ISTQB® poderão também ter interesse nos Níveis Nucleares Avançados (Analista de Testes, Analista de Testes Técnicos e Gestor de Testes) e, por conseguinte, no Nível Especialista (Gestão de Testes ou Melhoria do Processo de Teste). Qualquer pessoa que pretenda desenvolver competências no domínio das práticas de testes num ambiente *Agile* pode considerar obter as Certificações de Testador Técnico de *Agile* ou Liderança de Testes *Agile* em Grande Escala. O fluxo de Especialista permite aprofundar áreas com abordagens de testes e atividades de testes específicas (p. ex., automação de testes, testes com IA, testes baseados em modelos, testes para aplicações móveis), ou áreas que estão relacionadas com testes específicas (p. ex., teste de desempenho, teste de usabilidade, teste de aceitação, teste de segurança), ou ainda obter os conhecimentos necessários de testes para determinados sectores

da indústria (p. ex., automóveis ou jogos). Para obter as informações mais recentes sobre todo o Modelo de Certificação de Testador visite o site do ISTQB® em www.istqb.org.

0.4. Valor para o Negócio

Esta secção apresenta os 14 pontos que representam o valor para o negócio que é esperado de uma pessoa que tenha obtido a Certificação de Nível *Foundation*.

Um Testador Certificado de Nível *Foundatio*n pode…

FL-BO1	Compreender o que são os testes e saber por que motivo são vantajosos
FL-BO2	Compreender os conceitos fundamentais dos testes de software
FL-BO3	Identificar as atividades e a abordagem de teste a implementar em função 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 do software
FL-BO7	Compreender os princípios da gestão de testes
FL-BO8	Escrever e comunicar relatórios de defeitos claros e compreensíveis
FL-BO9	Compreender os fatores que influenciam as prioridades e os esforços relacionados com os testes
FL-BO10	Trabalhar no contexto de uma equipa multifuncional
FL-BO11	Saber quais os riscos e as vantagens relacionadas com a automação de testes
FL-BO12	Identificar as competências essenciais necessárias para os testes
FL-BO13	Compreender o impacto dos riscos nos testes
FL-BO14	Informar sobre o progresso e a qualidade dos testes com eficácia

0.5. Objetivos de Aprendizagem Passíveis de Avaliação e Nível Cognitivo de Conhecimento

Os objetivos de aprendizagem suportam os 14 pontos associados ao valor para o negócio e são utilizados para criar os exames de Certificação de Testador de Nível *Foundation*.

No geral, todos os conteúdos dos capítulos 1-6 deste *syllabus* são passíveis de avaliação no exame no nível K1. Ou seja, pode ser solicitado ao candidato que reconheça, se lembre ou recorde uma palavra-chave ou conceito mencionado em qualquer um dos seis capítulos. Os valores específicos dos níveis associados aos objetivos de aprendizagem são apresentados no início de cada capítulo e classificados da seguinte forma:

- K1: Lembrar

- K2: Compreender
- K3: Aplicar

São apresentados mais detalhes e exemplos de objetivos de aprendizagem no Apêndice A. As definições de todos os termos que figurem como palavras-chave logo após o título de cada capítulo devem ser lembradas (K1), mesmo que não sejam mencionadas explicitamente nos objetivos de aprendizagem.

0.6. O Exame da Certificação de Nível *Foundation*

O exame da Certificação de Nível *Foundation* é baseado neste *syllabus*. As respostas às questões de exame podem requerer a utilização de material incluído nas várias secções deste *syllabus*. Todas as secções do *Syllabus* são passíveis de examinação, exceto a Introdução e os Apêndices. Estão incluídas no *Syllabus* as normas e os livros como referências (Capítulo 7), mas o seu conteúdo não é passível de examinação, para além do que está resumido neste respetivo *Syllabus* relativamente a tais normas e livros. Consulte o documento Estruturas e Regras dos Exames de Nível *Foundation*.

0.7. Acreditação

Um Conselho Nacional do ISTQB® pode acreditar entidades formadoras, cujo material do curso siga este *syllabus*. As entidades formadoras devem obter orientações de acreditação do Conselho Nacional ou de agentes que realizem a acreditação. Um curso acreditado é reconhecido como estando de acordo com este *syllabus*, e é permitido que exista um exame do ISTQB® como parte desse mesmo curso. As orientações de acreditação deste *Syllabus* seguem as Orientações de Acreditação gerais publicadas pelo Grupo de Trabalho responsável pela Gestão de Processos e Conformidade.

0.8. Tratamento de Normas

Existem normas referidas no Syllabus de Nível *Foundation* (p. ex., normas IEEE ou ISO). Estas referências fornecem uma estrutura (tal como referido na norma ISO 25010 relativamente às características de qualidade) ou fornecem uma fonte de informação adicional, se pretendido pelo leitor. Os documentos das normas não são passíveis de examinação. Para obter mais informações sobre as normas, consulte o Capítulo 7.

0.9. Manter-se Atualizado

A indústria do software está em constante evolução. Para lidar com estas alterações e proporcionar a todos os *stakeholders* o acesso a informações pertinentes e atuais, os Grupos de Trabalho do ISTQB® criaram ligações no site www.istqb.org, que se referem à documentação de apoio e às alterações das normas. Estas informações não são passíveis de examinação no âmbito do *Syllabus* de Nível *Foundation*.

0.10. Nível de Detalhe

O nível de detalhe deste *Syllabus* permite uma normalização dos cursos e dos exames a nível internacional. Para tal, o programa consiste em:

- Objetivos gerais de ensino descrevendo a intenção do Nível *Foundation*
- Lista de termos (palavras-chave) que o formando deve ser capaz de recordar
- Objetivos de aprendizagem para cada área de conhecimento, descrevendo os resultados relativos à aprendizagem cognitiva a alcançar
- Descrição dos principais conceitos, incluindo referências a fontes relevantes

O conteúdo programático não é uma descrição exaustiva da área de conhecimento de testes de software, reflete antes o nível de detalhe que deve ser abrangido em cursos de formação de Nível *Foundation*. Concentra-se em conceitos e técnicas de teste que podem ser aplicados a todos os projetos de software, independentemente do SDLC implementado.

0.11. Como está o *Syllabus* Organizado

Existem seis capítulos com conteúdo passível de examinação. Junto ao título de cada capítulo é especificada a duração desse capítulo em tempo de formação. A duração não é fornecida para os subcapítulos associados. Para cursos de formação acreditados, o *Syllabus* requer um mínimo de 1135 minutos (18 horas e 55 minutos) de instrução, distribuídos pelos seis capítulos da seguinte forma:

- Capítulo 1: Fundamentos dos Testes (180 minutos)
    - O formando aprende os princípios básicos relacionados com os testes, os motivos pelos quais os testes são necessários e quais são os objetivos dos testes.
    - O formando compreende o processo de teste, as atividades principais do teste e o *testware*.
    - O formando compreende as competências essenciais dos testes.
- Capítulo 2: Testes ao Longo do Ciclo de Vida de Desenvolvimento de *Software* (130 minutos)
    - O formando aprende como os testes são incorporados nas várias abordagens de desenvolvimento.
    - O formando aprende o conceito da abordagem *test-first*, bem como *DevOps.*
    - O formando aprende sobre os vários níveis de teste, tipos de teste e testes de manutenção.
- Capítulo 3: Testes Estáticos (80 minutos)
    - O formando aprende sobre os princípios básicos dos testes estáticos, o processo de *feedback* e o processo de revisão.
- Capítulo 4: Análise e Conceção de Testes (390 minutos)
    - O formando aprende a aplicar as técnicas de teste caixa-preta, técnicas de teste caixa-branca e técnicas de teste baseadas na experiência para obter casos de teste a partir dos vários produtos de trabalho de *software.*
    - O formando aprende sobre a abordagem de teste baseada na colaboração.

- Capítulo 5: Gestão das Atividades de Testes (335 minutos)
    - O formando aprende a planear os testes no geral, bem como a calcular uma estimativa do esforço de teste.
    - O formando aprende como os riscos influenciam o âmbito dos testes.
    - O formando aprende a monitorizar e controlar as atividades dos testes.
    - O formando aprende como a gestão de configurações suporta os testes.
    - O formando aprende a comunicar defeitos de uma forma clara e compreensível.
- Capítulo 6: Ferramentas de Testes (20 minutos)
    - O formando aprende a classificar as ferramentas e a compreender os riscos e as vantagens da automação de testes.

1. Fundamentos dos Testes – 180 minutos

Palavras-Chave

análise de teste, base para testes, caso de teste, causa raiz, cobertura, conceção de teste, conclusão do teste, condição de teste, controlo de teste, dados de teste, *debugging*, defeito, erro, execução de teste, falha, garantia de qualidade, implementação do teste, monitorização do teste, objetivo de teste, objeto de teste, planeamento de testes, procedimento de teste, qualidade, resultado do teste, teste, *testware*, validação, verificação

Objetivos de Aprendizagem para o Capítulo 1:

1.1 O que são Testes?

FL-1.1.1 (K1) Identificar objetivos típicos dos testes

FL-1.1.2 (K2) Diferenciar o teste do *debugging*

1.2 Porque Precisamos de Testes?

FL-1.2.1 (K2) Exemplificar os motivos pelos quais os testes são necessários

FL-1.2.2 (K1) Recordar a relação entre teste e garantia de qualidade

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

1.3 Princípios dos Testes

FL-1.3.1 (K2) Explicar os sete princípios dos testes

1.4 Atividades de Teste, *Testware* e Funções de Teste

FL-1.4.1 (K2) Resumir as diversas 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 o *testware* que suporta as atividades de teste

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

FL-1.4.5 (K2) Comparar as várias funções nos testes

1.5 Competências Essenciais e Boas Práticas nos Testes

FL-1.5.1 (K2) Dar exemplos das competências genéricas necessárias para os testes

FL-1.5.2 (K1) Recordar as vantagens do *Whole Team Approach*

FL-1.5.3 (K2) Distinguir as vantagens e desvantagens da independência do teste

1.1. O que são os Testes?

Os sistemas de software são parte integrante da nossa vida atual. A maioria das pessoas já teve uma experiência com software que não funcionou de acordo com o esperado. O software que não funciona corretamente pode provocar muitos problemas, incluindo perdas de tempo, dinheiro ou reputação profissional e, em casos extremos, mesmo ferimentos ou morte. Os testes de software avaliam a qualidade do software e ajudam a reduzir o risco de falha do software em funcionamento.

Os testes de software são um conjunto de atividades concebidas para descobrir os defeitos e avaliar a qualidade dos artefactos de software. Estes artefactos, quando estão a ser testados, são conhecidos como objetos de teste. Um erro comum sobre os testes é que os mesmos consistem apenas na execução de testes (ou seja, na execução do software e verificação dos resultados do teste). Contudo, os testes de software também incluem outras atividades e têm de estar alinhadas com o ciclo de vida de desenvolvimento do software (ver Capítulo 2).

Outro erro comum sobre os testes é que os mesmos se concentram exclusivamente na verificação do objeto de teste. Embora os testes impliquem verificação, ou seja, verificar se o sistema cumpre os requisitos especificados, também implica validação, o que significa verificar se o sistema irá satisfazer as necessidades dos utilizadores e dos outros *stakeholders* nos respetivos ambientes operacionais.

Os testes podem ser dinâmicos ou estáticos. Os testes dinâmicos requerem a execução do software, ao contrário dos testes estáticos. Os testes estáticos incluem revisões e análises estáticas (ver Capítulo 3). Os testes dinâmicos utilizam vários tipos de técnicas e abordagens de teste para derivar os casos de teste (ver Capítulo 4).

Os testes não são apenas uma atividade técnica. Também têm de ser devidamente planeados, geridos, calculados, monitorizados e controlados (ver Capítulo 5).

Os testadores utilizam ferramentas (ver Capítulo 6), mas é importante lembrar que os testes são principalmente uma atividade intelectual, que exige aos testadores ter conhecimentos especializados, utilizar competências analíticas e aplicar pensamento crítico e pensamento sistémico (Myers 2011 e Roman 2018).

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

1.1.1. Objetivos de Teste

Os objetivos de teste típicos são:

- Avaliar produtos de trabalho, tais como requisitos, *user stories*, conceções e código
- Encontrar falhas e defeitos
- Assegurar a cobertura requerida do objeto de teste
- Reduzir o nível de risco de software com qualidade inadequada
- Verificar se todos os requisitos especificados foram cumpridos
- Verificar se o objeto de teste cumpre os requisitos contratuais, legais e regulamentares
- Fornecer informação aos *stakeholders* para que possam tomar decisões informadas

- Aumentar a confiança na qualidade do objeto de teste
- Validar se o objeto de teste está concluído e funciona como esperado pelos *stakeholders*

Os objetivos de teste podem variar, dependendo do contexto, incluindo o produto de trabalho que está a ser testado, o nível de teste, os riscos, o ciclo de vida do desenvolvimento de software (SDLC) a ser seguido e os fatores relacionados com o contexto empresarial, p. ex., estrutura empresarial, fatores concorrenciais ou tempo de colocação no mercado.

1.1.2. Testes e *Debugging*

Os testes e o *debugging* são atividades separadas. Os testes podem acionar falhas que são causadas por defeitos no software (teste dinâmico) ou podem encontrar defeitos diretamente no objeto de teste (teste estático).

Quando o teste dinâmico (ver Capítulo 4) aciona uma falha, o *debugging* tem como foco encontrar as causas desta falha (defeitos), analisar as respetivas causas e proceder à sua eliminação. O processo típico de *debugging* neste caso implica:

- Reproduzir a falha
- Diagnóstico (encontrar a causa raiz do defeito)
- Corrigir a causa

Os testes de confirmação subsequentes verificam se as correções resolveram o problema. Idealmente, o teste de confirmação deve ser efetuado pela mesma pessoa que efetuou o teste inicial. Os testes de regressão subsequentes também podem ser efetuados para verificar se as correções estão a causar falhas noutras partes do objeto de teste (para obter mais informações sobre os testes de confirmação e os testes de regressão, consulte a Secção 2.2.3).

Quando o teste estático identifica um defeito, o *debugging* tem como foco a respetiva remoção. Não é necessário efetuar uma reprodução ou diagnóstico, uma vez que o teste estático encontra os defeitos diretamente e não consegue acionar falhas (ver Capítulo 3).

1.2. Porque Precisamos de Testes?

Os testes, como forma de controlo de qualidade, ajudam a concretizar os objetivos acordados dentro do âmbito, tempo, qualidade e restrições orçamentais estabelecidos. A contribuição dos testes para o sucesso não deve estar limitada às atividades da equipa de teste. Qualquer *stakeholder* pode utilizar as suas competências de teste para aproximar o projeto do sucesso. Testar os componentes, os sistemas e a documentação associada ajuda a identificar defeitos no software.

1.2.1. As Contribuições dos Testes para o Sucesso

Os testes são uma forma rentável de detetar defeitos. Por conseguinte, estes defeitos podem ser eliminados (através de *debugging* – uma atividade não relacionada com testes), o que significa que os testes contribuem indiretamente para obter objetos de teste com maior qualidade.

Os testes permitem avaliar diretamente a qualidade de um objeto de teste nas várias fases do SDLC. Estas medidas são utilizadas como parte de uma atividade de gestão de projetos mais vasta, contribuindo para as decisões de avançar para a próxima fase do SDLC, tal como a decisão de lançamento.

Os testes proporcionam aos utilizadores uma representação indireta no projeto de desenvolvimento. Os testadores asseguram que compreendem as necessidades dos utilizadores ao longo do ciclo de vida do desenvolvimento. A alternativa é envolver um conjunto representativo de utilizadores como parte do projeto de desenvolvimento, o que normalmente não é possível devido aos custos elevados e à falta de disponibilidade de utilizadores adequados.

Os testes podem também ser necessários para cumprir requisitos contratuais ou legais, ou normas regulamentares.

1.2.2. Testes e Garantia de Qualidade (QA)

Apesar de as pessoas utilizarem os termos "testes" e "QA" (*Quality Assurance*) de forma indistinta, os testes e a garantia de qualidade (QA) não são iguais. Os testes são uma forma de controlo de qualidade (QC).

O controlo de qualidade (QC) é uma abordagem corretiva, orientada para o produto, que se concentra nas atividades que suportam a concretização dos níveis adequados de qualidade. Os testes são uma das principais formas de controlo de qualidade, enquanto outras incluem métodos formais (verificação de modelos e prova de conformidade), simulação e criação de protótipos.

A garantida de qualidade (QA) é uma abordagem preventiva, orientada para os processos, que se concentra na implementação e melhoria dos processos. Funciona com base no facto de que se um bom processo for seguido corretamente irá, por conseguinte, gerar um bom produto. Também se aplica aos processos de desenvolvimento e de testes, e é da responsabilidade de todos os envolvidos num projeto.

Os resultados do teste são utilizados por QA e QC. No controlo de qualidade (QC), os testes são utilizados para corrigir defeitos, enquanto na garantia de qualidade (QA) fornecem *feedback* sobre o nível de desempenho dos processos de desenvolvimento e teste.

1.2.3. Erros, Defeitos, Falhas e Causas Raiz do Defeito

Os seres humanos cometem erros (enganos), o que pode levar à introdução de defeitos (bugs), o que, por sua vez, pode resultar em falhas. Os seres humanos cometem erros por vários motivos, tais como pressão de tempo, complexidade dos produtos de trabalho, processos, infraestrutura ou interações, ou simplesmente devido ao cansaço ou porque não têm a formação adequada.

Os defeitos podem ser encontrados na documentação, tal como numa especificação de requisitos ou num *script* de teste, no código-fonte, ou num artefacto de suporte como um ficheiro de compilação. Os defeitos em artefactos produzidos anteriormente no SDLC, se não forem detetados, conduzem frequentemente a artefactos com defeitos mais tarde no ciclo de vida. Se um defeito no código for executado, o sistema poderá deixar de fazer o que deveria, ou fazer algo que não deveria, causando uma falha. Alguns defeitos irão resultar sempre numa falha se forem executados, enquanto outros apenas resultarão numa falha em circunstâncias específicas, e alguns poderão nunca resultar numa falha.

Os erros e os defeitos não são a única causa das falhas. As falhas também podem ser causadas por condições ambientais, tais como radiação e campos eletromagnéticos, que podem causar defeitos no *firmware*.

Uma causa raiz é o motivo principal para a ocorrência de um problema (p. ex., uma situação que resulta num erro). As causas raiz são identificadas através da análise da raiz do problema, que é normalmente efetuada quando ocorre uma falha ou é identificado um defeito. Estima-se que mais falhas ou defeitos semelhantes possam ser evitados, ou a sua frequência possa ser reduzida, se a causa raiz for atacada, nomeadamente através da sua eliminação.

1.3. Princípios dos Testes

Têm sido sugeridos ao longo dos anos um número de princípios dos testes que apresentam orientações gerais aplicáveis a todos os testes. Este *syllabus* descreve sete desses princípios.

1. Os testes mostram a presença de defeitos, não a sua ausência. Os testes podem mostrar a presença de defeitos no objeto de teste, mas não provam a inexistência dos mesmos (Buxton 1970). Os testes reduzem a probabilidade de haver defeitos que permanecem por descobrir no objeto de teste. Contudo, na hipótese de não serem detetados defeitos, os testes não permitem provar a conformidade do objeto de teste.

2. Os testes exaustivos são impossíveis. Testar tudo não é viável, exceto para casos triviais (Manna 1978). Em vez de se tentar testar exaustivamente, as técnicas de teste (ver Capítulo 4), a priorização do caso de teste (ver Secção 5.1.5), e os testes baseados na avaliação do risco (ver Secção 5.2), devem ser utilizados para concentrar os esforços de teste.

3. Os testes antecipados poupam tempo e dinheiro. Os defeitos que são removidos no início do processo não causarão defeitos subsequentes nos produtos de trabalho obtidos. O custo da qualidade será reduzido uma vez que irão ocorrer menos falhas posteriormente no SDLC (Boehm 1981). Para encontrar defeitos mais cedo, o teste estático (ver Capítulo 3) e o teste dinâmico (ver Capítulo 4) devem ser iniciados o mais cedo possível.

4. Os defeitos agrupam-se. Um número reduzido de componentes do sistema normalmente apresenta a maioria dos defeitos detetados, ou é responsável pela maioria das falhas operacionais (Enders 1975). Este fenómeno é uma ilustração do princípio de Pareto. Os grupos de defeitos previstos e os grupos de defeitos reais observados durante os testes ou em operação, são um contributo importante nos testes baseados na avaliação do risco (ver Secção 5.2).

5. Os testes desgastam. A repetição exaustiva dos mesmos testes torna cada vez mais ineficaz a deteção de novos defeitos (Beizer 1990). Para superar este efeito, os testes e os dados de teste existentes podem ter de ser modificados e pode ser necessário escrever novos testes. Contudo, em alguns casos, a repetição dos mesmos testes pode ter um resultado benéfico, p. ex., nos testes de regressão automatizados (ver Secção 2.2.3).

6. Os testes são dependentes do contexto. Não existe uma única abordagem universal aplicável para os testes. Os testes são efetuados de forma diferente em diferentes contextos (Kaner 2011).

7. Falácia da ausência de defeitos. É uma falácia (ou seja, um erro comum) esperar que a verificação de *software* irá assegurar o sucesso de um sistema. Testar exaustivamente todos os requisitos especificados e corrigir todos os defeitos encontrados pode ainda assim produzir um sistema que não satisfaz as necessidades e as expetativas dos utilizadores, que não ajude a concretizar os objetivos de negócio do cliente, ou que é inferior em comparação com outros sistemas concorrentes. Para além da verificação, também deve ser efetuada a validação (Boehm 1981).

1.4. Atividades de Teste, *Testware* e Funções de Teste

Os testes são dependentes do seu contexto. Contudo, num nível superior, existem conjuntos de atividades de teste comuns sem as quais os testes terão menos probabilidade de concretizar os objetivos estabelecidos. Estes conjuntos de atividades de teste criam um processo de teste. O processo de teste pode ser adaptado a uma determinada situação com base em vários fatores. Questões como quais são as atividades de teste que estão incluídas no processo de teste, como são implementadas, e quando ocorrem, são normalmente decididas no âmbito do planeamento dos testes dessa situação específica (ver Secção 5.1).

As seguintes secções descrevem os aspetos gerais deste processo de teste em termos das atividades e tarefas de teste, do impacto do contexto, *testware*, rastreabilidade entre a base para testes e o *testware*, 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

Normalmente, um processo de teste consiste nos seguintes grupos de atividades principais descritos abaixo. Apesar de muitas destas atividades parecerem seguir uma sequência lógica, são muitas vezes implementadas iterativamente ou paralelamente. Estas atividades de teste têm de ser adaptadas ao sistema e ao projeto.

O planeamento de testes consiste em definir os objetivos dos testes e selecionar a abordagem que melhor concretizará os objetivos, dentro das limitações impostas pelo contexto global. O planeamento de testes é explicado mais adiante na secção 5.1.

Monitorização e controlo de testes. A monitorização do teste implica a verificação contínua de todas as atividades de teste e a comparação do progresso atual em relação ao planeado. O controlo de testes envolve a adoção de medidas necessárias para cumprir os objetivos do teste. A monitorização e o controlo de testes são explicados na secção 5.3.

A análise de teste é utilizada para analisar a base para testes e identificar as características passíveis de teste e definir as condições de teste associadas, juntamente com os riscos relacionados e os níveis de risco (ver Secção 5.2). A base para testes e os objetos de teste também são avaliados para identificar os defeitos que podem conter e avaliar a respetiva testabilidade. A análise de teste é frequentemente suportada pela utilização de técnicas de teste (ver Capítulo 4). Por conseguinte, a análise de teste responde à pergunta "o que testar?" em termos de critérios mensuráveis de cobertura.

A conceção de teste implica a concretização das condições de teste em casos de teste e outro testware (p. ex., cartas de testes). Esta atividade envolve frequentemente a identificação dos itens de cobertura, que servem de guia para especificar os dados de entrada do caso de teste. As técnicas de teste (ver Capítulo 4) podem ser utilizadas para suportar esta atividade. A conceção do teste também implica definir os requisitos dos dados de teste, conceber o ambiente de teste e identificar quaisquer outras infraestruturas e ferramentas necessárias. Consequentemente, a conceção de teste responde à pergunta "como testar?".

A implementação do teste implica criar ou adquirir o *testware* necessário para a execução de teste (p. ex., dados de teste). Os casos de teste podem ser organizados em procedimentos de teste e são frequentemente reunidos em baterias de testes. São criados *scripts* de teste manuais e automatizados. Os procedimentos de teste são priorizados e organizados num cronograma de execução de testes para obter

uma execução de teste eficiente (ver Secção 5.1.5). O ambiente de teste é criado e verificado para ficar configurado corretamente.

A execução de teste inclui executar os testes de acordo com o cronograma de execução de testes (execuções de teste). A execução de teste pode ser manual ou automatizada. A execução de teste pode assumir várias formas, incluindo testes contínuos ou sessões de teste em pares. Os resultados do teste são comparados com os resultados esperados. Os resultados do teste são registados. As anomalias são analisadas para identificar as causas prováveis. Por conseguinte, esta análise permite comunicar as anomalias com base nas falhas observadas (ver Secção 5.5).

Normalmente, as atividades de **conclusão do teste** ocorrem em marcos do projeto (p. ex., lançamento, fim da iteração, conclusão do nível de teste) para quaisquer defeitos não resolvidos, pedidos de alteração ou itens do *backlog* do produto criados. Qualquer *testware* que possa ser útil no futuro é identificado e arquivado ou entregue às equipas apropriadas. O ambiente de teste é encerrado para um estado acordado. As atividades de teste são analisadas para identificar as lições aprendidas e as melhorias a implementar em iterações, lançamentos ou projetos futuros (ver Secção 2.1.6). Por último, é criado e comunicado um relatório de conclusão do teste aos *stakeholders*.

1.4.2. Processo de Teste em Contexto

Os testes não são efetuados de forma isolada. As atividades de teste são uma parte integral dos processos de desenvolvimento realizados dentro de uma organização. Os testes também são financiados pelos *stakeholders* com o objetivo final de satisfazer as necessidades do negócio desses mesmos *stakeholders*. Por conseguinte, a forma como os testes serão realizados irá depender de um número de fatores contextuais, incluindo:

- *Stakeholders* (necessidades, expetativas, requisitos, vontade de colaborar, etc.)
- Membros da equipa (competências, conhecimento, nível de experiência, disponibilidade, necessidades de formação, etc.)
- Área de negócio (criticidade do objeto de teste, riscos identificados, necessidades de mercado, regulamentações legais específicas, etc.)
- Fatores técnicos (tipo de software, arquitetura do produto, tecnologia utilizada, etc.)
- Restrições do projeto (âmbito, prazo, 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.)

Estes fatores terão um impacto em vários tópicos relacionados com os testes, incluindo: estratégia de teste, técnicas de teste utilizadas, grau da automação de testes, nível de cobertura necessário, nível de detalhe para a documentação de teste, relatórios, etc.

1.4.3. *Testware*

O *testware* é criado como resultado dos produtos de trabalho (entregáveis) das atividades de teste descritas na Secção 1.4.1. Existe uma variação significativa na forma como as diferentes organizações

produzem, elaboram, atribuem um nome, organizam e efetuam a gestão dos produtos de trabalho. Uma gestão de configurações correta (ver Secção 5.4) assegura a consistência e a integridade dos produtos de trabalho. A seguinte lista de produtos de trabalho não é exaustiva:

- **Os produtos de trabalho do planeamento de testes** incluem: plano de testes, cronograma de testes, registo de riscos e critérios de entrada e de saída (ver Secção 5.1). O registo de riscos é uma lista de riscos que reúne a probabilidade e o impacto desses riscos, bem como informações sobre a mitigação do risco (ver Secção 5.2). Normalmente, o cronograma de testes, o registo de riscos, e os critérios de entrada e de saída fazem parte do plano de testes.
- **Os produtos de trabalho de monitorização e controlo de testes** incluem: relatórios de progresso de testes (ver Secção 5.3.2), documentação das diretivas de controlo (ver Secção 5.3) e informações dos riscos (ver Secção 5.2).
- **Os produtos de trabalho da análise de teste** incluem: condições de teste (priorizadas) (p. ex., critérios de aceitação, ver Secção 4.5.2), e relatórios de defeitos em relação aos defeitos na base para testes (se não forem corrigidos).
- **Os produtos de trabalho da conceção de teste** incluem: casos de teste (priorizados), cartas de testes, itens de cobertura, requisitos dos dados de teste e requisitos do ambiente de teste.
- **Os produtos de trabalho da implementação do teste** incluem: procedimentos de teste, *scripts* de teste automatizados, baterias de testes, dados de teste, cronogramas de execução de testes e elementos do ambiente de teste. Exemplos de elementos do ambiente de teste incluem: *stubs*, controladores, simuladores e virtualizações de serviços.
- **Os produtos de trabalho de execução de teste** incluem: registos de teste e relatório de defeitos (ver Secção 5.5).
- **Os produtos de trabalho de conclusão do teste** incluem: relatório de conclusão do teste (ver Secção 5.3.2), pontos de melhoria em projetos ou iterações subsequentes, documentação das lições aprendidas e pedidos de alteração (p. ex., itens do *backlog* do produto).

1.4.4. Rastreabilidade entre a Base para Testes e o *Testware*

Para implementar uma monitorização e um controlo eficaz dos testes é importante estabelecer e manter a rastreabilidade ao longo do processo de teste entre os vários elementos: base para testes, o *testware* associado aos respetivos elementos (p. ex., condições de teste, riscos, casos de teste), resultados do teste e defeitos detetados.

Uma rastreabilidade exata suporta a avaliação da cobertura, pelo que é bastante útil ter os critérios mensuráveis de cobertura definidos na base para testes. Os critérios de cobertura podem funcionar como indicadores-chave de desempenho (KPI) para impulsionar as atividades que mostram até que ponto os objetivos de teste podem ser concretizados (ver Secção 1.1.1). Por exemplo:

- A rastreabilidade dos casos de teste com requisitos permite verificar se os requisitos são cobertos por casos de teste.
- A rastreabilidade dos resultados do teste com riscos pode ser utilizada para avaliar o nível de risco residual num objeto de teste.

Para além de avaliar a cobertura, uma boa rastreabilidade permite determinar o impacto das alterações, facilitar as auditorias de teste e ajudar a cumprir os critérios de governança de TI. Uma boa rastreabilidade

também torna os relatórios de progresso de testes e de conclusão do teste mais fáceis de compreender ao incluir o estado na perspetiva dos elementos da base para testes. Isto também pode ajudar a comunicar os aspetos técnicos dos testes aos *stakeholders* de forma mais compreensível. A rastreabilidade fornece informações que permitem avaliar a qualidade do produto, a capacidade do processo e o progresso do projeto em relação aos objetivos de negócio.

1.4.5. Funções nos Testes

Neste *syllabus*, vamos abordar duas funções de teste: a função de gestão de testes e a função de teste. As atividades e tarefas atribuídas a estas duas funções dependem de fatores, tais como o projeto e o contexto do produto, das competências das pessoas nas funções e da organização.

A função de gestão de testes assume a responsabilidade geral pelo processo de teste, pela equipa de teste e pela liderança das atividades de teste. A função de gestão de testes concentra-se principalmente nas atividades de planeamento de testes, monitorização do teste e controlo e conclusão do teste. A forma como a função de gestão de testes é executada varia consoante o contexto. Por exemplo, no desenvolvimento ágil de software, algumas das tarefas de gestão de testes podem ser efetuadas pela equipa *Agile*. As tarefas que abrangem várias equipas ou toda a organização podem ser efetuadas por gestores de testes fora da equipa de desenvolvimento.

A função de teste assume a responsabilidade geral pelo aspeto (técnico) da engenharia do teste. A função de teste concentra-se principalmente nas atividades de análise de teste, conceção de teste, implementação do teste e execução de teste.

Várias pessoas podem assumir estas funções em diferentes níveis dos testes. Por exemplo, a função de gestão de testes pode ser executada por um líder de equipa, gestor de testes, gestor de desenvolvimento, etc. Também é possível que uma única pessoa assuma as funções de teste e gestão de testes em simultâneo.

1.5. Competências Essenciais e Boas Práticas nos Testes

As competências são as capacidades de efetuar algo corretamente que advêm das práticas, das aptidões e dos conhecimentos adquiridos. Os bons testadores devem ter algumas competências essenciais para efetuarem corretamente o seu trabalho. Também devem conseguir trabalhar em equipa com eficácia e serem capazes de efetuar testes em diferentes níveis da independência dos testes.

1.5.1. Competências Gerais Necessárias para os Testes

Apesar de serem gerais, as seguintes competências são particularmente relevantes para os testadores:

- Conhecimento de testes (para aumentar o nível de eficácia dos testes, p. ex., ao utilizar técnicas de teste)
- Minucioso, cuidado, curioso, atenção ao detalhe, ser metódico (a identificar defeitos, especialmente os mais difíceis de encontrar)
- Boa capacidade de comunicação, escuta ativa, espírito de equipa (para interagir eficazmente com todos os *stakeholders*, transmitir informações a outras pessoas, ser compreendido, bem como comunicar e falar sobre defeitos)

- Pensamento analítico, pensamento crítico, criatividade (para aumentar o nível de eficácia dos testes)
- Conhecimento técnico (para aumentar o nível de eficiência dos testes, p. ex., ao utilizar as ferramentas de teste adequadas)
- Conhecimento da área de negócio (ser capaz de compreender e comunicar com os utilizadores finais/representantes do negócio)

Os testadores são muitas vezes os portadores de más notícias e uma característica humana comum é culpar o portador de más notícias, o que torna as competências de comunicação um aspeto crucial para os testadores. A comunicação de resultados do teste pode ser percecionada como uma crítica ao produto e ao seu autor, e a confirmação tendenciosa deste pode dificultar a aceitação de informações que discordem com as suas crenças. Por conseguinte, algumas pessoas podem percecionar os testes como uma atividade destrutiva, apesar de contribuírem significativamente para o progresso do projeto e para a qualidade do produto. Para tentar melhorar este aspeto, as informações sobre os defeitos e as falhas devem ser comunicadas de forma construtiva.

1.5.2. *Whole Team Approach*

Uma das competências mais importantes para o testador é a capacidade de trabalhar com eficácia em contexto de equipa e contribuir de forma positiva para os objetivos da mesma. A abordagem *whole team approach* – uma prática que teve origem em eXtreme Programming (ver Secção 2.1) – tenta desenvolver ainda mais esta competência.

No *whole team approach*, qualquer membro da equipa com as competências e os conhecimentos necessários pode efetuar quaisquer tarefas e todos os envolvidos são responsáveis pela qualidade. Os membros da equipa partilham o mesmo espaço de trabalho (físico ou virtual), uma vez que a co-localização facilita e promove a comunicação e a interação. O *whole team approach* melhora a dinâmica, a comunicação e a colaboração da equipa, e cria sinergias ao permitir tirar proveito dos vários conjuntos de competências da equipa em benefício do projeto.

Os testadores trabalham em conjunto com os outros membros da equipa para assegurar que os níveis de qualidade pretendidos são alcançados. Isto inclui colaborar com os representantes do negócio no sentido de os ajudar a criar testes de aceitação adequados, trabalhar com os programadores para acordar a estratégia de teste e decidir as abordagens à automação de testes a seguir. Os testadores podem, desta forma, transferir o seu conhecimento dos testes aos outros membros da equipa e influenciar o desenvolvimento do produto.

Dependendo do contexto, o *whole team approach* pode nem sempre ser adequado. Por exemplo, em algumas situações, tais como nos sistemas críticos em segurança, poderá ser necessário um alto nível de independência dos testes.

1.5.3. Independência do Teste

Por vezes, um determinado grau de independência torna o testador mais eficaz na deteção de defeitos devido a diferenças entre os preconceitos cognitivos do autor e do testador (Salman 1995). A independência não é, no entanto, um substituto para o conhecimento daquilo que me é familiar. Por exemplo, os programadores podem encontrar defeitos no seu próprio código de forma eficiente.

Os produtos de trabalho podem ser testados pelo seu autor (sem independência), pelos pares do autor dentro da mesma equipa (alguma independência), pelos testadores fora da equipa do autor, mas dentro da organização (alta independência), ou pelos testadores de fora da organização (elevada independência). Na maior parte dos projetos é melhor efetuar os testes com vários níveis de independência (p. ex., programadores a efetuarem testes de componentes e testes de integração de componentes, equipa de teste a efetuar testes de sistema e testes de integração de sistemas, e representantes do negócio a efetuarem testes de aceitação).

A principal vantagem da independência do teste é permitir que os testadores independentes reconheçam diferentes tipos de falhas e defeitos em comparação com os programadores devido às suas diferentes origens, perspetivas técnicas e preconceitos. Além disso, um testador independente pode verificar, desafiar ou desaprovar pressupostos assumidos pelos *stakeholders* durante a especificação e implementação do sistema.

Contudo, existem também algumas desvantagens. Os testadores independentes podem estar isolados da equipa de desenvolvimento, o que pode resultar em falta de colaboração, problemas de comunicação ou numa relação antagónica com a equipa de desenvolvimento. Os programadores podem também perder o sentido de responsabilidade pela qualidade. Os testadores independentes podem ser vistos como um ponto de bloqueio ou como culpados dos atrasos no lançamento de novas versões.

2. Testes ao Longo do Ciclo de Vida de Desenvolvimento de Software – 130 minutos

Palavras-Chave

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

Objetivos de Aprendizagem para o Capítulo 2:

2.1 Testes no Contexto dos Ciclos de Vida de Desenvolvimento de Software

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

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

FL-2.1.3 (K1) Recordar os exemplos de abordagens *test-first* no desenvolvimento

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 utilizar retrospetivas como mecanismo de melhoria de processos

2.2 Níveis de Teste e Tipos de Teste

FL-2.2.1 (K2) Distinguir os vários níveis de teste

FL-2.2.2 (K2) Distinguir os vários tipos de teste

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

2.3 Testes de Manutenção

FL-2.3.1 (K2) Resumir os testes de manutenção e os fatores que os desencadeiam

2.1. Testes no Contexto dos Ciclos de Vida de Desenvolvimento de Software

O modelo de ciclo de vida de desenvolvimento de software (SDLC) é uma representação abstrata de alto nível do processo de desenvolvimento de software. Um modelo de SDLC define como as várias fases de desenvolvimento e os tipos de atividades realizadas neste processo estão relacionadas entre si, de forma lógica e cronológica. Exemplos dos modelos de SDLC incluem: modelos de desenvolvimento sequencial (p. ex., modelo *Waterfall*, modelo em V), modelos de desenvolvimento iterativo (p. ex., modelo em espiral, criação de protótipos) e modelos de desenvolvimento incremental (p. ex., *Unified Process*).

Algumas das atividades no processo de desenvolvimento de software também podem ser descritas por métodos de desenvolvimento de software mais detalhados e práticas *Agile*. Exemplos incluem: desenvolvimento orientado para testes de aceitação (ATDD), desenvolvimento orientado para comportamento (BDD), conceção orientada para domínio (DDD), *eXtreme Programming* (XP), desenvolvimento orientado para funcionalidades (FDD), Kanban, *Lean IT*, *Scrum* e desenvolvimento orientado para testes (TDD).

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

Os testes têm de ser adaptados ao SDLC para serem bem-sucedidos. A escolha de SDLC tem impacto nos seguintes aspetos:

- Âmbito e calendarização das atividades de teste (p. ex., níveis de teste e tipos de teste)
- Nível de detalhe para a documentação de teste
- Escolha das técnicas de teste e abordagens de teste
- Extensão da automação de testes
- Função e responsabilidades do testador

Nos modelos de desenvolvimento sequencial e nas fases iniciais, os testadores participam nas revisões de requisitos, análises de teste e conceções de teste. O código executável é criado nas fases posteriores e, por conseguinte, o teste dinâmico não pode efetuado no início de SDLC.

Em alguns modelos de desenvolvimento iterativos e incrementais, parte-se do princípio de que cada iteração fornece um protótipo funcional ou um incremento do produto. Isto implica que em cada iteração, os testes estáticos e dinâmicos podem ser efetuados em todos os níveis de teste. A entrega frequente dos incrementos requer *feedback* rápido e testes de regressão extensos.

O desenvolvimento ágil de software assume que a mudança pode ocorrer ao longo do projeto. Por conseguinte, é dada preferência à documentação de produto menos detalhada e à automação de testes extensa para simplificar o teste de regressão nos projetos *Agile*. Além disso, a maior parte dos testes manuais tendem a ser efetuados utilizando técnicas de teste baseadas na experiência (ver Secção 4.4) que não requerem a análise e a conceção de testes prévia e extensa.

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 selecionado, incluem os seguintes aspetos:

- Para cada atividade de desenvolvimento de software existe uma atividade de teste correspondente, para que todas as atividades de desenvolvimento estejam sujeitas a um controlo de qualidade
- Os diferentes níveis de teste (ver Capítulo 2.2.1) têm objetivos de teste específicos e distintos, o que permite que os testes sejam devidamente abrangentes e evitem a redundância
- A análise e a conceção de testes para um determinado nível de teste começam durante a fase de desenvolvimento correspondente do SDLC, para que os testes possam aderir ao princípio dos testes antecipados (ver Secção 1.3)
- Os testadores são envolvidos na revisão dos produtos de trabalho assim que os esboços desta documentação estiverem disponíveis, para que os testes antecipados e a deteção de defeitos possam suportar a estratégia *shift-left* (ver Secção 2.1.5)

2.1.3. Testes enquanto Orientador do Desenvolvimento de Software

TDD, ATDD e BDD são abordagens de desenvolvimento semelhantes, onde os testes são definidos como uma forma de orientar o desenvolvimento. Cada uma destas abordagens implementa o princípio dos testes antecipados (ver Secção 1.3) e segue uma abordagem *shift-left* (ver Secção 2.1.5), pois os testes são definidos antes da escrita do código. Também suportam um modelo de desenvolvimento iterativo. Estas abordagens são caracterizadas da seguinte forma:

Desenvolvimento Orientado para Testes (TDD):

- Orienta a codificação através de casos de teste (em vez de conceção de software extensa) (Beck 2003)
- Os testes são escritos em primeiro lugar. Em seguida, o código é escrito para satisfazer os testes e, por último, os testes e o código são reestruturados

Desenvolvimento Orientado para Testes de Aceitação (ATDD) (ver Secção 4.5.3):

- Os testes são derivados dos critérios de aceitação como parte do processo de conceção do sistema (Gärtner 2011)
- Os testes são escritos antes de a aplicação ser desenvolvida para satisfazer os testes

Desenvolvimento Orientado para Comportamento (BDD):

- Expressa o comportamento pretendido de uma aplicação com casos de teste escritos numa linguagem simples e natural, que é fácil de compreender pelos *stakeholders* – utilizando o formato Considerando/Quando/Então (Given/When/Then). (Chelimsky 2010)
- Os casos de teste são automaticamente traduzidos para testes executáveis

Em todas as abordagens indicadas acima, os testes podem persistir como testes automatizados para assegurar a qualidade do código nas adaptações/reestruturações futuras.

2.1.4. *DevOps* e Testes

DevOps é uma abordagem organizacional que visa criar sinergias ao promover a colaboração entre o desenvolvimento (incluindo testes) e as operações no sentido de alcançar um conjunto de objetivos comuns. O *DevOps* requer uma mudança cultural dentro da organização para colmatar as lacunas entre o

desenvolvimento (incluindo os testes) e as operações ao mesmo tempo que trata as suas funções com o mesmo valor. Também promove a autonomia da equipa, *feedback* rápido, conjuntos de ferramentas integrados e práticas técnicas como a integração contínua (CI – *Continuous Integration*) e a entrega contínua (CD – *Continuous Delivery*), o que permite às equipas criar, testar e entregar código de alta qualidade mais rapidamente através do *pipeline* de entrega do *DevOps* (Kim 2016).

Na perspetiva dos testes, algumas das vantagens do *DevOps* incluem:

- *Feedback* rápido sobre a qualidade do código, e se as alterações afetam negativamente o código atual
- A CI promove uma abordagem *shift-left* nos testes (ver Secção 2.1.5) ao incentivar os programadores a enviar código de alta qualidade juntamente com testes de componentes e análises estáticas
- Promove processos automatizados como CI/CD que facilitam a criação de ambientes de teste estáveis
- Aumenta a visibilidade das características de qualidade não funcional (p. ex., desempenho, confiabilidade)
- A automação através de um *pipeline* de entrega reduz a necessidade de testes manuais repetitivos
- O risco de regressão é minimizado devido à extensão e ao âmbito dos testes de regressão automatizados

Contudo, o *DevOps* não está isento de riscos e desafios, os quais incluem:

- Definir e estabelecer o *pipeline* de entrega do *DevOps*
- Introduzir e manter as ferramentas de CI/CD
- A automação de testes requer recursos adicionais e pode ser difícil de estabelecer e manter

Apesar de o *DevOps* incluir um nível elevado de testes automatizados, ainda será necessário efetuar testes manuais, especialmente na perspetiva do utilizador.

2.1.5. Abordagem *Shift-Left*

O princípio dos testes antecipados (ver Secção 1.3) é, por vezes, chamado de *shift-left*, uma vez que se trata de uma abordagem onde os testes são efetuados mais cedo no SDLC. *Shift-left* normalmente implica que os testes devem ser efetuados mais cedo (ou seja, não se deve aguardar pela implementação do código ou integração dos componentes). Contudo, isso não significa que se deve negligenciar a realização de testes mais tarde no SDLC.

Existem algumas boas práticas que mostram como se pode obter uma abordagem *shift-left* nos testes, as quais incluem:

- Rever a especificação na perspetiva dos testes. Estas atividades de revisão no âmbito das especificações detetam frequentemente potenciais defeitos, tais como ambiguidades, incompletudes e inconsistências

- Escrever casos de teste antes de escrever o código e executar o mesmo num equipamento de teste durante a implementação do código
- Utilizar a CI e uma melhor CD, dado que fornecem um *feedback* rápido e fornecem testes de componentes automatizados para acompanhar o código-fonte quando este é enviado para o repositório de código
- Concluir a análise estática do código-fonte antes de efetuar o teste dinâmico, ou como parte de um processo automatizado
- Efetuar testes não funcionais no início do nível de teste de componentes, sempre que possível. Isto funciona como uma forma de *shift-left*, uma vez que estes tipos de testes não funcionais tendem a ser efetuados mais tarde no SDLC quando o sistema estiver completo e existir um ambiente de teste representativo

Uma abordagem *shift-left* pode resultar em formação, esforços e/ou custos adicionais no início do processo, mas espera-se que poupe esforços e/ou custos mais tarde no processo.

Na abordagem *shift-left* é importante que os *stakeholders* acreditem e adiram a este conceito.

2.1.6. Retrospetivas e Melhoria de Processos

As retrospetivas (também conhecidas por "reuniões pós-projeto" e retrospetiva do projeto) são normalmente realizadas no final de um projeto ou iteração, num marco de entrega, ou realizadas sempre que necessário. A calendarização e a organização das retrospetivas dependem do modelo de SDLC específico que estiver a ser seguido. Nestas reuniões, os participantes (não só os testadores, mas também os programadores, arquitetos, responsáveis pelos produtos, analistas de negócio) debatem:

- Quais os aspetos que foram bem-sucedidos e devem ser mantidos?
- Quais os aspetos que não foram bem-sucedidos e devem ser melhorados?
- Como incluir as melhorias e manter os aspetos bem-sucedidos no futuro?

Os resultados devem ser registados e normalmente são incluídos no relatório de conclusão do teste (ver Secção 5.3.2). As retrospetivas são importantes para uma implementação bem-sucedida da melhoria contínua, e também é importante que todas as melhorias recomendadas sejam seguidas.

Os benefícios típicos dos testes incluem:

- Aumento da eficácia/eficiência do teste (p. ex.; implementar as sugestões na melhoria de processos)
- Aumento da qualidade de *testware* (p. ex., revisão conjunta do processo de teste)
- Aprendizagem e união da equipa (p. ex., como resultado da oportunidade de colocar questões e propor pontos de melhoria)
- Melhor qualidade da base para testes (p. ex., deficiências no âmbito e na qualidade dos requisitos que podem ser abordadas e resolvidas)
- Melhor colaboração entre o desenvolvimento e os testes (p. ex., a colaboração é revista 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 geridas em conjunto. Cada nível de teste é uma instância do processo de teste efetuada em relação ao software numa determinada fase de desenvolvimento, desde componentes individuais até sistemas completos ou, se aplicável, sistemas de sistemas.

Os níveis de teste estão relacionados com outras atividades dentro do SDLC. Nos modelos de SDLC sequenciais, os níveis de teste são frequentemente definidos de forma que os critérios de saída de um nível façam parte dos critérios de entrada do nível seguinte. Em alguns modelos iterativos isto pode não ser aplicável. As atividades de desenvolvimento podem abranger vários níveis de teste. Os níveis de teste podem sobrepor-se ao longo do tempo.

Os tipos de teste são grupos de atividades de teste relacionados com características de qualidade específicas. A maior parte destas atividades de teste podem ser efetuadas em todos os níveis de teste.

2.2.1. Níveis de Teste

Neste *syllabus* são descritos os seguintes cinco níveis de teste:

- **O teste de componentes** (também conhecido como teste unitário) centra-se em testar os componentes isoladamente e muitas vezes requer suportes específicos, tais como equipamentos de teste ou *frameworks* de teste unitário. O teste de componentes é normalmente efetuado pelos programadores nos respetivos ambientes de desenvolvimento.
- **O teste de integração de componentes** (também conhecido como teste de integração) centra-se em testar as interfaces e interações entre os componentes. O teste de integração de componentes depende fortemente das abordagens na estratégia de integração como ascendente (*bottom-up*), descendente (*top-down*) ou *big-bang*.
- **O teste de sistema** centra-se no comportamento e nas capacidades globais de um sistema ou produto completo, que inclui frequentemente um teste funcional das tarefas de ponta a ponta (do início ao fim) e um teste não funcional das características de qualidade. Para algumas das características de qualidade não funcionais (p. ex., usabilidade), é preferível testá-las num sistema completo com um ambiente de teste representativo. Também é possível utilizar simulações de subsistemas. O teste de sistema pode ser efetuado por uma equipa de teste independente e está relacionado com as especificações do sistema.
- **O teste de integração de sistemas** centra-se em testar as interfaces do sistema em teste, bem como outros sistemas e serviços externos. O teste de integração de sistemas requer ambientes de teste adequados, de preferência semelhantes ao ambiente operacional.
- **O teste de aceitação** centra-se em validar e demonstrar a prontidão para implementação, o que significa que o sistema cumpre as necessidades de negócio do utilizador. Idealmente, o teste de aceitação deve ser efetuado pelos utilizadores previstos. As principais formas do teste de aceitação são: testes de aceitação de utilizador (UAT), testes de aceitação operacional, testes de aceitação contratual e de regulamentação, testes alfa e testes beta.

Os níveis de teste distinguem-se pela seguinte lista de atributos, não exaustiva, para evitar a sobreposição das atividades de teste:

- Objeto de teste
- Objetivos de teste
- Base para testes
- Defeitos e falhas
- Abordagem e responsabilidade

2.2.2. Tipos de Teste

Existem vários tipos de teste que podem ser aplicados nos projetos. Neste *syllabus* são descritos os seguintes quatro tipos de teste:

O teste funcional avalia as funções que devem ser realizadas por um componente ou sistema. As funções são "o que" deve ser efetuado pelo objeto de teste. O objetivo principal do teste funcional é verificar a completude funcional, correção funcional e 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 objetivo principal de um teste não funcional é verificar a qualidade das características não funcionais do software. A norma ISO/IEC 25010 apresenta a seguinte classificação das características não funcionais de qualidade do software:

- Eficiência de desempenho
- Compatibilidade
- Usabilidade
- Confiabilidade
- Segurança
- Manutenibilidade
- Portabilidade

Por vezes, é recomendado que os testes não funcionais comecem mais cedo no ciclo de vida (p. ex., como parte das revisões, dos testes de componentes ou testes de sistema). Muitos dos testes não funcionais são derivados dos testes funcionais, uma vez que utilizam os mesmos testes funcionais e verificam se durante a execução da função a exigência não funcional é satisfeita (p. ex., verificar se uma função é efetuada durante um período de tempo especificado, ou se uma função pode ser migrada para uma nova plataforma). A descoberta tardia de defeitos não funcionais pode representar uma ameaça grave para o sucesso de um projeto. Os testes não funcionais necessitam ocasionalmente de um ambiente de teste muito específico, tal como um laboratório para o teste de usabilidade.

O teste caixa-preta (ver Secção 4.2) é um teste baseado nas especificações e concebe os testes a partir da documentação que é externa ao próprio 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 Secção 4.3) é um teste baseado na estrutura e concebe os testes a partir 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 abranger a estrutura subjacente ao efetuar testes até atingir um nível aceitável de cobertura.

Todos os quatro tipos de teste indicados acima podem ser aplicados a todos os níveis de teste, apesar de o foco ser diferente para cada nível. Podem ser utilizadas várias técnicas de teste para derivar as condições de teste e conceber os casos de teste para todos os tipos de teste indicados.

2.2.3. Teste de Confirmação e Teste de Regressão

Normalmente, são efetuadas alterações a um componente ou sistema para o melhorar, ao adicionar uma nova funcionalidade, ou para o corrigir, ao remover um defeito. Os testes também devem incluir o teste de confirmação e o teste de regressão.

O teste de confirmação confirma se um defeito original foi corrigido com êxito. Dependendo do risco, é possível efetuar um teste da versão corrigida do software de várias formas, incluindo:

- ao executar todos os casos de teste que falharam anteriormente devido ao defeito, ou, também
- adicionar novos testes para abranger todas as alterações necessárias para corrigir o defeito

No entanto, quando existe escassez de tempo ou dinheiro para corrigir os defeitos, os testes de confirmação podem ser limitados apenas à realização dos passos que reproduzem a falha causada pelo defeito e verificar se a falha não ocorre.

O teste de regressão confirma se não surgiram consequências adversas por uma alteração, incluindo uma correção que já foi testada e confirmada. Estas consequências adversas podem afetar o mesmo componente onde foi efetuada a alteração, outros componentes no mesmo sistema, ou inclusive outros sistemas que haja interligação. O teste de regressão pode não estar limitado ao respetivo objeto de teste, mas também pode estar relacionado com o ambiente de teste. É recomendado efetuar primeiro uma análise de impacto para otimizar a extensão do teste de regressão. A análise de impacto mostra as partes do software que podem ser afetadas.

A bateria dos testes de regressão é executada várias vezes e, por norma, o número dos casos de testes de regressão irá aumentar com cada iteração ou lançamento, pelo que o teste de regressão é um forte candidato à automação. A automação destes testes deve começar no início do projeto. Quando a integração contínua é utilizada, tal como em *DevOps* (ver Secção 2.1.4), é considerada boa prática incluir também os testes de regressão automatizados. Dependendo da situação, isto pode incluir os testes de regressão em diferentes níveis.

Os testes de confirmação e/ou testes de regressão do objeto de teste são necessários em todos os níveis de teste se os defeitos forem corrigidos e/ou se forem efetuadas alterações nestes níveis de teste.

2.3. Testes de Manutenção

Existem várias categorias de manutenção, que podem ser corretivas e adaptativas às alterações no ambiente ou melhorar o desempenho ou manutenibilidade (para obter mais detalhes, consulte a norma ISO/IEC 14764), pelo que a manutenção pode implicar implementações/lançamentos planeados e implementações/lançamentos não planeados (*hot fixes*). Pode ser feita uma análise de impacto antes de ser efetuada uma alteração, para ajudar a decidir se a alteração deve ser efetuada, com base nas potenciais consequências noutras áreas do sistema. Testar as alterações num sistema em produção inclui

avaliar o sucesso da implementação da mudança, e verificar a ocorrência de possíveis regressões nas partes do sistema que permanecem inalteradas (por norma, a maior parte do sistema).

O âmbito dos testes de manutenção depende dos seguintes fatores:

- Grau de risco da alteração
- Dimensão do sistema existente
- Dimensão da alteração

Os gatilhos (*triggers*) que motivam a manutenção e os testes de manutenção podem ser classificados da seguinte forma:

- Modificações, como melhorias planeadas (ou seja, baseadas em versões), alterações para correção ou *hot fixes*.
- Atualizações ou migrações do ambiente operacional, como seja de uma plataforma para outra, que pode exigir testes associados ao novo ambiente, bem como do software alterado, ou testes de conversão de dados quando os dados de outra aplicação são migrados para o sistema que está a ser mantido.
- Desativação, que ocorre, muitas das vezes, quando uma aplicação atinge o fim da sua vida útil. Quando um sistema é descontinuado, isso pode exigir testes de arquivamento de dados, se forem necessários períodos longos de retenção de dados. Também pode ser necessário efetuar testes de procedimentos de restauro/recuperação após o arquivamento no caso de serem necessários determinados dados durante o período de arquivo.

3. Testes Estáticos – 80 minutos

Palavras-Chave

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

Objetivos de Aprendizagem para o Capítulo 3:

3.1 Conceitos Básicos dos Testes Estáticos

FL-3.1.1 (K1) Reconhecer tipos de produtos que podem ser examinados através das diferentes técnicas de teste estático

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

FL-3.1.3 (K2) Comparar e diferenciar entre testes estáticos e testes dinâmicos

3.2 *Feedback* e Processo de Revisão

FL-3.2.1 (K1) Identificar as vantagens do *feedback* antecipado e frequente aos *stakeholders*

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

FL-3.2.3 (K1) Recordar as responsabilidades que estão atribuídas às funções principais durante as revisões

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

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

3.1. Conceitos Básicos dos Testes Estáticos

Ao contrário do teste dinâmico, não é necessário executar o software no teste estático. O código, a especificação do processo, a arquitetura do sistema ou os outros produtos de trabalho são avaliados através de um exame manual (p. ex., revisões) ou com a ajuda de uma ferramenta (p. ex., análise estática). Os objetivos de teste incluem melhorar a qualidade, detetar defeitos e avaliar características como legibilidade, completude, correção, testabilidade e consistência. O teste estático pode ser aplicado tanto na verificação como na validação.

Os testadores, representantes do negócio e programadores trabalham em conjunto durante o mapeamento de exemplos, a escrita de *user stories* colaborativas e as sessões de aperfeiçoamento do *backlog* para assegurar que as *user stories* e os produtos de trabalho relacionados cumprem os critérios estabelecidos, p. ex., a Definição de Concluído (Definition of Done) (ver Secção 5.1.3). As técnicas de revisão podem ser aplicadas para assegurar que as *user stories* estão completas, são compreendidas e incluem critérios de aceitação testáveis. Ao colocarem as questões corretas, os testadores podem explorar, desafiar e ajudar a melhorar as *user stories* propostas.

A análise estática permite identificar problemas anteriores ao teste dinâmico, uma vez que requer menos esforço, não necessita de casos de teste e, por norma, recorre ao uso de ferramentas (ver Capítulo 6). A análise estática é frequentemente incorporada nos *frameworks* de integração contínua (ver Secção 2.1.4). Apesar de ser utilizada para detetar defeitos específicos no código, a análise estática também é utilizada para avaliar a manutenibilidade e a segurança do *software*. Os verificadores ortográficos (*spelling checkers*) e as ferramentas de legibilidade são outros exemplos de ferramentas de análise estática.

3.1.1. Produtos de Trabalho Passíveis de Examinação pelos Testes Estáticos

Praticamente qualquer produto de trabalho é passível de ser examinado através dos testes estáticos. Os exemplos incluem documentos de especificação de requisitos, código-fonte, planos de testes, casos de teste, itens do *backlog* do produto, cartas de testes, documentação do projeto, contratos e modelos.

Todos os produtos de trabalhos que possam ser lidos e compreendidos podem ser sujeitos a revisão. Contudo, na análise estática, os produtos de trabalho necessitam de uma estrutura em relação à qual possam ser verificados (p. ex., modelos, código ou texto com sintaxe formal).

Os produtos de trabalho que não são adequados para os testes estáticos incluem os que são difíceis de interpretar por seres humanos e que não devem ser analisados por ferramentas (p. ex., código executável de terceiros devido a motivos jurídicos).

3.1.2. O Valor dos Testes Estáticos

Os testes estáticos permitem detetar defeitos nas fases precoces do SDLC, o que cumpre o princípio dos testes antecipados (ver Secção 1.3). Também permitem identificar defeitos que não são detetados pelos testes dinâmicos (p. ex., código inatingível, padrões de conceção não implementados como pretendido, defeitos em produtos de trabalho não executáveis).

Os testes estáticos permitem avaliar a qualidade e aumentar a confiança nos produtos de trabalho. Ao verificar os requisitos documentados, os *stakeholders* também podem certificar-se de que os respetivos requisitos descrevem as suas necessidades atuais. Tendo em conta que os testes estáticos podem ser efetuados no início do SDLC, é possível gerar um consenso entre os *stakeholders* envolvidos, melhorando também a comunicação entre eles. Por este motivo, é recomendado envolver uma grande variedade de *stakeholders* nos testes estáticos.

Apesar da implementação das revisões poder ser dispendiosa, os custos globais do projeto são normalmente mais baixos do que quando não são efetuadas revisões, uma vez que é necessário despender menos tempo e esforço na correção de defeitos quando comparado com uma fase posterior do projeto.

Os defeitos no código podem ser detetados através da análise estática com maior eficiência do que no teste dinâmico, resultando geralmente em menos defeitos no código e menor esforço de desenvolvimento global.

3.1.3. Diferenças entre Testes Estáticos e Testes Dinâmicos

As práticas dos testes estáticos e dinâmicos complementam-se de forma mútua. Os objetivos são semelhantes, tais como suportar a deteção de defeitos nos produtos de trabalho (ver Secção 1.1.1), contudo, também existem algumas diferenças, como:

- Os testes estáticos e dinâmicos (com análise de falhas) podem resultar na deteção de defeitos, no entanto, existem alguns tipos de defeitos que apenas podem ser encontrados nos testes estáticos ou dinâmicos.
- Os testes estáticos encontram defeitos de forma imediata, enquanto os testes dinâmicos provocam falhas a partir das quais se chega aos defeitos associados, através de análise subsequente
- Os testes estáticos podem detetar mais facilmente os defeitos que se encontram nos caminhos do código, e que são executados raramente ou difíceis de alcançar, através dos testes dinâmicos
- Os testes estáticos podem ser aplicados aos produtos de trabalho não executáveis, enquanto que os testes dinâmicos apenas podem ser aplicados aos produtos de trabalho executáveis
- Os testes estáticos podem ser utilizados para medir as características de qualidade, (p. ex., manutenibilidade) que não dependem do código executado, enquanto que os testes dinâmicos podem ser utilizados para medir as características de qualidade (p. ex., eficiência de desempenho) que dependem do código executado

Os defeitos típicos que são mais fáceis e/ou mais baratos de encontrar e corrigir através dos testes estáticos incluem:

- Defeitos nos requisitos (p. ex., inconsistências, ambiguidades, contradições, omissões, imprecisões e redundâncias)
- Defeitos de conceção (p. ex., estruturas de bases de dados ineficientes, modularização inadequada)
- Determinados tipos de defeitos no código (p. ex., variáveis com valores indefinidos, variáveis não declaradas, código inacessível ou duplicado, complexidade excessiva do código)
- Desvios em relação às normas (p. ex., falta de cumprimento das convenções de nomenclatura nas normas de codificação)
- Especificações de interface incorretas (p. ex., números, tipos ou ordem de parâmetros incompatíveis)
- Tipos específicos de vulnerabilidades de segurança (p. ex, *buffer overflows*)

- Lacunas ou imprecisões na cobertura da base para testes (p. ex., testes em falta para um critério de aceitação)

3.2. *Feedback* e Processo de Revisão

3.2.1. As Vantagens do *Feedback* Antecipado e Frequente dos *Stakeholders*

O *feedback* antecipado e frequente permite comunicar os potenciais problemas de qualidade mais cedo. Se existir pouca participação dos *stakeholders* durante o SDLC, o produto que está a ser desenvolvido poderá não satisfazer a visão original ou atual desses *stakeholders*. O incumprimento na satisfação das necessidades dos *stakeholders* pode resultar em retrabalhos dispendiosos, prazos falhados, jogos de atribuição de culpas e pode mesmo levar a um fracasso total do projeto.

Um *feedback* frequente dos *stakeholders* ao longo do SDLC pode evitar equívocos sobre os requisitos e assegurar que as alterações efetuadas aos requisitos são compreendidas e implementadas antecipadamente. Isto ajuda a que a equipa de desenvolvimento compreenda melhor o que está a ser construído. Também permite à equipa concentrar-se nas características que proporcionam um maior valor aos *stakeholders* e que tenham um impacto, o mais positivo possível, 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 uma *framework* estruturada, mas flexível, a partir do qual um processo de revisão específico pode ser adaptado a uma determinada situação. Se for necessário realizar uma revisão mais formal, significa que será necessário efetuar ainda mais tarefas nas atividades descritas.

A dimensão de muitos produtos de trabalho torna-os demasiado grandes para serem abrangidos por uma única revisão. Assim, o processo de revisão pode ser aplicado várias vezes para se poder concluir a revisão do produto de trabalho na totalidade.

As atividades no processo de revisão são:

- **Planeamento.** Durante a fase de planeamento, será definido o âmbito da revisão, que inclui o objetivo, o produto de trabalho a rever, as características de qualidade a avaliar, as áreas de foco, os critérios de saída, as informações de suporte, tais como as normas, o esforço e os prazos da revisão.
- **Início da revisão.** Durante o início da revisão, o objetivo é assegurar que tudo e todos os envolvidos estão preparados para iniciar a revisão. Isto inclui assegurar que todos os participantes têm acesso ao produto de trabalho em revisão, compreendem a sua função e as suas responsabilidades e recebem tudo o que necessitam para efetuar a revisão.
- **Revisão individual.** Todos os revisores efetuam uma revisão individual para avaliar a qualidade do produto de trabalho em revisão, identificar anomalias, recomendações e questões, aplicando uma ou mais técnicas de revisão (p. ex., revisão baseada em *checklists*, revisão com base em cenários). A norma ISO/IEC 20246 fornece mais informações sobre as diferentes técnicas de revisão. Os revisores registam todas as anomalias, recomendações e questões que identificarem.
- **Comunicação e análise.** Uma vez que as anomalias identificadas durante a revisão não são necessariamente defeitos, todas as anomalias identificadas têm de ser analisadas e debatidas.

Para cada anomalia, a decisão deve ser tomada com base no seu estado, de quem detém a posse da anomalia e das ações necessárias. Normalmente, isto é feito numa reunião de revisão, na qual os participantes também decidem o nível de qualidade do produto de trabalho revisto e as ações subsequentes que serão necessárias. Poderá ser necessário efetuar uma revisão de seguimento (*follow-up*) para concluir as ações.

- **Correção e elaboração de relatórios.** Para cada defeito, deve ser criado um relatório de defeitos para dar seguimento às ações corretivas. Após concretizar os critérios de saída, o produto de trabalho pode ser aceite e os resultados da revisão podem ser comunicados.

3.2.3. Funções e Responsabilidades nas Revisões

As revisões implicam vários *stakeholders*, que podem assumir muitas funções. As funções e as responsabilidades principais são:

- **Gestor** – Decide o que deve ser revisto e fornece os recursos, tais como os colaboradores e a alocação de tempo necessária para efetuar a revisão
- **Autor** – Cria e corrige o produto de trabalho em revisão
- **Moderador** (também conhecido como **Facilitador**) – Assegura um funcionamento eficaz das reuniões de revisão, incluindo a mediação e a gestão do tempo, bem como um ambiente de revisão seguro onde todos possam participar livremente
- **Redator** (também conhecido como **Anotador**) – Reúne anomalias dos revisores e regista as informações da revisão, tais como as decisões e as novas anomalias encontradas durante a reunião de revisão
- **Revisor** – efetua as revisões. Um revisor pode ser alguém que está a trabalhar no projeto, um especialista na matéria, ou qualquer outro *stakeholder*
- **Líder de revisão** – Assume responsabilidade global pela revisão, tal como decidir quem estará envolvido e define quando e onde será efetuada a revisão

Também são possíveis funções mais detalhadas, tal como descrito na norma ISO/IEC 20246.

3.2.4. Tipos de Revisão

Existem vários tipos de revisão desde as revisões informais até às revisões formais. O nível de formalidade necessário depende de fatores como o SDLC que está a ser seguido, a maturidade do processo de desenvolvimento, a criticidade e complexidade do produto de trabalho que está a ser revisto, requisitos legais ou regulamentares e a necessidade de um rastreio auditável. O mesmo produto de trabalho pode ser revisto com diferentes tipos de revisão, p. ex., primeiro uma revisão informal e, posteriormente, uma revisão formal.

A seleção do tipo de revisão mais adequado é essencial para concretizar os objetivos da revisão (ver Secção 3.2.5). A seleção não se deve basear apenas nos objetivos, mas também em fatores como as necessidades do projeto, os recursos disponíveis, o tipo e os riscos do produto de trabalho, área de negócio e cultura da empresa.

Alguns tipos de revisão utilizados são:

- **Revisão Informal.** As revisões informais não seguem um processo definido e não necessitam de um resultado formal documentado. O objetivo principal é detetar anomalias.
- ***Walkthrough*.** O *walkthrough*, liderado pelo autor, pode ter vários objetivos, tais como avaliar a qualidade e aumentar a confiança no produto de trabalho, educar os revisores, chegar a um consenso, gerar novas ideias, motivar e permitir que os autores possam efetuar melhorias e detetar anomalias. Os revisores podem efetuar uma revisão individual antes do *walkthrough*, mas tal não é obrigatório.
- **Revisão técnica.** A revisão técnica é efetuada por revisores técnicos qualificados e liderada por um moderador. Os objetivos da revisão técnica são chegar a um consenso e tomar decisões relativamente a problemas técnicos, mas também para detetar anomalias, avaliar a qualidade e aumentar a confiança no produto de trabalho, gerar novas ideias, bem como motivar e permitir que os autores possam efetuar melhorias.
- **Inspeção.** As inspeções são o tipo de revisão mais formal, uma vez que seguem o processo genérico de forma completa (ver Secção 3.2.2). O objetivo principal é encontrar o maior número possível de anomalias. Os outros objetivos consistem em avaliar a qualidade, aumentar a confiança no produto de trabalho, bem como motivar e permitir que os autores possam efetuar melhorias. São recolhidas e utilizadas métricas para melhorar o SDLC, mas também para melhorar o processo de inspeção em si. Nas inspeções, o autor não pode agir como o líder de revisão ou redator.

3.2.5. Fatores de Sucesso para as Revisões

Existem vários fatores que determinam o sucesso das revisões, incluindo:

- Definir objetivos claros e critérios mensuráveis de saída. A avaliação dos participantes nunca deve ser um objetivo
- Escolher o tipo de revisão adequado para concretizar os objetivos pretendidos e adaptar ao tipo de produto de trabalho, aos participantes da revisão, às necessidades e ao contexto do projeto
- Realizar revisões em partes pequenas, de modo que os revisores não percam a concentração durante a revisão individual e/ou a reunião de revisão (quando realizada)
- Fornecer *feedback* das revisões aos *stakeholders* e aos autores para que possam melhorar o produto e as respetivas atividades (ver Secção 3.2.1)
- Fornecer tempo adequado aos participantes para se preparem para a revisão
- Receber suporte da gestão para o processo de revisão
- Tornar as revisões parte da cultura da organização, para promover a aprendizagem e a melhoria de processos
- Fornecer formação adequada a todos os participantes para que saibam como desempenhar a sua função
- Facilitar as reuniões

4. Análise e Conceção de Testes – 390 minutos

Palavras-Chave

abordagens de teste baseadas na colaboração, análise de valor fronteira, antecipação de erros, cobertura, cobertura de instruções, cobertura de ramos, critérios de aceitação, desenvolvimento orientado para testes de aceitação, item de cobertura, particionamento por equivalências, técnica de teste, técnica de teste baseada na experiência, técnica de teste caixa-branca, técnica de teste caixa-preta, teste baseado em *checklists*, teste de tabelas de decisão, teste de transição de estados, teste exploratório

Objetivos de Aprendizagem para o Capítulo 4:

4.1 Descrição Geral das Técnicas de Teste

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

4.2 Técnicas de Teste Caixa-preta

FL-4.2.1 (K3) Utilizar o particionamento por equivalências para obter casos de teste

FL-4.2.2 (K3) Utilizar a análise de valor fronteira para obter casos de teste

FL-4.2.3 (K3) Utilizar teste de tabelas de decisão para obter casos de teste

FL-4.2.4 (K3) Utilizar teste de transição de estados para obter casos de teste

4.3 Técnicas de Teste Caixa-branca

FL-4.3.1 (K2) Explicar o teste de instruções

FL-4.3.2 (K2) Explicar o teste de ramos

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 antecipação de erros

FL-4.4.2 (K2) Explicar o teste exploratório

FL-4.4.3 (K2) Explicar o teste baseado em *checklists*

4.5 Abordagens de Teste Baseadas na Colaboração

FL-4.5.1 (K2) Explicar como escrever *user stories* em colaboração com programadores e representantes do negócio

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

FL-4.5.3 (K3) Utilizar o desenvolvimento orientado para testes de aceitação (ATDD) para obter casos de teste

4.1. Descrição Geral das Técnicas de Teste

As técnicas de teste suportam o testador na análise de teste (o que testar?) e na conceção de teste (como testar?). As técnicas de teste contribuem para o desenvolvimento de um conjunto relativamente pequeno, mas suficiente, de casos de teste de uma forma sistemática. As técnicas de teste também ajudam o testador a definir as condições de teste, bem como a identificar os itens de cobertura e os dados de teste durante a análise e a conceção do teste. Para obter mais informações sobre as técnicas de teste e as respetivas medidas, consulte a norma ISO/IEC/IEEE 29119-4, e (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 nas especificações) são baseadas na análise do comportamento especificado do objeto de teste, sem referência à estrutura interna. Por conseguinte, os casos de teste não dependem da forma como o software é implementado. Consequentemente, se existirem alterações na implementação, mas o comportamento necessário permanecer o mesmo, os casos de teste ainda são úteis.

As técnicas de teste caixa-branca (também conhecidas como técnicas baseadas na estrutura) são baseadas na análise da estrutura e do processamento interno do objeto de teste. Uma vez que os casos de teste dependem da forma como o software foi concebido, os mesmos apenas podem ser criados após a conceção ou implementação do objeto de teste.

As técnicas de teste baseadas na experiência utilizam, de forma efetiva, o conhecimento e a experiência dos testadores na conceção e implementação dos casos de teste. O nível de eficácia destas técnicas depende imenso das competências do testador. As técnicas de teste baseadas na experiência permitem detetar defeitos que podem passar despercebidos utilizando as técnicas de teste caixa-preta e caixa-branca. Desta forma, as técnicas de teste baseadas na experiência servem de complemento às técnicas de teste caixa-preta e caixa-branca.

4.2. Técnicas de Teste Caixa-Preta

As técnicas de teste caixa-preta frequentemente utilizadas e abordadas nas seguintes secções são:

- Particionamento por Equivalências
- Análise de Valor Fronteira
- Teste de Tabelas de Decisão
- Teste de Transição de Estados

4.2.1. Particionamento por Equivalências

O particionamento por equivalências (EP) divide os dados em partições (conhecidas como partições de equivalência) com base na expetativa de que todos os elementos de uma determinada partição deverão ser processados da mesma forma pelo objeto de teste. A teoria subjacente a esta técnica é que se um caso de teste, que testa um valor de uma partição de equivalência, detetar um defeito, o mesmo também deve ser detetado pelos casos de teste que testam os outros valores da mesma partição. Por conseguinte, é suficiente um teste para cada partição.

As partições de equivalência podem ser identificadas para qualquer dado relacionado com o objeto de teste, incluindo dados de entrada, saída, dados de configuração, valores internos, valores relacionados com o tempo e parâmetros de interface. As partições podem ser contínuas ou discretas, ordenadas ou não ordenadas, finitas ou infinitas. As partições não se podem sobrepor e têm de ser conjuntos não vazios.

As EP de objetos de teste simples podem ser fáceis, mas na prática, compreender a forma como o objeto de teste irá tratar os diferentes valores é muitas vezes complicado. Por conseguinte, a partição deve ser efetuada com atenção.

Uma partição de equivalência que contenha valores válidos é designada por partição válida. Uma partição de equivalência com valores inválidos é designada por partição inválida. As definições dos valores válidos e inválidos podem variar entre as equipas e as organizações. Por exemplo, os valores válidos podem ser interpretados como os valores que devem ser processados pelo objeto de teste ou como os valores em que a especificação define o processamento esperado. Os valores inválidos podem ser interpretados como os valores que devem ser ignorados ou rejeitados pelo objeto de teste ou como os valores em que não existe nenhum processamento definido na especificação do objeto de teste.

Na EP, os itens de cobertura são as partições de equivalência. Para alcançar 100% de cobertura com esta técnica, os casos de teste têm de executar todas as partições identificadas (incluindo partições 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, normalmente expresso em percentagem.

Muitos objetos de teste incluem vários conjuntos de partições (p. ex., objetos de teste com mais do que um parâmetro de entrada), o que significa que um caso de teste irá cobrir as partições de diferentes conjuntos de partições. O critério de cobertura mais simples no caso de vários conjuntos de partições é designado por cobertura de Cada Escolha (Ammann 2016). A cobertura de Cada Escolha (*Each Choice coverage*) requer que os casos de teste executem cada partição de cada conjunto de partições pelo menos uma vez. A cobertura de Cada Escolha não tem em conta as combinações das partições.

4.2.2. Análise de Valor Fronteira

A análise de valor fronteira (BVA – *Boundary Value Analysis*) é uma técnica baseada na execução das fronteiras das partições de equivalência. Por conseguinte, a BVA apenas pode ser utilizada nas partições ordenadas. Os valores mínimos e máximos de uma partição são os seus valores fronteira. No caso da BVA, se dois elementos pertencerem à mesma partição, todos os elementos entre eles também têm de pertencer a essa partição.

A BVA concentra-se nos valores fronteira das partições, uma vez que existe uma probabilidade muito maior de os programadores cometerem erros com estes valores fronteira. Os defeitos típicos encontrados pela BVA estão localizados onde a implementação das fronteiras está mal colocada, ocupando posições acima ou abaixo das posições pretendidas ou, então, as fronteiras estão totalmente omissas.

Este *syllabus* aborda duas versões da BVA: BVA de 2 valores e de 3 valores, as quais diferem em termos de itens de cobertura por fronteira, obrigando à execução de diferentes fronteiras para se obter 100% de cobertura.

Na BVA de 2 valores (Craig 2002, Myers 2011), para cada valor fronteira existem dois itens de cobertura: este valor fronteira e o seu valor vizinho mais próximo que pertencem à partição adjacente. Para obter 100% cobertura com a BVA de 2 valores, os casos de teste têm de executar todos os itens de cobertura, ou seja, todos os valores fronteira identificados. A cobertura é medida como o número de valores fronteira

que foram executados, dividido pelo número total de valores fronteira identificados, normalmente expresso em percentagem.

Na BVA de 3 valores (Koomen 2006, O'Regan 2019), para cada valor fronteira existem três itens de cobertura: este valor fronteira e os seus valores vizinhos mais próximos. Por conseguinte, na BVA de 3 valores alguns dos itens de cobertura podem não ser valores fronteira. Para obter 100% cobertura com a BVA de 3 valores, os casos de teste têm de executar todos os itens de cobertura, ou seja, todos os valores fronteira identificados e os seus valores vizinhos. A cobertura é medida como o número de valores fronteira onee os respetivos valores vizinhos que foram executados, dividido pelo número total de valores fronteira identificados e os seus valores vizinhos, normalmente expresso em percentagem.

A BVA de 3 valores é mais rigorosa do que a BVA de 2 valores, uma vez que pode detetar defeitos negligenciados pela BVA de 2 valores. Por exemplo, se a decisão "se (x ≤ 10) …" estiver implementada incorretamente como "se (x = 10) …", nenhum dos dados de teste derivados da BVA de 2 valores (x = 10, x = 11) conseguem detetar o defeito. Contudo, x = 9, derivado da BVA de 3 valores, irá provavelmente detetar o defeito.

4.2.3. Teste de Tabelas de Decisão

As tabelas de decisão são utilizadas para testar a implementação dos requisitos de sistema quando estes especificam que diferentes combinações de condições de entrada resultam em diferentes resultados à saída. As tabelas de decisão são uma forma eficaz de sistematizar uma lógica complexa, como pode ser o caso da implementação de regras de negócio.

Ao criar as tabelas de decisão, começa-se por definir as condições e as ações resultantes do sistema. Estas 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 com entradas limitadas, todos os valores das condições e ações (exceto os irrelevantes ou inexequíveis; ver infra) são apresentados como valores booleanos (verdadeiro ou falso). Em alternativa, numa tabela de decisão com entradas alargadas, algumas ou todas as condições e ações podem também assumir vários valores (p. ex., intervalos de números, partições de equivalência, valores discretos).

A notação das 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 inexequível para uma determinada regra. Para ações: "X" significa que a ação deve ocorrer. Em branco significa que a ação não deve ocorrer. Também podem ser utilizadas outras notações.

Uma tabela de decisão completa tem colunas suficientes para cobrir todas as combinações de condições. A tabela pode ser simplificada ao eliminar as colunas que contêm combinações inexequíveis de condições. A tabela também pode ser minimizada ao juntar colunas, nas quais algumas das condições não afetam o resultado, numa única coluna. Os algoritmos de minimização da tabela de decisão estão fora do âmbito deste syllabus.

No teste de tabelas de decisão, os itens de cobertura são as colunas que contêm combinações de condições exequíveis. Para obter 100% cobertura com esta técnica, os casos de teste têm de executar todas estas colunas. A cobertura é medida como o número de colunas executadas, dividido pelo número total de colunas exequíveis, normalmente expresso em percentagem.

A importância do teste de tabelas de decisão é proporcionar uma abordagem sistemática para identificar todas as combinações de condições, algumas das quais poderiam, de outra forma, ser negligenciadas. Esta técnica também ajuda a encontrar eventuais lacunas ou contradições nos requisitos. Se existirem

muitas condições, a execução de todas as regras de decisão pode ser demorada, uma vez que o número de combinações cresce exponencialmente com o número de condições. Neste caso, para reduzir o número de regras que precisam de ser executadas, pode ser utilizada uma tabela de decisão minimizada ou uma abordagem baseada na avaliação do risco.

4.2.4. Teste de Transição de Estados

O diagrama de transição de estados modela o comportamento de um sistema ao mostrar os possíveis estados e as transições de estados válidas. Uma transição é iniciada por um evento, que pode ser adicionalmente qualificada por uma condição de guarda. As transições são consideradas instantâneas e podem, por vezes, levar o software a tomar uma ação. A sintaxe comum para a etiquetagem da transição é a seguinte: "evento [condição de guarda] / ação". As condições de guarda e as ações podem ser omitidas se não existirem ou forem irrelevantes para o testador.

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

Um caso de teste num diagrama de transição de estados ou numa tabela de estados é representado como uma sequência de eventos, que resultam numa sequência de alterações de estados (e ações, se aplicável). Um caso de teste pode e, por norma, irá abranger várias transições entre os estados.

Existem imensos critérios de cobertura para o teste de transição de estados. Este *syllabus* aborda três destes critérios.

Na **cobertura de todos os estados**, os itens de cobertura são os estados. Para obter 100% de cobertura em todos os estados, os casos de teste têm de assegurar que todos os estados são percorridos. A cobertura é medida como o número de estados percorridos, dividido pelo número total de estados, normalmente expresso em percentagem.

Na **cobertura de transições válidas** (também denominada de cobertura 0-switch), os itens de cobertura são as transições válidas numa perspetiva individual. Para obter 100% de cobertura de transições válidas, os casos de teste têm de executar todas as transições válidas. A cobertura é medida como o número de transições válidas executadas, dividido pelo número total de transições válidas, normalmente expresso em percentagem.

Na **cobertura de todas as transições**, os itens de cobertura são todas as transições apresentadas numa tabela de estados. Para obter 100% de cobertura de todas as transições, os casos de teste têm de executar não só todas as transições válidas, mas também tentar executar as transições inválidas. Efetuar o teste de uma única transição inválida num só caso de teste ajuda a evitar o encobrimento de defeitos, ou seja, uma situação em que um defeito impede a deteção de outro defeito. A cobertura é medida como o número de transições válidas e inválidas executadas ou com tentativa de execução, dividido pelo número total de transições válidas e inválidas, normalmente expresso em percentagem.

A cobertura de todos os estados é mais fraca do que a cobertura de transições válidas, uma vez que pode ser normalmente obtida sem executar todas as transições. A cobertura de transições válidas é o critério de cobertura mais utilizado. A cobertura total das transições válidas assegura a cobertura total de todos os estados. A cobertura de todas as transições assegura a cobertura total de todos os estados e a cobertura total de transições válidas, e deve ser um requisito mínimo para o software crítico em termos de funcionalidade e de segurança (*mission and safety critical software*).

4.3. Técnicas de Teste Caixa-Branca

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

- Teste de instruções
- Teste de ramos

Existem técnicas mais rigorosas que são utilizadas em alguns ambientes críticos em segurança, críticos em funcionalidades (*mission-critical*) ou que exigem ambientes de elevada integridade, permitindo obter uma cobertura de código mais abrangente. Também existem técnicas de teste caixa-branca utilizadas em níveis de teste mais elevados (p. ex., testes de API), ou utilizando uma medida de cobertura não relacionada com o código (p. ex., cobertura de neurónios num teste de rede neuronal). Estas técnicas não são abordadas neste *syllabus*.

4.3.1. Teste de Instruções e Cobertura de Instruções

No teste de instruções, os itens de cobertura são as instruções executáveis. O objetivo é conceber casos de teste que executem instruções no código até ser obtido um nível de cobertura aceitável. A cobertura é medida como o número de instruções executadas pelos casos de teste, divido pelo número total de instruções executáveis no código, normalmente expresso em percentagem.

Quando se obtém 100% de cobertura de instruções, é possível assegurar que todas as instruções executáveis do código foram executadas, pelo menos, uma vez. Isto significa, nomeadamente, que cada instrução com um defeito será executada, o que pode causar uma falha e demonstrar a presença desse defeito. Contudo, a execução de uma instrução com um caso de teste não irá detetar defeitos em todos os casos. Por exemplo, pode não detetar defeitos que são dependentes de dados (p. ex., uma divisão por zero que apenas falha quando o denominador está definido como zero). Além disso, ter 100% cobertura de instruções não assegura que todas as decisões lógicas foram testadas como, por exemplo, será o caso se não executar todos os ramos do código (ver Capítulo 4.3.2).

4.3.2. Teste de Ramos e Cobertura de Ramos

O ramo é uma transferência de controlo entre dois nós no grafo de fluxo de controlo, que apresenta as sequências possíveis onde as instruções do código-fonte são executadas no objeto de teste. Cada transferência de controlo pode ser incondicional (ou seja, código linear) ou condicional (ou seja, resultado da decisão).

No teste de ramos, os itens de cobertura são ramos que têm como objetivo conceber casos de teste para executar os ramos no código até ser obtido um nível de cobertura aceitável. A cobertura é medida como o número de ramos executados pelos casos de teste, dividido pelo número total de ramos, normalmente expresso em percentagem.

Quando se obtém 100% de cobertura de ramos, todos os ramos no código, incondicionais e condicionais, são executados pelos casos de teste. Os ramos condicionais correspondem normalmente a um resultado de verdadeiro ou falso a partir de uma decisão "se... então", um resultado a partir de uma instrução de caso/*switch*, ou uma decisão para sair ou continuar num ciclo. Contudo, a execução de um ramo com um caso de teste não irá detetar defeitos em todos os casos. Por exemplo, pode não detetar defeitos que requerem a execução de um caminho específico no código.

A cobertura de ramos inclui a cobertura de instruções. Isto significa que qualquer conjunto de casos de teste que obtenha 100% cobertura de ramos também obtém 100% cobertura de instruções (mas não o inverso).

4.3.3. O Valor do Teste Caixa-Branca

Um ponto forte fundamental que todas as técnicas de teste caixa-branca partilham é o facto de toda a implementação do software ser tida em conta durante os testes, o que facilita a deteção de defeitos mesmo quando a especificação de software é vaga, está desatualizada ou incompleta. Um ponto fraco correspondente é quando o software não implementa um ou mais requisitos, o teste caixa-branca pode não detetar os defeitos que resultem dessa omissão (Watson 1996).

As técnicas de teste caixa-branca podem ser utilizadas no teste estático (p. ex., durante um esboço da execução do código). Estas técnicas são bastante adequadas para a revisão do código que ainda não está pronto para execução (Hetzel 1988), bem como para a revisão de pseudocódigo e de outras lógicas de alto nível ou lógicas *top-down* que podem ser modeladas com um grafo de fluxo de controlo.

A realização de apenas testes caixa-preta não fornece uma medida da cobertura de código. As medidas de cobertura de caixa-branca fornecem uma medição objetiva da cobertura e disponibilizam as informações necessárias para permitir a criação de testes adicionais para aumentar esta cobertura e, subsequentemente, aumentar a confiança no código.

4.4. Técnicas de Teste Baseadas na Experiência

As técnicas de teste baseadas na experiência frequentemente utilizadas e abordadas nas seguintes secções são:

- Antecipação de erros
- Teste exploratório
- Teste baseado em *checklists*

4.4.1. Antecipação de Erros

A antecipação de erros é uma técnica utilizada para antecipar a ocorrência de erros, defeitos e falhas, com base no conhecimento do testador, incluindo:

- A forma como a aplicação funcionou no passado
- Os tipos de erros que tendem a ser cometidos pelos programadores e os tipos de defeitos que resultam desses erros
- Os tipos de falhas que ocorreram noutras aplicações semelhantes

No geral, os erros, os defeitos e as falhas podem estar relacionadas com: entrada (p. ex., entrada correta não aceite, parâmetros incorretos ou em falta), saída (p. ex., formato incorreto, resultado incorreto), lógica (p. ex., falta de um caso, operador incorreto), computação (p. ex., incorreta operação ou computação incorreta), interfaces (p. ex., parâmetros errados, tipos incompatíveis), ou dados (p. ex., inicialização incorreta, tipo incorreto).

Os ataques a falhas são uma abordagem metódica para implementar a antecipação de erros. Esta técnica requer que o testador crie ou adquira uma lista de erros, defeitos e falhas possíveis, e conceba testes que irão identificar os defeitos associados aos erros, expor esses defeitos, ou causar as falhas respetivas. Estas listas podem ser construídas com base na experiência, com base em dados de defeitos e de falhas, ou a partir do conhecimento comum sobre os motivos do software falhar.

Para obter mais informações sobre a antecipação de erros e os ataques a falhas, consulte (Whittaker 2002, Whittaker 2003, Andrews 2006).

4.4.2. Teste Exploratório

No teste exploratório, os testes são concebidos, executados e avaliados simultaneamente, enquanto o testador aprende mais sobre o objeto de teste. Os testes são utilizados para obter mais informações sobre o objeto de teste, para explorá-lo mais profundamente com testes focados e criar testes para as áreas não testadas.

Por vezes, o teste exploratório é efetuado utilizando testes baseados em sessões para estruturar o teste. Na abordagem baseada em sessões, os testes exploratórios são efetuados dentro de uma delimitação de tempo definida (*time-boxing*) e o testador utiliza uma carta de testes com objetivos de teste específicos para o orientar nos testes. A sessão de teste é normalmente seguida de uma reunião de balanço (*debrief*) que inclui um debate entre o testador e os *stakeholders* interessados nos resultados da sessão de teste. Nesta abordagem, os objetivos de 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 utilizar as folhas de sessão de teste para documentar os passos seguidos e as descobertas feitas.

Os testes exploratórios são úteis quando existem poucas especificações ou especificações inadequadas, ou existe uma pressão significativa do tempo reservado ao teste. Também são úteis para complementar outras técnicas de teste mais formais. Os testes exploratórios serão mais eficazes se o testador tiver experiência, possuir conhecimentos da área de negócio e um elevado grau de competências essenciais, tais como competências analíticas, curiosidade e criatividade (ver Secção 1.5.1).

Os testes exploratórios podem incorporar a utilização de outras técnicas de teste (p. ex., particionamento por equivalências). Para obter mais informações sobre os testes exploratórios, consulte (Kaner 1999, Whittaker 2009, Hendrickson 2013).

4.4.3. Teste Baseado em *Checklists*

No teste baseado em *checklists*, o testador concebe, implementa e executa os testes para abranger as condições de teste encontradas numa lista de verificação (*checklist*). Estas *checklists* podem ser criadas com base na experiência, no conhecimento sobre o que é importante para o utilizador ou na compreensão do porquê e como o software falha. As *checklists* não devem conter itens que possam ser verificados automaticamente, itens que funcionem melhor como critérios de entrada/saída, ou itens demasiado abrangentes (Brykczynski 1999).

Os itens da *checklist* são frequentemente formulados sob a forma de uma pergunta. Também deve ser possível verificar cada item de forma separada e direta. Estes itens podem referir-se a requisitos, propriedades da interface gráfica, características de qualidade ou outras formas de condições de teste. As *checklists* podem ser criadas para suportar vários tipos de teste, incluindo testes funcionais e não funcionais (p. ex., 10 heurísticas para os testes de usabilidade (Nielsen 1994)).

Algumas entradas da *checklist* podem tornar-se gradualmente menos eficazes ao longo do tempo, uma vez que os programadores irão aprender a evitar cometer os mesmos erros. Pode também ser necessário adicionar novas entradas para refletir os novos defeitos de elevada gravidade encontrados. Por conseguinte, as listas de verificação devem ser atualizadas regularmente com base na análise de defeitos. Contudo, é necessário ter atenção para evitar que a *checklist* se torne demasiado grande (Gawande 2009).

Na ausência de casos de teste detalhados, os testes baseados em *checklists* podem fornecer orientações e algum grau de consistência para os testes. Se as listas de verificação forem de alto nível, é provável que ocorra alguma variabilidade nos testes reais, resultando numa cobertura potencialmente maior, mas com menor repetibilidade.

4.5. Abordagens de Teste Baseadas na Colaboração

Cada uma das técnicas indicadas acima (ver Secções 4.2, 4.3 e 4.4) tem um objetivo específico no que diz respeito à deteção de defeitos. Por outro lado, as abordagens baseadas na colaboração, também se concentram em evitar os defeitos através da colaboração e comunicação.

4.5.1. Escrita de *User Stories* Colaborativas

A *user story* representa uma característica que será valiosa para o utilizador ou o comprador do sistema ou software. As *user stories* incluem três aspetos importantes (Jeffries 2000), denominados como os "3 C's":

- Cartão – O suporte que descreve uma *user story* (p. ex., um cartão indexado, uma entrada num quadro eletrónico)
- Conversação – Explica a forma como o software será utilizado (pode ser documentada ou verbal)
- Confirmação – Os critérios de aceitação (ver Secção 4.5.2)

O formato mais utilizado numa *user story* é "Enquanto [função], pretendo [objetivo a concretizar], para poder [valor de negócio resultante da função]", seguido dos critérios de aceitação.

A autoria colaborativa da *user story* pode utilizar técnicas como o *brainstorming* e o *mind mapping*. A colaboração permite à equipa obter uma visão partilhada do que deve ser fornecido, ao ter em conta três perspetivas: negócio, desenvolvimento e testes.

Uma boa *user story* deve ser: independente, negociável, valiosa, estimável, pequena e testável (INVEST). Se um *stakeholder* não souber como testar uma *user story*, poderá indicar que a *user story* não é clara o suficiente, ou não reflete algo de valioso, ou que o *stakeholder* apenas necessita de ajuda no teste (Wake 2003).

4.5.2. Critérios de Aceitação

Os critérios de aceitação de uma *user story* são as condições que uma implementação da *user story* tem de cumprir para ser aceite pelos *stakeholders*. Nesta perspetiva, os critérios de aceitação podem ser considerados como as condições de teste que devem ser executadas pelos testes. Os critérios de aceitação são, por norma, o resultado de uma Conversação (ver Secção 4.5.1).

Os critérios de aceitação são utilizados para:

- Definir o âmbito da *user story*

- Obter um consenso entre os *stakeholders*
- Descrever os cenários positivos e negativos
- Servir como base para o teste de aceitação da *user story* (ver Secção 4.5.3)
- Permitir estimativas e planeamentos exatos

Existem várias formas de escrever os critérios de aceitação para uma *user story*. Os dois formatos mais comuns são:

- Orientado para os cenários (p. ex., o formato Dado que/Quando/Então (Given/When/Then) utilizado em BDD, ver Secção 2.1.3)
- Orientado para as regras (p. ex., *checklist* com tópicos, ou tabela com mapeamento entre entrada-saída)

A maior parte dos critérios de aceitação podem ser documentados num destes dois formatos. Contudo, a equipa pode utilizar outro formato personalizado, desde que os critérios de aceitação estejam bem definidos e sejam inequívocos.

4.5.3. Desenvolvimento Orientado para Testes de Aceitação (ATDD)

O ATDD é uma abordagem de teste inicial (ver Secção 2.1.3). Os casos de teste são criados antes da implementação da *user story*. Os casos de teste são criados pelos membros da equipa com perspetivas diferentes, p. ex., clientes, programadores e testadores (Adzic 2009). A execução dos casos de teste pode ser manual ou automatizada.

O primeiro passo é um *workshop* de especificação, no qual a *user story* e (se ainda não estiverem definidos) os respetivos critérios de aceitação são analisados, debatidos e escritos pelos membros da equipa. Quaisquer sinais de incompletude, ambiguidades ou defeitos na *user story* são resolvidos durante este processo. O passo seguinte consiste na criação dos casos de teste. Isto pode ser efetuado em conjunto, por toda a equipa, ou individualmente, pelo testador. Os casos de teste baseiam-se nos critérios de aceitação e podem ser vistos como exemplos de funcionamento do software. Isto irá ajudar a equipa a implementar a *user story* corretamente.

Uma vez que os termos “exemplo” e o “teste” significam o mesmo, estes termos são geralmente utilizados de forma indiferenciada. Podem ser aplicadas as técnicas de teste descritas nas Secções 4.2, 4.3 e 4.4 durante a conceção de teste.

Geralmente, os primeiros casos de teste são positivos, que confirmam o comportamento correto sem exceções ou condições de erro, e são compostos pela sequência de atividades executadas se tudo correr dentro do esperado. Após a conclusão dos casos de teste positivos, a equipa deve efetuar os testes negativos. Por último, a equipa deve abranger também as características de qualidade não funcionais (p. ex., eficiência no desempenho, usabilidade). Os casos de teste devem ser expressos de uma forma compreensível para os *stakeholders*. Normalmente, os casos de teste contêm frases em linguagem natural que envolvem as pré-condições necessárias (caso existam), as entradas e as pós-condições.

Os casos de teste têm de abranger todas as características da *user story* e não devem exceder o seu conteúdo. Contudo, os critérios de aceitação podem detalhar algumas das questões descritas na *user story*. Além disso, não devem existir casos de teste duplicados, ou seja, dois exemplos que descrevam as mesmas características da *user story*.

Quando descritos num formato suportado pela *framework* de automação de testes, os programadores podem automatizar os casos de teste, escrevendo o código respetivo à medida que implementam a funcionalidade descrita pela *user story*. Por conseguinte, os testes de aceitação tornam-se em requisitos executáveis.

5. Gestão das Atividades de Testes – 335 minutos

Palavras-Chave

abordagem de teste, análise de riscos, avaliação de riscos, controlo de riscos, controlo de testes, critérios de entrada, critérios de saída, gestão de defeitos, gestão de riscos, identificação de riscos, mitigação do risco, monitorização de riscos, monitorização do teste, nível de riscos, pirâmide de testes, planeamento de testes, plano de testes, quadrantes de testes, relatório de conclusão do teste, relatório de defeitos, relatório de progresso de testes, risco, risco de produto, risco de projeto, testes baseados na avaliação do risco

Objetivos de Aprendizagem para o Capítulo 5:

5.1 Planeamento de Testes

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

FL-5.1.2 (K1) Reconhecer como um testador acrescenta valor ao planeamento da iteração e planeamento da entrega

FL-5.1.3 (K2) Comparar e diferenciar entre critérios de entrada e critérios de saída

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

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

FL-5.1.6 (K1) Recordar os conceitos da pirâmide de testes

FL-5.1.7 (K2) Resumir os quadrantes de testes e as respetivas relações com os níveis de teste e os tipos de teste

5.2 Gestão de Risco

FL-5.2.1 (K1) Identificar o nível de risco utilizando a probabilidade do 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 de produto pode influenciar a profundidade e o âmbito dos testes

FL-5.2.4 (K2) Explicar as medidas que podem ser tomadas em resposta aos riscos de produto analisados

5.3 Monitorização do Teste, Controlo de Teste e Conclusão do Teste

FL-5.3.1 (K1) Recordar métricas utilizadas para testes

FL-5.3.2 (K2) Resumir as finalidades, conteúdos e público-alvo para relatórios de testes

FL-5.3.3 (K2) Exemplificar como comunicar o estado dos testes

5.4 Gestão de Configurações

FL-5.4.1 (K2) Resumir como a gestão de configurações suporta os testes

5.5 Gestão de Defeitos

FL-5.5.1 (K3) Elaborar um relatório de defeitos

5.1. Planeamento de Testes

5.1.1. Objetivo e Conteúdo de um Plano de Testes

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

- Documenta os meios e o calendário para concretizar os objetivos de teste
- Ajuda a assegurar que as atividades de teste efetuadas irão cumprir os critérios estabelecidos
- Serve como forma de comunicação entre os membros da equipa e os outros *stakeholders*
- Demonstra que os testes irão respeitar a política de teste e a estratégia de teste existente (ou explica o motivo de divergência dos testes)

O planeamento de testes orienta o pensamento dos testadores e obriga-os a enfrentar os desafios futuros relacionados com os riscos, os calendários, as pessoas, as ferramentas, os esforços, etc. O processo de elaborar um plano de testes é uma forma útil de refletir sobre os esforços necessários para concretizar os objetivos do projeto de teste.

O conteúdo típico de um plano de testes inclui:

- Contexto do teste (p. ex., âmbito, objetivos de teste, restrições, base para testes)
- Pressupostos e restrições do projeto de teste
- *Stakeholders* (p. ex., funções, responsabilidades, importância para o teste, necessidades de contratação e formação)
- Comunicação (p. ex., formas e frequência da comunicação, modelos de documentação)
- Registo de riscos (p. ex., riscos de produto, riscos de projeto)
- Abordagem de teste (p. ex., níveis de teste, tipos de teste, técnicas de teste, resultados do teste, critérios de entrada e critérios de saída, independência do teste, métricas a recolher, requisitos de dados de teste, requisitos do ambiente de teste, desvios da política de teste e estratégia de teste da organização)
- Orçamento e calendário

Para obter mais informações sobre o plano de testes e os respetivos conteúdos, consulte a norma ISO/IEC/IEEE 29119-3.

5.1.2. Contribuição do Testador para o Planeamento da Iteração e Planeamento da Entrega

Nos SDLC iterativos, ocorrem normalmente dois tipos de planeamento: o planeamento da entrega e o planeamento da iteração.

O planeamento da entrega concentra-se no futuro, na entrega de um produto, define e redefine o *backlog* do produto e pode envolver o refinamento de *user stories* maiores num conjunto de *user stories* mais pequenas. Também serve de base para a abordagem de teste e para o plano de testes ao longo de todas as iterações. Os testadores envolvidos no planeamento da entrega participam no processo de escrita das

user stories passíveis de teste e dos critérios de aceitação (ver Secção 4.5), participam também na análise de risco de qualidade e do projeto (ver Secção 5.2), calculam uma estimativa do esforço de teste associado às *user stories* (ver Secção 5.1.4), determinam a abordagem de teste e planeiam os testes para a entrega (*Release*).

O planeamento da iteração concentra-se no resultado de uma única iteração e está focado no *backlog* dessa iteração. Os testadores envolvidos no planeamento da iteração participam na análise de risco detalhada de *user stories*, determinam a testabilidade das *user stories*, dividem as *user stories* em tarefas (especialmente, tarefas de teste), calculam uma estimativa do esforço de teste para todas as tarefas de teste, e identificam e melhoram os aspetos 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 cumpridos, é provável que a atividade se revele mais difícil, demorada, dispendiosa e arriscada. Os critérios de saída definem o que deve ser concretizado para declarar uma atividade como concluída. Os critérios de entrada e de saída devem ser definidos para cada nível de teste e diferem com base nos objetivos de teste.

Os critérios de entrada típicos incluem: disponibilidade de recursos (p, ex., pessoas, ferramentas, ambientes, dados de teste, orçamento, tempo), disponibilidade de *testware* (p. ex., base para testes, requisitos passíveis de teste, *user stories*, casos de teste) e nível de qualidade inicial de um objeto de teste (p. ex., todos os testes rápidos estarem aprovados).

Os critérios de saída típicos incluem: medidas de profundidade (p. ex., nível de cobertura alcançado, número de defeitos não resolvidos, densidade de defeitos, número de casos de teste falhados) e critérios de conclusão (p. ex., testes planeados que foram executados, testes estáticos que foram efetuados, todos os defeitos encontrados estarem registados, todos os testes de regressão estarem automatizados).

A falta de tempo ou de orçamento também podem ser vistos como critérios de saída válidos. Mesmo sem satisfazer os outros critérios de saída, pode ser aceitável terminar o teste em tais circunstâncias, se os *stakeholders* tiverem revisto e aceitado o risco de disponibilização sem mais testes.

No desenvolvimento *Agile* de software, os critérios de saída são frequentemente designados por Definição de Concluído (*Definition of Done*), uma vez que definem as métricas associadas aos objetivos da equipa para o item que se quer entregar. Os critérios de entrada que uma *user story* tem de cumprir para se poder iniciar as atividades de desenvolvimento e/ou teste são designadas por *Definition of Ready*.

5.1.4. Técnicas de Estimativa

A estimativa do esforço de teste implica a previsão da quantidade de trabalho, relacionada com o teste, com vista ao cumprimento dos objetivos de um projeto de teste. É importante deixar claro para os *stakeholders* que a estimativa é baseada num conjunto de pressupostos, pelo que está sempre sujeita a um erro de estimativa. A estimativa das tarefas pequenas é, por norma, mais exata do que para as tarefas grandes. Por conseguinte, ao calcular a estimativa de uma tarefa grande, a mesma pode ser decomposta num conjunto de tarefas mais pequenas, as quais, por sua vez, podem ser estimadas.

Neste *Syllabus*, são descritas as seguintes quatro técnicas de estimativa:

Estimativa baseada em rácios. Nesta técnica baseada em métricas, são recolhidos valores de projetos anteriores da organização, o que permite derivar os rácios "padrão" de projetos semelhantes. Os rácios dos projetos da organização (p. ex., obtidos a partir de dados históricos) são normalmente a melhor fonte

para utilizar no processo de estimativa. Por conseguinte, estes rácios padrão podem ser utilizados para calcular uma estimativa do esforço de teste para o novo projeto. Por exemplo, se no projeto anterior existir uma relação de esforço de desenvolvimento de 3:2, e no projeto atual existe uma previsão de esforço de desenvolvimento de 600 pessoas/dia, é possível estimar que o esforço de teste seja de 400 pessoas/dia.

Extrapolação. Nesta técnica baseada em métricas, as medidas são efetuadas o mais cedo possível no projeto atual para recolher os dados. Ao obter observações suficientes, é possível aproximar o esforço necessário para o trabalho restante extrapolando estes dados (geralmente, aplicando um modelo matemático). Este método é bastante adequado nos SDLC iterativos. Por exemplo, a equipa pode extrapolar o esforço de teste na iteração futura como a média de esforço das últimas três iterações.

Wideband Delphi. Nesta técnica iterativa baseada em especialistas, os especialistas efetuam estimativas com base na experiência. Cada especialista, em isolamento, calcula uma estimativa do esforço. Os resultados são recolhidos e se existirem desvios fora do intervalo de fronteiras acordado, os especialistas debatem as respetivas estimativas atuais. Em seguida, é solicitado a cada especialista para efetuar uma nova estimativa com base no *feedback*, novamente em isolamento. Este processo é repetido até ser obtido um consenso. O planeamento poker é uma variante de *Wideband Delphi*, normalmente utilizado no desenvolvimento *Agile* de software. No planeamento poker, as estimativas são geralmente efetuadas utilizando cartões com números que representam o tamanho do esforço.

Estimativa de três pontos. Nesta técnica baseada em especialistas, são efetuadas três estimativas pelos especialistas: a estimativa mais otimista (a), a estimativa mais provável (m) e a estimativa mais pessimista (b). A estimativa final (E) é a média aritmética ponderada. Na versão mais popular desta técnica, a estimativa é calculada como E = (a + 4*m + b) / 6. A vantagem desta técnica é o facto de permitir aos especialistas calcular o erro de medição: SD = (b – a) / 6. Por exemplo, se a estimativa (em pessoas/hora) for: a=6, m=9 e b=18, a estimativa final será 10±2 pessoas/hora (ou seja, entre 8 e 12 pessoas/hora), uma vez que E = (6 + 4*9 + 18) / 6 = 10 e SD = (18 – 6) / 6 = 2.

Para obter mais informações sobre estas e muitas outras técnicas de estimativa de teste, consulte (Kan 2003, Koomen 2006, Westfall 2009).

5.1.5. Priorização de Casos de Teste

Uma vez especificados os casos de teste e os procedimentos de teste e agrupados em baterias de testes, essas baterias podem ser organizadas num cronograma de execução que define a ordem em que devem ser executados os testes. Ao priorizar os casos de teste, pode-se ter em conta vários fatores. As estratégias de priorização de casos de teste mais utilizadas são as seguintes:

- Priorização baseada em risco, onde a ordem da execução de teste é baseada nos resultados da análise de risco (ver Secção 5.2.3). Os casos de teste que abrangem os riscos mais importantes são executados primeiro.
- Priorização baseada em cobertura, onde a ordem da execução de teste é baseada na cobertura (p. ex., cobertura de instruções). Os casos de teste que alcançam maior cobertura são executados primeiro. Noutra variante, denominada por priorização de cobertura adicional, o caso de teste que alcança maior cobertura é executado primeiro e cada caso de teste subsequente é aquele que alcança maior cobertura adicional.
- Priorização baseada em requisitos, onde a ordem da execução de teste é baseada nas prioridades dos requisitos rastreados de acordo com os casos de teste correspondentes. As prioridades dos requisitos são definidas pelos *stakeholders*. Os casos de teste relacionados com os requisitos mais importantes são executados primeiro.

Idealmente, os casos de teste devem ser ordenados para serem executados com base nos seus níveis de prioridade, utilizando, por exemplo, uma das estratégias de priorização indicadas acima. Contudo, esta prática pode não funcionar se os casos de teste ou se os recursos que estão a ser testados tiverem dependências. Se um caso de teste com prioridade mais alta depender de um caso de teste com prioridade mais baixa, o caso de teste de prioridade mais baixa tem de ser executado primeiro.

A ordem de execução do teste também tem de ter em conta a disponibilidade dos recursos. Por exemplo, as ferramentas de teste, as pessoas ou os ambientes de teste necessários podem estar disponíveis apenas durante um período de tempo específico.

5.1.6. Pirâmide de Testes

A pirâmide de testes é um modelo que mostra que diferentes testes podem ter diferentes granularidades. O modelo da pirâmide de testes suporta a automação de testes e a alocação do esforço de teste pela equipa pois ilustra que diferentes objetivos são suportados por diferentes níveis de automação de testes. As camadas da pirâmide representam os grupos de testes. Quanto mais elevada for a camada, menor será a granularidade do teste, o isolamento do teste e maior será o tempo de execução de teste. Os testes na camada inferior são pequenos, isolados, rápidos e verificam uma pequena parte da funcionalidade, o que significa que são necessários muitos testes para obter uma cobertura razoável. A camada superior representa testes complexos, de alto nível e ponta a ponta (do início ao fim). Estes testes de alto nível são mais lentos do que os testes das camadas inferiores e verificam uma grande parte da funcionalidade, pelo que são necessários apenas alguns testes para obter uma cobertura razoável. O número e a nomenclatura das camadas pode variar. Por exemplo, o modelo da pirâmide de teste original (Cohn 2009) define três camadas: "testes unitários", "testes de serviço" e "testes de IU". Outro modelo popular define os testes unitários (componentes), testes de integração (integração de componentes) e testes ponta a ponta (do início ao fim). Também podem ser utilizados outros níveis de teste (ver Secção 2.2.1).

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 adequados no desenvolvimento *Agile* de software. O modelo permite à gestão de testes visualizar os mesmos para assegurar que todos os tipos e níveis de teste adequados estão incluídos no SDLC e compreender que alguns tipos de teste são mais relevantes para determinados níveis de teste do que outros. Este modelo também fornece uma forma de diferenciar e descrever os tipos de testes a todos os *stakeholders*, incluindo programadores, testadores e representantes do negócio.

Neste modelo, os testes podem incidir sobre o negócio ou sobre a tecnologia. Os testes também podem suportar a equipa (ou seja, orientar o desenvolvimento) ou criticar o produto (ou seja, medir o respetivo comportamento face às expetativas). A combinação destes dois pontos de vista determina os quatro quadrantes:

- Quadrante Q1 (incide sobre a tecnologia, suporta a equipa). Este quadrante contém testes de componentes e testes de integração de componentes. Estes testes devem ser automatizados e incluídos no processo de integração contínua.
- Quadrante Q2 (incide sobre o negócio, suporta a equipa). Este quadrante contém testes funcionais, exemplos, testes de *user stories*, protótipos para a experiência do utilizador, testes de API e simulações. Estes testes verificam os critérios de aceitação e podem ser manuais ou automatizados.

- Quadrante Q3 (incide sobre o negócio, critica o produto). Este quadrante contém testes exploratórios, testes de usabilidade e testes de aceitação de utilizador. Estes testes são frequentemente manuais e orientados para o utilizador.
- Quadrante Q4 (incide sobre a tecnologia, critica o produto). Este quadrante contém testes rápidos e testes não funcionais (exceto testes de usabilidade). Estes testes são frequentemente automatizados.

5.2. Gestão de Risco

As organizações enfrentam vários fatores internos e externos que tornam incerto se e quando irão concretizar os seus objetivos (ISO 31000). A gestão de risco permite às organizações aumentar a probabilidade de concretizar os objetivos, melhorar a qualidade dos produtos e aumentar a confiança e segurança dos *stakeholders*.

As principais atividades da gestão de risco são:

- Análise de risco (inclui identificação do risco e avaliação do risco; ver Secção 5.2.3)
- Controlo do risco (inclui mitigação do risco e monitorização do risco; ver Secção 5.2.4)

A abordagem de teste, na qual as atividades de teste são selecionadas, priorizadas e geridas com base na análise de risco e no controlo do risco, é designada por testes baseados na avaliação do risco.

5.2.1. Definição de Risco e Atributos de Risco

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

- Probabilidade de risco – A probabilidade de ocorrência do risco (superior a zero e inferior a um)
- Impacto do risco (danos) – As consequências desta ocorrência

Estes dois fatores expressam o nível de risco, que são uma medida para o risco. Quanto mais elevado for o nível de risco, mais importante é o seu tratamento.

5.2.2. Riscos de Projeto e Riscos de Produto

Nos testes de software, existe uma preocupação geral com dois tipos de risco: riscos de projeto e riscos de produto.

Os **riscos de projeto** estão relacionados com a gestão e o controlo do projeto. Os riscos de projeto incluem:

- Problemas organizacionais (p. ex., atrasos nas entregas dos produtos de trabalho, estimativas imprecisas, corte de custos)
- Questões relacionadas com pessoas (p. ex., competências insuficientes, conflitos, problemas de comunicação, falta de recursos humanos)
- Problemas técnicos (p. ex., não cumprimento do âmbito, ferramentas de suporte insatisfatórias)
- Problemas de fornecedor (p. ex., falha na entrega por terceiros, falência da empresa de suporte)

Os riscos de projeto, quando ocorrem, podem ter um impacto no calendário do projeto, orçamento ou âmbito, o que afeta a capacidade de concretizar os objetivos do projeto.

Os **riscos de produto** estão relacionados com as características de qualidade do produto (p. ex., descritos no modelo de qualidade da norma ISO 25010). Exemplos de riscos de produto incluem: funcionalidades em falta ou incorretas, cálculos incorretos, erros de execução, arquitetura insatisfatória, algoritmos ineficientes, tempos de resposta inadequados, experiência de utilizador insatisfatória, vulnerabilidades de segurança. Os riscos de produto, quando ocorrem, podem resultar em várias consequências negativas, incluindo:

- Insatisfação do utilizador
- Perda de receitas, confiança, reputação
- Danos a terceiros
- Custos de manutenção elevados, sobrecarga do serviço de apoio (*Helpdesk*)
- Sanções penais
- Em casos extremos, danos físicos, lesões ou mesmo morte

5.2.3. Análise de Risco de Produto

Numa perspetiva do teste, o objetivo da análise de risco do produto é proporcionar uma consciencialização para o risco de produto no sentido de concentrar o esforço de teste de forma a minimizar o nível do risco de produto. Idealmente, a análise de risco de produto começa no início do SDLC.

A análise de risco de produto consiste na identificação e avaliação de riscos. A identificação do risco consiste em gerar uma lista exaustiva de riscos. Os *stakeholders* podem identificar os riscos utilizando várias técnicas e ferramentas, p. ex., brainstorming, workshops, entrevistas ou diagramas de causa-efeito. A avaliação do risco implica: categorizar os riscos identificados, determinar a probabilidade do risco, o impacto do risco e o nível de risco, priorizar e propor formas de tratamento. A categorização ajuda na atribuição das ações de mitigação, uma vez que os riscos que se enquadram na mesma categoria podem ser mitigados utilizando uma abordagem semelhante.

A avaliação do risco pode utilizar uma abordagem quantitativa ou qualitativa, ou uma combinação das duas. Na abordagem quantitativa, o nível de risco é calculado como a multiplicação da probabilidade de risco com o impacto de risco. Na abordagem qualitativa, o nível de risco pode ser determinado utilizando uma matriz de risco.

A análise de risco de produto pode influenciar a profundidade e o âmbito do teste. Os resultados são utilizados para:

- Determinar o âmbito dos testes a efetuar
- Determinar os níveis de teste específicos e tipos de teste propostos a efetuar
- Determinar as técnicas de teste a implementar e a cobertura a alcançar
- Calcular uma estimativa do esforço de teste necessário para cada tarefa
- Priorizar os testes na tentativa de encontrar defeitos críticos o mais cedo possível

- Determinar se existem quaisquer atividades adicionais ao teste que possam ser empregues para reduzir o risco

5.2.4. Controlo do Risco de Produto

O controlo do risco de produto abrange todas as medidas que são tomadas em resposta aos riscos de produto identificados e avaliados. O controlo do risco de produto consiste na mitigação e monitorização do risco. A mitigação do risco implica implementar as ações propostas na avaliação do risco para reduzir o nível de risco. O objetivo da monitorização do risco é assegurar que as ações de mitigação são eficazes, obter mais informações para melhorar a avaliação do risco e identificar os riscos emergentes.

No que se refere ao controlo do risco de produto, após a análise de risco, é possível aplicar várias opções de resposta, p. ex., mitigação do risco através de testes, aceitação do risco, transferência do risco ou plano de contingência (Veenendaal 2012). As seguintes ações podem ser efetuadas para mitigar os riscos de produto através de testes:

- Selecionar os testadores com o nível correto de experiência e competências, adequado a um determinado tipo de risco
- Aplicar um nível adequado de independência do teste
- Efetuar revisões e análises estáticas
- Aplicar as técnicas de teste e os níveis de cobertura adequados
- Aplicar os tipos de teste adequados que abordem as características de qualidade afetadas
- Efetuar testes dinâmicos, incluindo testes de regressão

5.3. Monitorização do Teste, Controlo de Teste e Conclusão do Teste

A monitorização do teste tem como objetivo recolher informações sobre os testes. Estas informações são utilizadas para avaliar o progresso dos testes e para determinar se os critérios de saída dos testes, ou as tarefas de teste associadas aos critérios de saída foram cumpridas, como, por exemplo, o cumprimento dos objetivos para cobertura de riscos de produto, de requisitos ou dos critérios de aceitação do produto.

O controlo de testes utiliza as informações da monitorização do teste para fornecer, sob a forma de diretivas de controlo, orientação e as ações corretivas necessárias para obter testes mais eficazes e eficientes. Exemplos das diretivas de controlo incluem:

- Repriorizar os testes quando um risco identificado se torna num problema
- Reavaliar se um item de teste cumpre os critérios de entrada ou de saída devido a retrabalho
- Ajustar o cronograma de testes para resolver um atraso na entrega do ambiente de teste
- Adicionar novos recursos quando e onde necessário

A conclusão do teste recolhe os dados das atividades de teste já encerradas a fim de consolidar experiências, *testware* e quaisquer outras informações relevantes. As atividades de conclusão do teste ocorrem em marcos do projeto, tais como a conclusão de um nível de teste, a conclusão de uma iteração em *Agile*, a conclusão (ou cancelamento) de um projeto de teste, o lançamento do sistema de software ou a conclusão de uma versão de manutenção.

5.3.1. Métricas Utilizadas em Testes

As métricas de teste são recolhidos para mostrar o progresso efetuado em relação ao calendário e orçamento planeado, a qualidade atual do objeto de teste e a eficácia das atividades de teste no que diz respeito aos objetivos ou objetivo da iteração. A monitorização do teste recolhe várias métricas para suportar o controlo de testes e a conclusão do teste.

As métricas de teste comuns incluem:

- Métricas de progresso do projeto (p. ex., conclusão de tarefas, utilização de recursos, esforço de teste)
- Métricas de progresso do teste (p. ex., progresso de implementação dos casos de teste, progresso na preparação do ambiente de teste, número de casos de teste executados/não executados, aprovados/falhados, tempo de execução de teste)
- Métricas da qualidade do produto (p. ex., disponibilidade, tempo de resposta, tempo médio para falhar)
- Métricas de defeito (p. ex., número e prioridades de defeitos encontrados/corrigidos, densidade de defeitos, percentagem de deteção 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 testes, custo da qualidade na organização)

5.3.2. Finalidade, Conteúdo e Público para Relatórios de Teste

O relatório de testes resume e comunica as informações do teste durante e após a realização dos testes. Os relatórios de progresso de testes suportam o controlo contínuo do teste e têm de fornecer informações suficientes para poder efetuar modificações no cronograma de testes, nos recursos, ou no plano de testes, quando estas alterações forem necessárias devido a desvios do plano ou alteração das circunstâncias. Os relatórios de conclusão do teste resumem uma fase específica dos testes (p. ex., nível de teste, ciclo de teste, iteração) e podem fornecer informações para os testes subsequentes.

Durante a monitorização e o controlo de testes, o gestor de testes cria relatórios de progresso de testes para os *stakeholders* para os manter informados. Os relatórios de progresso de testes são gerados numa base regular (p. ex., diariamente, semanalmente, etc.) e incluem:

- Período de teste
- Progresso do teste (p. ex., adiantado ou atrasado em relação ao calendário), incluindo quaisquer desvios significativos
- Impedimentos nos testes e soluções alternativas encontradas
- Métricas de teste (ver Secção 5.3.1 para exemplos)
- Riscos novos e/ou alterações de riscos no período de teste
- Testes planeados para o próximo período

O relatório de conclusão do teste é elaborado durante a conclusão do teste, quando um projeto, nível de teste, ou tipo de teste está concluído e quando, idealmente, os critérios de saída foram cumpridos. Este relatório utiliza dados do relatório de progresso de testes e complementa com dados adicionais. Um relatório de conclusão do teste típico inclui:

- Resumo do teste
- Avaliação do teste e da qualidade do produto com base no plano de testes original (ou seja, avaliação dos objetivos de teste e dos critérios de saída)
- Desvios do plano de testes (p. ex., diferenças entre o calendário, a duração e o esforço planeado).
- Impedimentos nos testes e soluções alternativas encontradas
- Métricas de teste com base nos relatórios de progresso de testes
- Riscos não mitigados, defeitos não corrigidos
- Lições aprendidas relevantes para a atividade de teste

Públicos diferentes requerem informações diferentes nos relatórios, sendo algo que depois influencia o grau de formalidade e a frequência dos relatórios. A comunicação do progresso dos testes para outros membros da mesma equipa é frequente e informal, enquanto a comunicação dos testes de um projeto concluído segue um modelo definido e ocorre apenas uma vez.

A norma ISO/IEC/IEEE 29119-3 inclui modelos e exemplos dos relatórios de progresso de testes (designados por relatório de estado do teste) e relatórios de conclusão do teste.

5.3.3. Comunicar o Estado do Teste

A melhor forma de comunicar o estado do teste varia, podendo focar-se nas preocupações da gestão de testes, depender das estratégias de teste organizacionais, das normas regulamentares, ou, no caso de equipas auto-geridas (ver Secção 1.5.2), da própria equipa. As opções incluem:

- Comunicação verbal entre os membros da equipa e os outros *stakeholders*
- *Dashboard* (p. ex., painéis de instrumentos (*dashboards*) da CI/CD, quadros de tarefas e gráficos de *burndown*)
- Canais de comunicação eletrónicos (p. ex., e-mail, conversação (*chat*))
- Documentação online
- Relatórios de teste formais (ver Secção 5.3.2)

É possível utilizar uma ou mais destas opções. Uma comunicação mais formal pode ser mais adequada para as equipas distribuídas, onde a comunicação direta e presencial nem sempre é possível devido à distância geográfica ou às diferenças de fuso horário. Normalmente, os diferentes *stakeholders* estão interessados em vários tipos de informação, pelo que a comunicação deve ser adaptada em conformidade.

5.4. Gestão de Configurações

Nos testes, a gestão de configurações (CM – *Configuration Management*) fornece uma disciplina para identificar, controlar e rastrear os produtos de trabalho, tais como planos de testes, estratégias de teste, condições de teste, casos de teste, *scripts* de teste, resultados do teste, registos de teste e relatórios de testes como elementos de configuração.

Num elemento de configuração complexo (p. ex., um ambiente de teste), a CM regista os itens em que consiste, bem como as respetivas relações e as versões. Se o elemento de configuração for aprovado para efetuar testes, irá tornar-se numa linha de base (*baseline*) e apenas pode ser alterado por um processo formal de controlo de alterações.

A gestão de configurações mantém um registo dos elementos de configuração alterados quando é criada uma nova linha de base (*baseline*). É possível reverter para uma linha de base (*baseline*) anterior para reproduzir os resultados de teste anteriores.

Para suportar o teste de forma adequada, a CM assegura o seguinte:

- Todos os elementos de configuração, incluindo os itens de teste (partes individuais do objeto de teste), são identificados de forma unívoca, com as versões controladas, as alterações rastreadas, e respetivos relacionamentos com os outros elementos de configuração para que a sua rastreabilidade possa ser mantida durante todo o processo de teste
- Todos os documentos e todos os itens de software identificados são referenciados de forma inequívoca na documentação de testes

A integração contínua, a entrega contínua, a implementação contínua e os testes associados são normalmente implementados como parte de um *pipeline* automatizado do *DevOps* (ver Secção 2.1.4), onde está incluída a CM automatizada.

5.5. Gestão de Defeitos

Uma vez que um dos principais objetivos de teste é detetar defeitos, é importante ter um processo de gestão de defeitos estabelecido. Apesar de ser feita aqui referência a "defeitos", as anomalias comunicadas podem revelar-se como defeitos reais ou algo mais (p. ex., falso positivo, pedido de alteração). Isto é resolvido durante o processo de tratamento dos relatórios de defeitos. As anomalias podem ser comunicadas durante qualquer fase do SDLC e a forma de o fazer depende do SDLC. O processo tem de ser seguido por todos os *stakeholders* envolvidos. No mínimo, o processo de gestão de defeitos inclui um fluxo de trabalho para tratar cada anomalia de forma individual, desde a sua descoberta até ao seu encerramento, e também definir as regras para a sua classificação. O fluxo de trabalho abrange atividades para registar, analisar e classificar as anomalias comunicadas, decidir um processo de resposta adequado, tal como corrigir ou manter o defeito atual e, finalmente, encerrar o relatório de defeitos. É recomendado lidar com os defeitos oriundos do teste estático (especialmente da análise estática) de uma forma semelhante.

Os relatórios de defeitos típicos têm os seguintes objetivos:

- Fornecer aos responsáveis pelo tratamento e pela resolução dos defeitos informações suficientes para resolver o problema
- Fornecer uma forma de rastrear a qualidade do produto de trabalho

- Fornecer ideias para a melhoria dos processos de desenvolvimento e de teste.

Um relatório de defeitos registado durante os testes dinâmicos normalmente inclui:

- Identificador único
- Título e breve resumo da anomalia
- Data de observação da anomalia, entidade responsável e autor, incluindo a respetiva função
- Identificação do objeto de teste e do ambiente de teste
- Contexto do defeito (p. ex., caso de teste em execução, atividade de teste em curso, fase do SDLC e outras informações relevantes, tais como técnica de teste, *checklist* ou dados de teste em utilização)
- Descrição da falha para permitir a reprodução e resolução, incluindo os passos que detetaram a anomalia e qualquer registo de teste relevante (*logs*), dados da base de dados, capturas de ecrã ou gravações
- Resultados esperados e resultados obtidos
- Gravidade do defeito (grau do impacto) nos interesses dos *stakeholders* ou requisitos
- Prioridade para corrigir
- Estado do defeito (p. ex., em aberto, adiado, duplicado, à espera de correção, a aguardar teste de confirmação, reaberto, fechado, rejeitado)
- Referências (p. ex., para o caso de teste)

Alguns destes dados podem ser incluídos automaticamente ao utilizar ferramentas de gestão de defeitos (p. ex., identificador, data, autor e estado inicial). Os 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 Testes – 20 minutos

Palavras-Chave

automação de testes

Objetivos de Aprendizagem para o Capítulo 6:

6.1 Ferramentas de Suporte aos Testes

FL-6.1.1 (K2) Explicar como as diferentes ferramentas de teste suportam os testes

6.2 Vantagens e Riscos da Automação de Testes

FL-6.2.1 (K1) Recordar as vantagens e os riscos da automação de testes

6.1. Ferramentas de Suporte aos Testes

As ferramentas de teste suportam e simplificam várias atividades de teste. Os exemplos incluem, mas não se limitam a:

- Ferramentas de gestão – Aumentam a eficiência do processo de teste ao simplificar a gestão do SDLC, requisitos, testes, defeitos, configuração
- Ferramentas de teste estático – Suportam o testador na execução de revisões e análises estáticas
- Ferramentas de conceção de testes e de implementação do teste – Simplificam a criação dos casos de teste, de dados de teste e dos procedimentos de teste
- Ferramentas de execução de teste e de cobertura de testes – Simplificam a automatização da execução de teste e a medição da cobertura
- Ferramentas de testes não funcionais – Permitem ao testador executar os testes não funcionais que são difíceis ou impossíveis de efetuar manualmente
- Ferramentas de *DevOps* – Suportam o *pipeline* de entrega do *DevOps*, rastreamento do fluxo de trabalho, processos de compilação automatizados, CI/CD
- Ferramentas de colaboração – Simplificam a comunicação
- Ferramentas que suportam a escalabilidade e a normalização da implementação (p. ex., máquinas virtuais, ferramentas de contentorização (*containerization*))
- Qualquer outra ferramenta que auxilia os testes (p. ex., uma folha de cálculo pode ser considerada uma ferramenta de teste no contexto do teste)

6.2. Vantagens e Riscos da Automação de Testes

Adquirir simplesmente uma ferramenta não garante o sucesso. Cada nova ferramenta exigirá esforço para alcançar vantagens reais e duradouras (p. ex., na introdução, manutenção e formação para o uso de ferramentas). Também existem alguns riscos, que necessitam de análise e mitigação.

As potenciais vantagens de utilizar a automação de testes incluem:

- Tempo poupado ao reduzir o trabalho manual repetitivo (p. ex., execução de testes de regressão, reintrodução dos mesmos dados de teste, comparação dos resultados esperados vs. resultados obtidos, e verificação face a normas de codificação)
- Prevenção de erros humanos simples através de uma maior consistência e repetibilidade (p. ex., os testes são consistentemente derivados de requisitos, os dados de teste são criados de forma coerente, e os testes são executados por uma ferramenta numa determinada ordem e com a mesma frequência)
- Avaliação mais objetiva (p. ex. cobertura) e apresentação de medidas que são demasiado complexas para serem obtidas por seres humanos
- Acesso mais fácil à informação sobre os testes para suportar a gestão de testes e o relatório de testes (p. ex., estatísticas, gráficos e dados agregados sobre o progresso de testes, taxas de defeitos e duração da execução de teste)

- Redução dos tempos de execução de testes para proporcionar uma deteção antecipada dos defeitos, um *feedback* mais rápido e um tempo de colocação no mercado mais rápido
- Mais tempo para os testadores poderem conceber testes novos, mais exaustivos e mais eficazes

Os potenciais riscos de utilizar a automação de testes incluem:

- Expetativas irrealistas sobre as vantagens de uma ferramenta (incluindo funcionalidade e facilidade de utilização)
- Estimativas imprecisas de tempo, custos, esforço necessário para introduzir uma ferramenta, manter os *scripts* de teste e alterar o processo de teste manual existente
- Utilizar uma ferramenta de teste quando o teste manual é mais apropriado
- Depender demasiado de uma ferramenta, p. ex., ignorar a necessidade do pensamento crítico realizado por humanos
- A dependência do fornecedor da ferramenta, que pode encerrar a atividade, descontinuar a ferramenta, vender a ferramenta a outro fornecedor ou fornecer um suporte insatisfatório (p. ex., respostas a questões, atualizações e correções de defeitos)
- Utilizar um software de código aberto que pode ser abandonado, o que significa que não existem atualizações disponíveis, ou os seus componentes internos poderão necessitar de atualizações bastante frequentes representando um desenvolvimento adicional
- A ferramenta de automação não é compatível com a plataforma de desenvolvimento
- Escolher uma ferramenta inadequada que não está em conformidade com os requisitos regulamentares e/ou as normas de segurança.

7. 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

Livros

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

Artigos e Páginas Web

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/old-blog/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-good-stories-and-smart-tasks/

8. Apêndice A – Objetivos de Aprendizagem/Nível Cognitivo de Conhecimento

Os seguintes objetivos de aprendizagem são definidos como aplicáveis a este *Syllabus*. Cada tópico será examinado de acordo com o objetivo de aprendizagem definido para o mesmo. Os objetivos de aprendizagem começam com um verbo de ação que corresponde ao respetivo nível cognitivo de conhecimento indicado abaixo.

Nível 1: Lembrar (K1) – O candidato deverá reconhecer, lembrar e recordar um termo ou conceito.

Verbo de ação: identificar, recordar, lembrar, reconhecer.

Exemplos:

- "Identificar objetivos típicos dos testes."
- "Recordar os conceitos da pirâmide de testes."
- "Reconhecer como um testador acrescenta valor ao planeamento da iteração e planeamento da entrega."

Nível 2: Compreender (K2) – O candidato consegue selecionar os motivos ou as explicações para instruções relacionadas com o tópico, e consegue resumir, comparar, classificar e dar exemplos para o conceito de testes.

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

Exemplos:

- "Classificar as diferentes opções para escrever critérios de aceitação."
- "Comparar as várias funções nos testes" (procurar semelhanças, diferenças ou ambos).
- "Distinguir entre riscos de projeto e riscos de produto" (permite diferenciar conceitos).
- "Exemplificar a finalidade e o conteúdo de um plano de testes."
- "Explicar o impacto do contexto no processo de teste."
- "Resumir as atividades do processo de revisão."

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

Verbos de ação: aplicar, implementar, elaborar, utilizar.

Exemplos:

- "Aplicar a priorização de casos de teste" (deve fazer referência a um procedimento, técnica, processo, algoritmo, etc.).
- "Elaborar um relatório de defeitos."
- "Utilizar a análise de valor fronteira para obter casos de teste."

Referências para os níveis cognitivos dos objetivos de aprendizagem:

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

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

9. Apêndice B – Matriz de Rastreabilidade do Valor para o Negócio com os Objetivos de Aprendizagem

Esta secção apresenta o número de Objetivos de Aprendizagem do Nível *Foundation* relacionados com o Valor para o Negócio (BO – Business Outcomes) e a rastreabilidade entre o Valor para o Negócio do Nível *Foundation* e os Objetivos de Aprendizagem do Nível *Foundation*.

Valor para o Negócio: Nível *Foundation*		FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
BO1	Compreender o que são os testes e saber por que motivo são vantajosos	6													
BO2	Compreender os conceitos fundamentais dos testes de software		22												
BO3	Identificar as atividades e a abordagem de teste a implementar em função 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 do software						6								
BO7	Compreender os princípios da gestão 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 com os testes									7					
BO10	Trabalhar no contexto de uma equipa multifuncional										8				
BO11	Saber quais os riscos e as vantagens relacionadas com a automação de testes											1			
BO12	Identificar as competências essenciais necessárias para os testes												5		
BO13	Compreender o impacto dos riscos nos testes													4	
BO14	Informar sobre o progresso e a qualidade dos testes com eficácia														4

Capítulo/ Secção/ Subsecção	Objetivo de Aprendizagem	Nível K	VALOR PARA O NEGÓCIO													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
Capítulo 1	**Fundamentos dos Testes**															
1.1	**O que são os Testes?**															
1.1.1	Identificar objetivos típicos dos testes	K1	X													
1.1.2	Diferenciar o teste do *debugging*	K2		X												
1.2	**Porque Precisamos de Testes?**															
1.2.1	Exemplificar os motivos pelos quais os testes são necessários	K2	X													
1.2.2	Recordar a relação entre teste e garantia de qualidade	K1		X												
1.2.3	Distinguir entre causa raiz, erro, defeito e falha	K2		X												
1.3	**Princípios dos Testes**															
1.3.1	Explicar os sete princípios dos testes	K2		X												
1.4	**Atividades de Teste, *Testware* e Funções de Teste**															
1.4.1	Resumir as diversas atividades e tarefas de teste	K2			X											
1.4.2	Explicar o impacto do contexto no processo de teste	K2			X			X								
1.4.3	Diferenciar o *testware* que suporta as atividades de teste	K2			X											
1.4.4	Explicar o valor de manter a rastreabilidade	K2				X	X									
1.4.5	Comparar as várias funções nos testes	K2										X				

Capítulo/ Secção/ Subsecção	Objetivo de Aprendizagem	Nível K	VALOR PARA O NEGÓCIO													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
1.5	**Competências Essenciais e Boas Práticas nos Testes**															
1.5.1	Dar exemplos das competências genéricas necessárias para os testes	K2												X		
1.5.2	Recordar as vantagens do *Whole Team Approach*	K1										X				
1.5.3	Distinguir as vantagens e desvantagens da independência do teste	K2			X											
Capítulo 2	**Teste ao Longo do Ciclo de Vida de Desenvolvimento de Software**															
2.1	**Testes no Contexto dos Ciclos de Vida de Desenvolvimento de Software**															
2.1.1	Explicar o impacto do ciclo de vida de desenvolvimento de software nos testes	K2						X								
2.1.2	Recordar boas práticas de teste que se aplicam a todos os ciclos de vida de desenvolvimento de software	K1						X								
2.1.3	Recordar os exemplos de abordagens *test-first* no desenvolvimento	K1					X									
2.1.4	Resumir como o *DevOps* pode ter um impacto nos testes	K2					X	X			X	X				
2.1.5	Explicar a abordagem *shift-left*	K2					X	X								
2.1.6	Explicar como utilizar retrospetivas como mecanismo de melhoria de processos	K2					X					X				
2.2	**Níveis de Teste e Tipos de teste**															
2.2.1	Distinguir os vários níveis de teste	K2		X	X											
2.2.2	Distinguir os vários tipos de teste	K2		X												
2.2.3	Distinguir o teste de confirmação do teste de regressão	K2		X												

Capítulo/ Secção/ Subsecção	Objetivo de Aprendizagem	Nível K	VALOR PARA O NEGÓCIO													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
2.3	**Testes de Manutenção**															
2.3.1	Resumir os testes de manutenção e os fatores que os desencadeiam	K2		X					X							
Capítulo 3	**Testes Estáticos**															
3.1	**Conceitos Básicos dos Testes Estáticos**															
3.1.1	Reconhecer tipos de produtos que podem ser examinados através das diferentes técnicas de teste estático	K1				X	X									
3.1.2	Explicar o valor dos testes estáticos	K2	X			X	X									
3.1.3	Comparar e diferenciar entre testes estáticos e testes dinâmicos	K2				X	X									
3.2	***Feedback* e Processo de Revisão**															
3.2.1	Identificar as vantagens do *feedback* antecipado e frequente aos *stakeholders*	K1	X			X						X				
3.2.2	Resumir as atividades do processo de revisão	K2			X	X										
3.2.3	Recordar as responsabilidades que estão atribuídas às funções principais durante as revisões	K1				X								X		
3.2.4	Comparar e contrastar os diferentes tipos de revisão	K2		X												
3.2.5	Recordar os fatores que contribuem para uma revisão bem-sucedida	K1					X							X		
Capítulo 4	**Análise e Conceção de Testes**															
4.1	**Descrição Geral das Técnicas de Teste**															
4.1.1	Distinguir entre técnicas de teste caixa-preta, caixa-branca e baseadas na experiência	K2		X												

Capítulo/ Secção/ Subsecção	Objetivo de Aprendizagem	Nível K	VALOR PARA O NEGÓCIO														
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
4.2	**Técnicas de Teste Caixa-preta**															
4.2.1	Utilizar o particionamento por equivalências para obter casos de teste	K3					X									
4.2.2	Utilizar a análise de valor fronteira para obter casos de teste	K3					X									
4.2.3	Utilizar teste de tabelas de decisão para obter casos de teste	K3					X									
4.2.4	Utilizar teste de transição de estados para obter casos de teste	K3					X									
4.3	**Técnicas de Teste Caixa-branca**															
4.3.1	Explicar o teste de instruções	K2		X												
4.3.2	Explicar o teste de ramos	K2		X												
4.3.3	Explicar o valor dos testes caixa-branca	K2	X	X												
4.4	**Técnicas de Teste Baseadas na Experiência**															
4.4.1	Explicar a antecipação de erros	K2		X												
4.4.2	Explicar o teste exploratório	K2		X												
4.4.3	Explicar o teste baseado em *checklists*	K2		X												
4.5	**Abordagens de Teste Baseadas na Colaboração**															
4.5.1	Explicar como escrever *user stories* em colaboração com programadores e representantes do negócio	K2				X						X				
4.5.2	Classificar as diferentes opções para escrever critérios de aceitação	K2										X				

Capítulo/ Secção/ Subsecção	Objetivo de Aprendizagem	Nível K	VALOR PARA O NEGÓCIO													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
4.5.3	Utilizar o desenvolvimento orientado para testes de aceitação (ATDD) para obter casos de teste	K3					X									
Capítulo 5	**Gestão das Atividades de Testes**															
5.1	**Planeamento de Testes**															
5.1.1	Exemplificar a finalidade e o conteúdo de um plano de testes	K2		X					X							
5.1.2	Reconhecer como um testador acrescenta valor ao planeamento da iteração e planeamento da entrega	K1	X									X		X		
5.1.3	Comparar e diferenciar entre critérios de entrada e critérios de saída	K2				X		X								X
5.1.4	Utilizar técnicas de estimativa para calcular o esforço de teste necessário	K3							X		X					
5.1.5	Aplicar a priorização de casos de teste	K3							X		X					
5.1.6	Recordar os conceitos da pirâmide de testes	K1		X												
5.1.7	Resumir os quadrantes de testes e as respetivas relações com os níveis de teste e os tipos de teste	K2		X							X					
5.2	**Gestão de Risco**															
5.2.1	Identificar o nível de risco utilizando a probabilidade do risco e o impacto do risco	K1							X						X	
5.2.2	Distinguir entre riscos de projeto e riscos de produto	K2		X											X	
5.2.3	Explicar como a análise de risco de produto pode influenciar a profundidade e o âmbito dos testes	K2					X				X				X	
5.2.4	Explicar as medidas que podem ser tomadas em resposta aos riscos de produto analisados	K2		X			X								X	

Capítulo/ Secção/ Subsecção	Objetivo de Aprendizagem	Nível K	VALOR PARA O NEGÓCIO													
			FL-BO1	FL-BO2	FL-BO3	FL-BO4	FL-BO5	FL-BO6	FL-BO7	FL-BO8	FL-BO9	FL-BO10	FL-BO11	FL-BO12	FL-BO13	FL-BO14
5.3	**Monitorização do Teste, Controlo de Teste e Conclusão do Teste**															
5.3.1	Recordar métricas utilizadas para testes	K1									X					X
5.3.2	Resumir as finalidades, conteúdos e público-alvo para relatórios de testes	K2					X				X					X
5.3.3	Exemplificar como comunicar o estado dos testes	K2												X		X
5.4	**Gestão de Configurações**															
5.4.1	Resumir como a gestão de configurações suporta os testes	K2					X		X							
5.5	**Gestão de Defeitos**															
5.5.1	Elaborar um relatório de defeitos	K3		X						X						
Capítulo 6	**Ferramentas de Testes**															
6.1	**Ferramentas de Suporte aos Testes**															
6.1.1	Explicar como as diferentes ferramentas de teste suportam os testes	K2					X									
6.2	**Vantagens e Riscos da Automação de Testes**															
6.2.1	Recordar as vantagens e riscos da automação de testes	K1					X						X			

10. Apêndice C – Notas de Lançamento do *Syllabus*

O *Syllabus* de Nível Foundation v4.0 do ISTQB® é uma atualização importante baseada no *Syllabus* de Nível Foundation v3.1.1 e no *Syllabus* de Testador (*Tester*) *Agile* 2014. Por esse motivo, não existem notas de lançamento detalhadas por capítulo e secção. No entanto, é fornecido um resumo das principais alterações abaixo. Além disso, num documento separado de Notas de Lançamento, o ISTQB® fornece rastreabilidade entre os objetivos de aprendizagem (LO) na versão 3.1.1 do *Syllabus* de Nível *Foundation*, na versão de 2014 do *Syllabus* de Testador (*Tester*) *Agile*, e os objetivos de aprendizagem do *Syllabus* de Nível Foundation v4.0, que apresentam os LO que foram adicionados, atualizados ou removidos.

Na altura em que este *Syllabus* foi elaborado (2022-2023), mais de 100 países tinham feito o exame de Nível *Foundation* e mais de 800 mil pessoas são testadores certificados em todo o mundo. Com a expectativa de que todos tenham lido o *Syllabus* de Nível *Foundation* para obterem aprovação no exame, isso torna o *Syllabus* de Nível *Foundation* provavelmente no documento de teste de software mais lido de sempre! Esta atualização importante é feita com respeito pela sua herança e para melhorar os pareceres de centenas de milhares de pessoas sobre o nível de qualidade que o ISTQB® oferece à comunidade global de testes.

Nesta versão, todos os LO foram editados para torná-los atómicos e para criar uma rastreabilidade única entre os LO e as Secções do *Syllabus* e, por conseguinte, não ter conteúdo sem ter também um LO associado. O objetivo é tornar esta versão mais fácil de ler, compreender, aprender e traduzir, focando-se em aumentar a utilidade prática e o equilíbrio entre conhecimento e competências.

Este lançamento importante inclui as seguintes alterações:

- Redução do tamanho global do *Syllabus*. Este *Syllabus* não é um manual, mas um documento que serve para delinear os elementos básicos de um curso introdutório sobre testes de software, incluindo quais os tópicos que devem ser abrangidos e o respetivo nível. Por conseguinte:
    - Na maior parte dos casos, os exemplos foram excluídos do texto. A entidade formadora é responsável por fornecer os exemplos e os exercícios durante a formação
    - Foi seguida a "*Checklist* de elaboração do *Syllabus*", que recomenda o tamanho máximo de texto para os LO em cada nível K (K1 = máx. de 10 linhas, K2 = máx. de 15 linhas, K3 = máx. de 25 linhas)
- Redução do número de LO comparados com os *Syllabus* do Nível *Foundation* (NF) v3.1.1 e *Tester Agile* (TA) v2014
    - 14 LO K1 em comparação com os 21 LO no NF v3.1.1 (15) e no TA 2014 (6)
    - 42 LO K2 em comparação com os 53 LO no NF v3.1.1 (40) e no TA 2014 (13)
    - 8 LO K3 em comparação com os 15 LO no NF v3.1.1 (7) e no TA 2014 (8)
- Mais referências exaustivas a livros e artigos clássicos e/ou conceituados sobre teste de software e tópicos relacionados
- Alterações importantes no Capítulo 1 (Fundamentos dos Testes)
    - Secção sobre competências de teste expandida e melhorada
    - Secção sobre *whole team approach* (K1) adicionada

- Secção sobre independência do teste movida do Capítulo 5 para o Capítulo 1

- Alterações importantes no Capítulo 2 (Testes ao Longo dos SDLC)
  - Secções 2.1.1 e 2.1.2 reformuladas e melhoradas com os LO correspondentes modificados
  - Maior foco em práticas tais como abordagem *test-first* (K1), abordagem *shift-left* (K2) e retrospetivas (K2)
  - Nova secção sobre testes no contexto do *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
- Alterações importantes no Capítulo 3 (Testes Estáticos)
  - Secção sobre técnicas de revisão, juntamente com o LO K3 (aplicar uma técnica de revisão) removida
- Alterações importantes no Capítulo 4 (Análise e Conceção de Testes)
  - Testes de casos de uso removidos (mas ainda presentes no *Syllabus* de nível Avançado – *Test Analyst*)
  - Maior foco na abordagem de teste baseada na colaboração: novo LO K3 sobre utilizar ATDD para derivar os casos de teste e dois novos LO K2 sobre *user stories* e critérios de aceitação
  - Testes de Decisão e Cobertura de Decisões substituídos por Testes de Ramos e Cobertura de Ramos (primeiro, a cobertura de ramos é mais utilizada na prática; segundo, as várias normas definem a decisão de forma diferente, contrariamente ao "ramo"; terceiro, resolve uma falha subtil, mas grave do antigo Nível *Foundation* de 2018 que refere que "100% de cobertura de decisões implica 100% cobertura de instruções" – esta afirmação não é verdadeira no caso dos programas sem decisões)
  - Secção sobre o valor dos testes caixa-branca melhorada
- Alterações importantes no Capítulo 5 (Gestão das Atividades de Testes)
  - Secção sobre estratégias/abordagens de teste removida
  - Novo LO K3 sobre técnicas de estimativa para calcular o esforço de teste
  - Maior foco sobre conceitos já bem conhecidos e relacionados com *Agile* e com ferramentas na gestão de testes: planeamento da iteração e planeamento da entrega (K1), pirâmide de testes (K1) e quadrantes de testes (K2)
  - Secção sobre gestão de risco com melhor estrutura e descrição de quatro atividades principais: identificação do risco, avaliação do risco, mitigação do risco e monitorização do risco
- Alterações importantes no Capítulo 6 (Ferramentas de Teste)
  - Conteúdo sobre alguns problemas de automação de testes reduzido por ser demasiado avançado para o Nível *Foundation*. Secção sobre seleção de ferramentas, efetuar projetos piloto e introdução de ferramentas na organização removida

11. Apêndice D – Siglas e Tradução do Inglês

A tradução para a língua portuguesa deste *Syllabus* de Nível *Foundation* v4.0 acarretou algumas decisões importantes, nomeadamente a manutenção dos termos originais (em inglês) para os casos em que a comunidade portuguesa não traduz esses termos na sua comunicação diária.

Os termos originais que foram mantidos são:

- *Agile*
- *Backlog*
- *Build* partida
- *Checklist*
- *Coaching*
- Cobertura 0-*switch*
- *Dashboard*
- *DevOps*
- *DoR - Definition of Ready*
- *Epic*
- *Feedback*
- Gráfico de *Burndown*
- *Hot fix*
- *Kanban*
- *Lean IT*
- *Scrum*
- *Stakeholder*
- *Syllabus*
- *Testware*
- *Unified Process*
- *User Story*
- *Walkthrough*
- *Wideband Delphi*
- *Whole team approach*

Alguns termos foram traduzidos, mas manteve-se a descrição em inglês para maior facilidade de reconhecimento:

- Definição de Concluído (*DoD - Definition of Done*)
- Delimitação do Tempo (*Time boxing*)
- Entrega Contínua (*Continuous Delivery*)
- Implementação Contínua (*Continuous Deployment*)
- Linha de base (*baseline*)
- Proprietário do Produto (*Product Owner*)
- Reestruturar (*Refactoring*)
- Reunião de balanço (*debrief*)
- Reunião diária (*Daily Scrum*)

Em termos de siglas, foram mantidas as seguintes siglas:

- ATDD - desenvolvimento orientado para testes de aceitação / *acceptance test-driven development*
- BDD - desenvolvimento orientado para comportamento / *behavior-driven development*
- BVA - análise valor fronteira / *boundary value analysis*
- CD - entrega contínua / *continuous delivery*
- CI - integração contínua / *continuous integration*
- CM - gestão de configurações / *configuration management*
- DDD - conceção orientada para domínio / *domain-driven design*
- FDD - desenvolvimento orientado para funcionalidades / *feature-driven development*
- GUI - interface gráfica do utilizador / *graphical user interface*
- LO - objetivos de aprendizagem / *learning objectives*
- QA - garantia de qualidade / *quality assurance*
- QC - controlo de qualidade / *quality control*
- SDLC - ciclo de vida de desenvolvimento de software / *software development lifecycle*
- TC - caso de teste / *test case*
- TDD - desenvolvimento orientado para testes / *test-driven development*
- UAT - testes de aceitação de utilizador / *user acceptance testing*
- XP - *eXtreme Programming*

12. Índice Remissivo