Certified Tester Advanced Level Agile Tester (CTAL-AT) Syllabus

v2.0

International Software Testing Qualifications Board

Aviso de Direitos Autorais

Histórico de Revisões

Versão	Data	Observações
CT-AT v1.0 (2014)	31/05/2014	Primeira versão (substituída)
CTAL-AT v2.0 (2026)	17/04/2026	Reestruturação e reposicionamento

Índice

Agradecimentos

Este documento foi formalmente lançado pela Assembleia Geral do ISTQB® em 17 de abril de 2026. Foi produzido por uma equipe do International Software Testing Qualifications Board: Wim Decoutere, Michaël Pilaeten, Adam Roman, Murian Song, Nele Van Asch.

A equipe agradece ao Dr. Stuart Reid e a Mark Rutz pela revisão técnica, bem como à equipe de revisão e aos Conselhos Membros por suas sugestões e contribuições.

As seguintes pessoas participaram da revisão, comentários e votação deste syllabus: Mariam Abdellatif, Hatim Abuazab, Bíró Ádám, Tom Adams, Gergely Agnecz, Laura Albert, Tobias Anderson, Chris Van Bael, André Baumann, Jürgen Beniermann, Menno van den Berg, Ruud van Berkum, Armin Born, Earl Burba, Simeone Chiumarulo, Marco Ciarlitto, Alessandro Collino, Walter Eder, Michael Fischlein, Valeriya Gagarina, Attila Gyuri, Ferenc Hamori, Andrea Horvath, Sagar Joshi, Mattijs Kemmink, Jędrzej Kwapiński, Thomas Letzkus, Ine Lutterman, Niranjan Maharajh, Judy McKay, Imre Meszaros, Gary Mogyorodi, Thaer Mustafa, Frank Neirynck, Andreas Neumeister, Melanie Neumueller, Ingvar Nordström, Tauhida Parveen, Tal Pe'er, Christoph Peters, Lukas Piska, Horst Pohlmann, Andrew Pollner, Nishan Portoyan, Meile Posthuma, Randall Rice, Jean François Riverin, Nicola De Rosa, Mark Rutz, Sarno Salvatore, Márton Siska, Klaus Skafte, Péter Sótér, Michael Stahl, Lucjan Stapp, Ioannis Symeonidis, Reifa Tangon, Richard Taylor, Benjamin Timmermans, Giancarlo Tomasig, Stephanie Ulrich, Csilla Varga, Linda Vreeswijk e Jana Zientková.

Grupo de Trabalho do Nível Fundamental do International Software Testing Qualifications Board na época em que o Syllabus de Extensão Ágil do Nível Fundamental v1.0 (2014) foi concluído: Rex Black (Presidente), Bertrand Cornanguer (Vice-Presidente), Gerry Coleman (Líder de Objetivos de Aprendizagem), Debra Friedenberg (Líder de Exames), Alon Linetzki (Líder de Resultados de Negócios e Marketing), Tauhida Parveen (Editora) e Leo van der Aalst (Líder de Desenvolvimento).

Autores do Syllabus de Extensão Ágil do Nível Fundamental v1.0 (2014): Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh e Stephan Weber.

Revisores internos do Syllabus de Extensão Ágil Nível Fundamental v1.0 (2014): Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael e Erik van Veenendaal.

A equipe agradece também às seguintes pessoas, dos Conselhos Membros e da comunidade de especialistas em Agile, que participaram da revisão, dos comentários e da votação da Extensão Agile de Nível Básico v1.0 (2014): Dani Almog, Richard Berns, Stephen Bird, Monika Bögge, Afeng Chai, Josephine Crawford, Tibor Csöndes, Huba Demeter, Arnaud Foucal, Cyril Fumery, Kobi Halperin, Inga Hansen, Hanne Hinz, Jidong Hu, Phill Isles, Shirley Itah, Martin Klonk, Kjell Lauren, Igal Levi, Rik Marselis, Johan Meivert, Armin Metzger, Peter Morgan, Ninna Morin, Ingvar Nordstrom, Chris O'Dea, Klaus Olsen, Ismo Paukamainen, Nathalie Phung, Helmut Pichler, Salvatore Reale, Stuart Reid, Hans Rombouts, Petri Säilynoja, Soile Sainio, Lars-Erik Sandberg, Dakar Shalom, Jian Shen, Marco Sogliani, Lucjan Stapp, Yaron Tsubery, Sabine Uhde, Stephanie Ulrich, Tommi Välimäki, Jurian Van de Laar, Marnix Van den Ent, António Vieira Melo, Wenye Xu, Ester Zabar, Wenqiang Zheng, Peter Zimmerer, Stevan Zivanovic e Terry Zuo.

O BSTQB® agradece aos voluntários de seu grupo de trabalho de traduções (WG Traduções) pelo empenho e esforço na tradução deste material: George Fialkovitz, Irene Nagase, José Correia, Osmar Higashi, Paula de Oliveira, Stênio Viveiros, Thiago Cesar Andrade, Wellington Oshida Avilla.

O BSTQB® também agradece aos ISTQB® Accredited Training Providers no Brasil que foram convidados a contribuir na revisão da tradução. No momento deste trabalho os seguintes provedores estavam credenciados: Conecteseaqui (conecteseaqui.com.br), Exseed (www.exseed.com.br), Iterasys (www.iterasys.com.br).

O BSTQB® também agradece às empresas brasileiras credenciadas no ISTQB® Partner Program no Brasil que foram convidados a contribuir na revisão da tradução. No momento deste trabalho as seguintes empresas estavam credenciadas: Cast Group (www.castgroup.com.br), Deltapoint (www.deltapoint.com.br), Keeggo (www.keeggo.com), Venturus (www.venturus.org.br).

0 Introdução

0.1 Objetivo deste Syllabus

Este syllabus constitui a base para a Qualificação Internacional em Testes de Software para CTAL-AT Agile Tester. O ISTQB® fornece este syllabus da seguinte forma:

1. Aos conselhos membros, para tradução para o idioma local e para credenciamento de provedores de treinamento. Os conselhos membros podem adaptar o syllabus às suas necessidades específicas de idioma e modificar as referências para adequá-las às publicações locais.
2. Aos organismos de certificação, para elaborar questões de exame em seu idioma local, adaptadas aos objetivos de aprendizagem deste syllabus.
3. Aos provedores de treinamento, para produzir materiais de treinamento e determinar métodos de ensino adequados.
4. Aos candidatos à certificação, para se prepararem para o exame de certificação (seja como parte de um curso de treinamento ou de forma independente).
5. À comunidade internacional de engenharia de software e sistemas, para promover a profissão de testes de software e sistemas e como base para livros e artigos.

0.2 A Certificação Agile Tester para o Teste de Software

A certificação ISTQB® Certified Tester Agile Tester (CTAL-AT) destina-se a qualquer pessoa envolvida em testes de software em projetos Ágeis. Isso inclui pessoas em funções como testadores, analistas de teste, engenheiros de teste, consultores de teste, gerentes de teste, testadores de aceite de usuário e desenvolvedores de software. Esta certificação também é adequada para qualquer pessoa que busque uma melhor compreensão dos testes de software Ágil, incluindo product owners, Scrum Masters, gerentes da qualidade, gerentes de desenvolvimento de software, analistas de negócios, diretores de TI e consultores de gestão.

0.3 Trajetória Profissional para Testadores

O esquema ISTQB® oferece suporte a profissionais de testes em todas as fases de suas carreiras, proporcionando conhecimento tanto em amplitude quanto em profundidade. Indivíduos que obtiverem a certificação ISTQB® CTAL-AT Agile Tester também podem se interessar por outros syllabi[1] especializados, com foco especial em Agile Test Leadership at Scale. Acesse *www.istqb.org* para obter as informações mais recentes sobre o Esquema de Certified Tester do ISTQB.

0.4 Resultados de Negócios

Esta seção lista os Resultados de Negócios esperados de um candidato que tenha obtido a certificação CTAL-AT Agile Tester.

Um Certified Agile Tester pode:

[1] Nota do BSTQB: plural de syllabus

Código	Descrição
CTAL-AT-BO1	Colaborar em equipes multifuncionais, estando familiarizado com os princípios e práticas básicas do desenvolvimento ágil de software
CTAL-AT-BO2	Adaptar a experiência e o conhecimento em testes existentes aos valores e princípios ágeis
CTAL-AT-BO3	Apoiar a equipe Ágil no planejamento de teste
CTAL-AT-BO4	Aplicar abordagens relevantes de desenvolvimento ágil de software e técnicas de teste para garantir que os testes ofereçam cobertura adequada
CTAL-AT-BO5	Auxiliar os stakeholders da empresa na definição de histórias do usuário, cenários, requisitos e critérios de aceite compreensíveis e com testabilidade, conforme apropriado
CTAL-AT-BO6	Criar e implementar várias abordagens de teste do desenvolvimento ágil de software
CTAL-AT-BO7	Apoiar e contribuir para a automação de testes em um projeto de desenvolvimento ágil de software
CTAL-AT-BO8	Trabalhar com — e compartilhar informações com — outros membros da equipe utilizando estilos e canais de comunicação eficazes

0.5 Objetivos de aprendizagem e Nível cognitivo de conhecimento

Os objetivos de aprendizagem apoiam os resultados de negócios e são utilizados para criar os exames de certificação CTAL-AT Agile Tester.

Em geral, todo o conteúdo deste syllabus é passível de avaliação, exceto as Introduções e os Apêndices. As questões do exame avaliarão o conhecimento de palavras-chave no nível K2 (veja abaixo) ou dos objetivos de aprendizagem no respectivo nível de conhecimento.

Os objetivos de aprendizagem específicos e seus níveis de conhecimento são apresentados no início de cada capítulo e classificados da seguinte forma:

- K1: Lembrar (os Objetivos de Aprendizagem K1 não estão presentes nesse syllabus)
- K2: Compreender
- K3: Aplicar
- K4: Analisar

Mais detalhes e exemplos de objetivos de aprendizagem são fornecidos na *Seção 11*.

Para todos os termos listados como palavras-chave logo abaixo dos títulos dos capítulos, o nome e a definição corretos do glossário do ISTQB® devem ser memorizados (K1), e a definição do glossário do ISTQB® deve ser compreendida (K2), mesmo que não seja explicitamente mencionada no objetivo de aprendizagem.

0.6 O Exame de Certificação CTAL-AT Agile Tester

O exame de certificação CTAL-AT Agile Tester será baseado neste syllabus. As respostas às questões do exame podem exigir o uso de material baseado em mais de uma seção deste syllabus. Todas as seções do syllabus são passíveis de avaliação, exceto a Introdução e os Apêndices. Normas e livros são incluídos como referências, mas seu conteúdo não é passível de avaliação, além do que está resumido no próprio syllabus a partir dessas normas e livros.

Consulte “Exam Structures and Rules” para CTAL - Agile Tester para obter mais detalhes.

Os critérios de entrada para realizar o exame CTAL - Agile Tester são:

- Que os candidatos tenham interesse em testes de software, especialmente em ambientes Ágeis.

- Que os candidatos tenham obtido a certificação ISTQB® Nível Fundamental.

No entanto, recomenda-se fortemente que os candidatos também:

- Tenham pelo menos um conhecimento básico no desenvolvimento ágil de software ou em testes de software, como seis meses de experiência como testador de sistemas ou de teste de aceite de usuário, ou como membro de uma equipe Ágil
- Faça um curso que tenha sido credenciado de acordo com os padrões ISTQB® (por um dos conselhos membros reconhecidos pelo ISTQB).

0.7 Credenciamento

Um Conselho Membro do ISTQB® pode credenciar provedores de treinamento cujo material didático siga este syllabus. Os provedores de treinamento devem obter as diretrizes de credenciamento junto ao Conselho Membro ou órgão responsável pelo credenciamento. Um curso credenciado é reconhecido como estando em conformidade com este syllabus e está autorizado a incluir um exame do ISTQB® como parte do curso.

As diretrizes de credenciamento para este syllabus seguem as Diretrizes Gerais de Credenciamento publicadas pelo Grupo de Trabalho de Gestão de Processos e Conformidade.

0.8 Tratamento de normas

Organizações internacionais de padronização, como IEEE e ISO, emitiram normas relacionadas a características de qualidade e testes de software. Tais normas são referenciadas neste syllabus. O objetivo dessas referências é fornecer um framework (como nas referências à ISO/IEC 25010 relativas a características de qualidade) ou fornecer uma fonte de informações adicionais, caso o leitor deseje. Observe que os syllabi do ISTQB® utilizam os documentos de normas como referências. Os documentos de normas não se destinam a ser objeto de exame. Consulte a *Seção 7* para obter mais informações sobre Normas.

0.9 Nível de detalhe

O nível de detalhamento deste syllabus permite a realização de cursos e exames com padrões internacionais consistentes. Para atingir esse objetivo, o syllabus é composto por:

- Objetivos instrucionais gerais que descrevem a intenção do syllabus do CTAL - Agile Tester
- Uma lista de termos que os alunos devem ser capazes de recordar
- Objetivos de aprendizagem para cada área de conhecimento, descrevendo os resultados cognitivos a serem alcançados
- Uma descrição dos conceitos-chave, incluindo referências a fontes como literatura ou normas reconhecidas

O conteúdo do syllabus não descreve toda a área de conhecimento de testes de software; ele reflete o nível de detalhe a ser abordado nos cursos de treinamento CTAL-AT Agile Tester. Ele se concentra em abordagens de teste e técnicas de teste que podem ser aplicadas a todos os projetos de software que seguem o desenvolvimento ágil de software.

O syllabus utiliza a terminologia (ou seja, os nomes e significados) dos termos de testes de software e garantia da qualidade de acordo com o *Glossário ISTQB®*.

Para a terminologia em disciplinas relacionadas, consulte os respectivos glossários: IREB-CPRE (*IREBCPRE para Engenharia de Requisitos*, [s.d.]), IEEE Pascal (*IEEE/Pascal para engenharia de software*, [s.d.]) e Agile Alliance para terminologia ágil (*Agile Alliance*, [s.d.]).

0.10 Como este syllabus está organizado

Existem seis capítulos com conteúdo passível de avaliação. O título principal de cada capítulo especifica a duração do capítulo. Para cursos de treinamento com credenciamento, o syllabus exige um mínimo de 13h de instrução distribuídas por pelo menos dois dias. O tempo mínimo de treinamento é distribuído pelos seis capítulos da seguinte forma:

- Capítulo 1: 60 minutos, Desafios da estratégia de teste e da abordagem de teste
- Capítulo 2: 60 minutos, Pessoas e equipes
- Capítulo 3: 210 minutos, Gerenciamento de teste e melhoria de processo de teste
- Capítulo 4: 135 minutos, shift left
- Capítulo 5: 285 minutos, Abordagens de teste e técnicas de teste ágeis
- Capítulo 6: 30 minutos, automação de testes e ferramentas de teste

1 Desafios da Estratégia de Teste e da Abordagem de Teste – 60 minutos

Palavras-chave

desenvolvimento orientado por teste de aceite, bug bash, caso de borda, teste de ponta-a-ponta, testes formais, testes holísticos, teste de regressão, tipo de teste, desenvolvimento orientado por teste

Palavras-chave específicas do domínio

alternância de recursos, iteração de endurecimento, iteração, jornada do usuário

Objetivos de aprendizagem:

1.1 Tipos de teste

CTAL-AT-1.1.1 (K2) Comparar os tipos de teste a serem realizados durante e após uma iteração

1.2 Teste de ponta-a-ponta

CTAL-AT-1.2.1 (K2) Explicar quando o teste de ponta-a-ponta deve ser realizado

1.3 Testes formais e testes holísticos

CTAL-AT-1.3.1 (K2) Comparar as vantagens e desvantagens dos testes formais e dos testes holísticos

1.4 Abordagens de testes de regressão

CTAL-AT-1.4.1 (K2) Diferenciar as abordagens de teste de regressão

Introdução ao Ágil

Embora os fundamentos básicos do Agile já tenham sido descritos no syllabus do Nível Fundamental, esta introdução abrangente aborda conceitos essenciais não abordados, mas críticos para a compreensão das práticas, funções e mentalidade tratadas ao longo deste syllabus.

O desenvolvimento ágil de software tornou-se uma das abordagens amplamente adotadas no desenvolvimento de software atualmente. Seus objetivos centrais são adaptar-se rapidamente às mudanças e entregar valor de forma consistente aos clientes. Para atingir essas metas, o desenvolvimento ágil de software adota uma abordagem distinta dos modelos de desenvolvimento sequencial, levando ao surgimento dos testes ágeis. Os testes ágeis vão além do simples teste de software; trata-se de um método colaborativo e integrado que incorpora a qualidade em todo o processo de desenvolvimento e destaca a responsabilidade de toda a equipe pela qualidade.

Desenvolvimento ágil de software e o Manifesto Ágil. Em 2001, um grupo de especialistas que representava as metodologias de desenvolvimento de software mais utilizadas chegou a um consenso sobre um conjunto comum de valores e princípios, que ficou conhecido como o Manifesto para o Desenvolvimento Ágil de Software ou o Manifesto Ágil (Beck; Beedle, et al., 2001). O Manifesto Ágil define os seguintes valores:

- Indivíduos e interações mais que processos e ferramentas. O desenvolvimento ágil de software é altamente centrado nas pessoas. Equipes de pessoas criam software, e é por meio da comunicação e interação contínuas, em vez da dependência de ferramentas ou processos, que as equipes podem trabalhar de forma mais efetiva.
- Software em funcionamento mais que documentação abrangente. Do ponto de vista do cliente, um software funcional é muito mais útil e valioso do que uma documentação excessivamente detalhada, e oferece a oportunidade de fornecer feedback rápido à equipe de desenvolvimento. Além disso, como o software funcional, mesmo com funcionalidade reduzida, é entregue muito mais cedo no ciclo de vida de desenvolvimento de software, o desenvolvimento ágil de software pode oferecer uma vantagem significativa no tempo de lançamento no mercado. O desenvolvimento ágil de software é especialmente útil em ambientes de negócios em rápida mudança, onde os problemas ou soluções não são claros ou onde a empresa deseja inovar em novos domínios de problemas.
- Colaboração com o cliente mais que negociação de contratos. Os clientes muitas vezes têm dificuldade em especificar os requisitos do sistema que necessitam. A colaboração direta com o cliente aumenta a probabilidade de compreender exatamente o que ele precisa. Embora ter contratos com os clientes possa ser importante, trabalhar em colaboração regular e estreita com eles provavelmente trará maior sucesso ao projeto.
- Responder a mudanças mais que seguir um plano. A mudança é inevitável em projetos de software. O ambiente em que a empresa opera, incluindo legislação, atividades da concorrência, avanços tecnológicos e outros fatores, pode impactar significativamente o projeto e seus objetivos. Esses fatores devem ser considerados no processo de desenvolvimento. Assim, ter flexibilidade nas práticas de trabalho para abraçar a mudança é mais importante do que aderir rigidamente a um plano.

Princípios ágeis. Os doze princípios do Manifesto Ágil capturam seus valores centrais, enfatizando a entrega de software valioso de forma antecipada e frequente, abraçando requisitos em constante mudança, lançamentos frequentes e colaboração diária entre as stakeholders da empresa e os desenvolvedores. Eles priorizam indivíduos motivados e apoiados, comunicação face a face, software funcional como a principal medição de progresso e desenvolvimento sustentável em um ritmo constante. Eles enfatizam a excelência técnica, a simplicidade, equipes auto-organizadas e reflexão regular para melhorar continuamente a efetividade e a adaptabilidade.

Práticas de desenvolvimento ágil de software. Equipes ágeis utilizam várias práticas prescritivas para colocar esses valores e princípios em ação. Práticas comuns na maioria das organizações ágeis incluem a criação colaborativa de histórias do usuário, retrospectivas, integração contínua e planejamento para cada iteração, bem como para o lançamento geral.

Metodologias ágeis. Várias metodologias de desenvolvimento ágil de software estão em uso nas organizações. As mais representativas são:

- Scrum – um framework leve que organiza o trabalho em iterações de duração fixa chamadas sprints, enfatizando funções definidas, cerimônias e um backlog de produto priorizado (Sutherland et al., 2020)
- Extreme Programming (XP) – uma metodologia centrada em práticas de engenharia, como desenvolvimento orientado por teste, integração contínua e colaboração estreita, com o objetivo de melhorar a qualidade e a agilidade do software (Beck; Andres, 2004)
- Kanban – um método visual de gerenciamento de fluxo de trabalho que limita o trabalho em andamento, enfatiza a entrega contínua e utiliza quadros para acompanhar e otimizar o fluxo (D. Anderson et al., 2010). **Definindo o Teste Ágil**. Janet Gregory e Lisa Crispin, vozes proeminentes na comunidade Ágil, definem o teste ágil como “práticas de teste colaborativas que ocorrem desde a concepção até a entrega, apoiando a entrega frequente de produtos de qualidade que agregam valor comercial para nossos clientes. As atividades de teste se concentram na prevenção de defeitos, em vez da detecção de defeitos, e trabalham para fortalecer e apoiar a ideia de responsabilidade de toda a equipe pela qualidade” (Gregory; Crispin, 2019).

Essa definição articula claramente os princípios básicos do teste ágil. Ele não se limita às etapas finais de um projeto; em vez disso, o teste é uma atividade contínua desde o início. Além disso, a função de teste é compartilhada e passa de simplesmente descobrir defeitos para preveni-los proativamente. Isso incorpora a abordagem de equipe inteira, na qual todos os envolvidos no processo de desenvolvimento compartilham a responsabilidade pela qualidade.

O teste ágil abrange uma ampla gama de atividades, incluindo — mas não se limitando a — orientar o desenvolvimento com exemplos concretos, questionar ideias e suposições com perguntas perspicazes, automatizar testes para melhorar a eficiência, realizar testes exploratórios para descobrir defeitos e avaliar características de qualidade, como eficiência de performance, confiabilidade e segurança. Essas atividades ilustram que, no desenvolvimento ágil de software, as funções de teste não se limitam apenas à execução de testes; elas são moderadoras (facilitadoras) e colaboradoras da qualidade.

O Manifesto de Testes Ágeis. O Manifesto de Testes Ágeis de Karen Greaves e Samantha Laing resume os valores e princípios que orientam os testes ágeis em cinco declarações concisas. Este manifesto destaca as diferenças fundamentais em relação aos testes no desenvolvimento sequencial de software e delineia a mentalidade crucial para as funções de teste em um ambiente ágil (Greaves et al., 2019):

- **Testes ao longo do processo, em vez de testes no final**. No desenvolvimento de software sequencial, os testes geralmente começam somente após a conclusão do desenvolvimento. O Ágil, no entanto, enfatiza testes contínuos ao longo de todo o ciclo de vida de desenvolvimento de software, desde a análise de requisitos e o projeto até a implementação e a implantação. Essa abordagem ajuda a identificar e corrigir defeitos antecipadamente, reduzindo significativamente custos e esforço. Ao adotar a abordagem shift-left, toda a equipe incorpora a qualidade desde o início, tornando os testes um processo de aprendizagem contínua que fortalece a confiança no software em teste.
- **Prevenir defeitos em vez de descobrir defeitos**. O objetivo final dos testes ágeis não é apenas descobrir defeitos, mas impedir que eles ocorram desde o início. Para isso, as funções de teste participam ativamente de discussões iniciais, como o esclarecimento de requisitos e a revisão de projetos, identificando e mitigando assim, de forma proativa, possíveis problemas.
- **Compreender o teste em vez de verificar a funcionalidade**. Além de simplesmente verificar se o software executa as funções declaradas, é vital testar a compreensão subjacente da finalidade do software e das necessidades do usuário. Isso envolve avaliar o sistema da perspectiva do usuário, frequentemente por meio de histórias do usuário e cenários, para garantir que ele esteja alinhado com o valor comercial.
- **Construir o melhor sistema em vez de quebrá-lo**. A função principal de um profissional de testes não é explorar as fraquezas do sistema ou quebrá-lo intencionalmente. Em vez disso, os profissionais de testes colaboram com os desenvolvedores para ajudar a construir um sistema da

mais alta qualidade. Isso inclui fornecer feedback construtivo, propor projetos testáveis e compartilhar ideias para a melhoria contínua da qualidade.

- **Responsabilidade da equipe pela qualidade em vez de responsabilidade do testador**. A qualidade não é responsabilidade exclusiva de um indivíduo ou de uma equipe de testes dedicada. Em vez disso, é um compromisso compartilhado entre todos os envolvidos no processo de desenvolvimento, incluindo desenvolvedores, representantes de negócio, testadores e outras stakeholders. O teste ágil promove essa cultura de responsabilidade compartilhada, incentivando cada membro da equipe a participar ativamente de atividades relacionadas à qualidade.

A **Importância** e o **Impacto dos Testes Ágeis**. Esses princípios fundamentais dos testes ágeis são essenciais para garantir a qualidade do software no desenvolvimento ágil de software, que prioriza a capacidade de resposta e o feedback rápido. No desenvolvimento sequencial de software, os testes muitas vezes se tornam um gargalo do projeto ou uma “barreira de qualidade” final, logo antes do lançamento. No entanto, no Ágil, os testes são perfeitamente integrados como parte fundamental e indispensável do fluxo de trabalho de desenvolvimento. Essa integração capacita as equipes a entregar consistentemente software de alta qualidade.

Os testes ágeis também fortalecem significativamente a colaboração e a comunicação dentro da equipe. As funções de teste trabalham em conjunto com desenvolvedores e stakeholders do negócio para esclarecer requisitos, identificar possíveis defeitos antecipadamente e incentivar todos a compartilhar um objetivo comum de qualidade. Esse ambiente colaborativo não apenas contribui para a prevenção de defeitos, mas também desempenha um papel vital no aumento da produtividade da equipe e da eficiência geral. A função de teste não está necessariamente vinculada a um cargo específico dentro da organização; ela pode ser desempenhada por qualquer pessoa a qualquer momento.

Em essência, o teste ágil é mais do que apenas um conjunto de técnicas de teste; representa uma profunda mudança na cultura e na mentalidade do desenvolvimento ágil de software. É essencial focar no valor para o cliente, adaptar-se com flexibilidade às mudanças e entregar continuamente produtos de alta qualidade sob a responsabilidade coletiva de toda a equipe.

1.1 Tipos de Teste

Os tipos de teste são selecionados com base nos objetivos específicos do teste e no contexto da iteração, incluindo critérios de aceite da história do usuário, riscos de qualidade e necessidades de feedback. Todos os tipos de teste podem ser usados em qualquer estágio do projeto, mas seu uso pode diferir durante e após a iteração. Qualquer decisão tomada deve se refletir na Definição de Feito e na abordagem de teste.

Durante a iteração, os testes funcionais verificam se os recursos implementados atendem aos requisitos funcionais e critérios de aceite relacionados. Após a iteração, os testes funcionais ampliados podem incluir testes da integração de diferentes recursos, incluindo aqueles desenvolvidos por outras equipes.

Durante a iteração, os testes não funcionais geralmente se limitam a testar a eficiência de performance básica, usabilidade ou segurança em novos recursos. Após a iteração, testes não funcionais completos podem ser realizados em ambientes integrados ou de pré-produção (staging). Testes exploratórios adicionais e testes de usabilidade com um grupo mais amplo também podem ser realizados após a iteração para validar a adequação do produto à finalidade. Durante a iteração, o teste caixa-branca é usado antes ou durante a codificação, por exemplo, como parte do desenvolvimento orientado por teste (TDD). Isso auxilia na detecção precoce de defeitos e ajuda a manter uma base de código estável. Raramente é realizado como uma atividade separada pós-iteração — no entanto, a cobertura de código pode ser mensurada durante o teste caixa-preta para fornecer insights adicionais sobre a efetividade e a completude dos testes caixa-preta.

Durante a iteração, o teste caixa-preta concentra-se na verificação de recursos novos ou modificados em relação às especificações definidas tipicamente como critérios de aceite das histórias do usuário. Isso é consistente com o teste funcional utilizando o desenvolvimento orientado por teste de aceite (ATDD), ajudando a garantir que a funcionalidade entregue atenda às expectativas dos stakeholders. Após a iteração, podem ser realizados testes caixa-preta mais amplos, em nível de sistema, incluindo testes de regressão e testes de ponta-a-ponta.

Durante a iteração, os testes exploratórios desempenham um papel crítico. Essa abordagem específica para testes funcionais complementa tanto os testes caixa-branca quanto os testes caixa-preta, revelando riscos desconhecidos e comportamentos emergentes por meio de sessões estruturadas e com duração definida.

Para obter informações sobre a relação entre os tipos de teste e os quadrantes de teste, consulte *a Seção 3.1.2*.

1.2 Teste de Ponta-a-Ponta

No desenvolvimento ágil de software, o teste de ponta-a-ponta é realizado quando agrega valor ao verificar a integração entre sistemas e fluxos de trabalho do usuário. Isso inclui testar toda a aplicação e, se relevante, múltiplos sistemas ou serviços. No entanto, como medida de mitigação, deve ser ponderado em relação ao risco envolvido na cadeia de ponta-a-ponta, com a história do usuário específica, pois é caro de manter, demorado de executar e fornece feedback diagnóstico limitado. Os testes de ponta-a-ponta são normalmente realizados mais perto de um lançamento, durante iterações de endurecimento, ou quando pontos de integração em todo o sistema são introduzidos recentemente ou apresentam alto risco de falha. Embora as iterações de endurecimento sejam utilizadas em algumas organizações, elas podem indicar atenção insuficiente à qualidade durante as iterações regulares. Os testes de ponta-a-ponta são especialmente úteis em sistemas distribuídos, jornadas complexas do usuário ou cenários com dependências de terceiros que não podem ser testadas de forma confiável isoladamente. Seu objetivo é validar fluxos críticos do usuário em um ambiente de pré-produção, não detectar defeitos de baixo nível.

Vários fatores influenciam a decisão de incluir testes de ponta-a-ponta:

- Risco. O impacto do risco e a probabilidade de risco de uma falha no nível do sistema justificam testes de ponta-a-ponta direcionados, mesmo que a confiança seja construída por meio de testes de nível inferior.
- Tempo de feedback. Depender excessivamente de testes de ponta-a-ponta retarda a entrega. As equipes costumam preferir testes mais rápidos em níveis de teste inferiores e selecionam apenas alguns fluxos-chave para a cobertura de ponta-a-ponta.
- Maturidade do pipeline de Integração Contínua/Entrega Contínua (CI/CD). Quando as equipes contam com boa automação de testes, observabilidade e monitoramento em produção, a confiança pode se voltar para o feedback de produção, alternância de recursos ou testes A/B como alternativas ou complementos aos extensos testes de ponta-a-ponta.

Do ponto de vista da estratégia de teste, a pirâmide de teste continua válida: feedback amplo e rápido na base (teste de componentes), menos testes no nível de serviço e testes de ponta-a-ponta mínimos no topo. Desviar-se dessa estrutura pode aumentar custos e riscos sem agregar valor proporcional.

Em microsserviços e arquiteturas distribuídas, os testes de contrato oferecem uma alternativa aos extensos testes de ponta-a-ponta. Os testes de contrato verificam se os serviços se comunicam corretamente, validando interfaces acordadas (contratos) entre consumidores e provedores de serviços. Ao contrário dos testes de ponta-a-ponta, que exigem que todos os serviços estejam em execução, os testes de contrato podem ser executados de forma independente, proporcionando feedback mais rápido e diagnóstico de falhas mais fácil. Os testes de contrato orientados ao consumidor, nos quais os consumidores definem suas expectativas e os provedores verificam se elas são atendidas, são particularmente eficazes em ambientes Ágeis, onde os serviços evoluem de forma independente. Os testes de contrato complementam, em vez de substituir, um pequeno conjunto de testes de ponta-a-ponta que validam jornadas críticas do usuário.

Os testes de integração devem ser preferencialmente realizados no nível da interface, a menos que comportamentos no nível do sistema sejam explicitamente afetados. Os testes de ponta-a-ponta devem estar alinhados com os objetivos de qualidade definidos pelas stakeholders e não devem ser usados indiscriminadamente (*ISO 29119-6 Engenharia de software e sistemas – Testes de software*, 2021).

Testes de ponta-a-ponta roteirizados podem deixar de detectar comportamentos inesperados que surgem no uso real. Testes holísticos (ver *Seção 1.3*) e testes exploratórios (ver *Seção 5.1.5*) complementam esses testes, permitindo que os testadores examinem a qualidade sob múltiplas perspectivas e utilizem testes baseados na experiência para ajudar a descobrir comportamentos inesperados, defeitos de usabilidade e casos de borda que scripts predefinidos podem deixar passar.

1.3 Testes Formais e Testes Holísticos

O **Teste Formal** enfatiza a precisão e a formalização. Envolve a derivação de casos de teste a partir de requisitos explicitamente definidos. É tipicamente usado em contextos que exigem conformidade estrita ou controle de risco (por exemplo, teste de regressão, quando a regressão é considerada um risco).

Os benefícios dos testes formais incluem:

- **Rastreabilidade**: a rastreabilidade é garantida entre requisitos e casos de teste.
- **Alinhamento**: os casos de teste estão alinhados com as métricas de cobertura.
- **Verificação**: o teste formal auxilia na verificação, testando se o sistema foi construído corretamente.
- **Conformidade**: os casos de teste podem ser submetidos a inspeção para garantir a conformidade com normas e diretrizes.
- **Repetibilidade**: como os casos de teste são formalmente documentados, a repetibilidade é garantida.
- **Automação de testes**: a formalização e a precisão facilitam a transição para a automação de testes. Os desafios dos testes formais incluem:
- **Rigidez excessiva**: tende a restringir a exploração e a capacidade de resposta às mudanças.
- **Dependência excessiva de requisitos documentados**: a base de teste é tipicamente incompleta, inconsistente ou incorreta. Quando os casos de teste são criados exclusivamente a partir de requisitos explícitos inadequados ou falhos, isso resulta em cobertura incompleta, casos de teste desalinhados e maior probabilidade de defeitos não detectados.
- **Ignorar o inesperado**: ao focar apenas nos resultados esperados, pode-se ignorar riscos e falhas emergentes.
- **Feedback atrasado**: muitas vezes é executado tardiamente no ciclo de teste, o que adia a descoberta de problemas críticos.
- **Baixo envolvimento cognitivo**: incentiva a execução mecânica de testes em vez de uma análise criteriosa.

Os **Testes Holísticos** (Gregory, 2021) vão além das atividades de teste planejadas para abranger uma visão da qualidade em todo o sistema. Eles incorporam testes contínuos ao longo do ciclo de vida de desenvolvimento de software e incluem uma ampla gama de perspectivas: do cliente, da equipe e da organização. Eles se concentram na colaboração, na consciência do contexto, no pensamento sistêmico (ver (ISTQB-ATLAS, v1.0), seção 3.2.1) e na exploração.

Os benefícios dos testes holísticos incluem:

- Abordagem de diferentes perspectivas: combinação simultânea de pontos de vista baseados em processos, produtos e usuários.
- First Time Right (FTR): garantir que a qualidade seja incorporada, e não apenas testada após o fato.
- Testes contínuos: combina verificação (construir o sistema corretamente) e validação (construir o sistema certo).

Os desafios dos testes holísticos incluem:

- Falta de foco: sem uma estrutura disciplinada, as equipes podem realizar testes abrangentes, mas deixar de identificar riscos críticos ou regressões.
- Dificuldade de medição: a avaliação da qualidade é sensível ao contexto e carece de métricas fixas, o que desafia a confiança dos stakeholders.
- Requer mudança cultural: depende de forte colaboração e pensamento sistêmico, e sem liderança solidária e maturidade da equipe, a adoção pode falhar.

- Ambiguidade na responsabilidade: a responsabilidade de “toda a equipe” pode diluir a contabilização se não for cuidadosamente gerenciada.
- Altamente dependente do contexto: não é universalmente aplicável e pode não satisfazer requisitos regulatórios ou contratuais que exigem uma verificação mais formal.
- Problemas de automação de testes: a falta de formalidade compromete a automação adequada dos testes.

Para maximizar os benefícios dessas duas abordagens, elas podem ser usadas de forma complementar. Os testes formais devem se concentrar em componentes críticos, áreas de alto risco ou requisitos regulatórios e legais, e ser automatizados sempre que possível para minimizar esforços repetitivos. Os testes holísticos devem ser usados para controle da qualidade geral, feedback antecipado e exploração de cenários complexos, priorizando áreas com maior impacto para o usuário.

1.4 Abordagens de Teste de Regressão

No desenvolvimento ágil de software, os testes de regressão devem ser organizados para dar suporte à entrega contínua e às mudanças rápidas. Diferentes abordagens de testes de regressão podem ser usadas individualmente ou em combinação, dependendo do contexto, das capacidades da equipe, do perfil de risco e da maturidade da automação de testes:

- **Testes de regressão incrementais**: Baseiam-se em uma suíte de testes automatizados que é mantida e executada continuamente. Em vez de reexecutar uma suíte completa de testes de regressão apenas no final de uma iteração ou lançamento, um subconjunto de testes de regressão é acionado automaticamente a cada commit de código ou evento de integração, selecionado com base no impacto do risco, na probabilidade de risco, na localização e nas dependências da alteração no código. Isso permite a detecção precoce de defeitos de regressão e apoia ciclos contínuos de feedback.
- **Teste de regressão baseado em risco**: utiliza workshops frequentes de avaliação de risco ou discussões em equipe para identificar áreas onde as alterações provavelmente causarão defeitos, e orienta os esforços de regressão tanto automatizados quanto manuais. Esse método ajuda a decidir onde a automação de testes é desnecessária e onde a investigação humana agrega mais valor.
- **Testes de regressão orientados para DevOps**: Integra testes de regressão ao pipeline de CI/CD, com testes de fumaça atuando como barreiras de qualidade e testes de regressão de alta prioridade executados automaticamente após a implantação em ambientes de pré-produção ou mesmo de produção; o uso de interruptores de recursos e lançamentos canários permite a exposição seletiva, enquanto a observabilidade e o monitoramento podem, em alguns casos, substituir os tradicionais testes de regressão.
- **Teste de regressão exploratório**: é aplicado continuamente ou em intervalos regulares, complementando a automação de testes ao usar os insights dos testadores para explorar interações entre recursos e testar por defeitos inesperados.
- **Testes de regressão colaborativos**: incluem atividades de teste como bug bashes ou walkthroughs de fluxos críticos por toda a equipe, aumentando a conscientização da equipe sobre o estado atual do produto e as áreas de risco. Podem ser limitados por tempo e usados quando a automação de testes ainda não está madura, ou como um exercício de aprendizagem e descoberta de riscos.

A rastreabilidade entre histórias do usuário, requisitos e critérios de aceite e seus casos de teste correspondentes desempenha um papel fundamental nos testes de regressão. Ela permite que as equipes identifiquem rapidamente quais testes devem ser atualizados ou executados novamente quando alterações são feitas, reduzindo assim esforços desnecessários e concentrando os testes de regressão nas funcionalidades afetadas. Ela também destaca lacunas de cobertura, ajudando as equipes a equilibrar testes automatizados, testes exploratórios e testes de regressão colaborativos. A rastreabilidade simplificada (por exemplo, marcar testes em histórias do usuário em uma ferramenta de backlog) permite que as equipes mantenham a confiança em lançamentos frequentes sem atrasar a entrega.

2 Pessoas e Equipes - 60 minutos

Palavras-chave

desenvolvimento orientado por teste de aceite, desenvolvimento orientado pelo comportamento, testador de primeiro contato

Objetivos de aprendizagem:

2.1 Abordagem de equipe completa

CTAL-AT-2.1.1 (K2) Comparar generalização e especialização dentro de uma equipe

CTAL-AT-2.1.2 (K2) Dar exemplos de como motivar os representantes de negócio a realizar atividades de teste

CTAL-AT-2.1.3 (K2) Resumir como a abordagem de equipe como um todo pode auxiliar a equipe de desenvolvimento

2.2 Testadores de Tecido

CTAL-AT-2.2.1 (K2) Explicar como e quando usar testadores de tecido

Introdução

Em projetos de desenvolvimento ágil de software, pessoas e ferramentas formam a base para entregar qualidade com rapidez. A abordagem de equipe inteira enfatiza que a qualidade é uma responsabilidade compartilhada entre desenvolvedores, testadores e representantes de negócios que colaboram estreitamente. Normalmente, as equipes são compostas por membros multidisciplinares ou multifuncionais. Enquanto especialistas trazem profundo conhecimento, generalistas se adaptam a diferentes funções, permitindo maior flexibilidade e divisão da carga de trabalho. Motivar e envolver os representantes de negócio para que contribuam com os testes por meio do envolvimento precoce na definição de critérios de aceite, na revisão dos resultados dos testes e na participação em testes exploratórios melhora a qualidade do produto e o alinhamento com as metas de negócios. Essa colaboração também beneficia os desenvolvedores, pois o compartilhamento de insights sobre os testes reduz defeitos e acelera o feedback. Equipes ágeis também podem envolver testadores iniciais – que oferecem uma perspectiva “nova” para detectar defeitos precocemente. Saber como e quando usar esses testadores ajuda a fornecer feedback equilibrado e imparcial sem atrasar a entrega.

2.1 Abordagem de Equipe Inteira

2.1.1 Generalização e especialização dentro de uma equipe

A essência da abordagem de equipe completa reside no fato de testadores, desenvolvedores e representantes de negócios trabalharem juntos como uma equipe ágil em todas as etapas do processo de desenvolvimento. A generalização e a especialização de funções dentro de uma equipe ágil impactam diretamente como as atividades de teste são distribuídas, assumidas e aprimoradas. O Agile promove uma abordagem de equipe completa para a qualidade, na qual todos são responsáveis pela qualidade, independentemente do cargo ou posição.

Testadores ágeis são tipicamente especialistas generalistas em formato de T. Eles mantêm profundo conhecimento em testes e ampla expertise em várias disciplinas, como programação, análise de negócios e DevOps. Isso apoia a colaboração multifuncional e permite flexibilidade na composição da equipe.

A generalização permite que vários membros da equipe realizem uma determinada atividade. Isso sustenta um ritmo sustentável, reduz gargalos e aumenta a resiliência da equipe. Também incentiva a responsabilidade compartilhada pela qualidade,

o que é essencial em ambientes de entrega contínua. A especialização de funções ainda existe, particularmente para tarefas de teste que exigem conhecimento específico, como técnicas de teste, testes exploratórios, automação de testes ou testes de usabilidade. No entanto, as equipes ágeis se esforçam para minimizar os silos, onde os indivíduos trabalham dentro de limites estreitos de função, incentivando o compartilhamento de conhecimento e o trabalho em pares.

A programação em grupo e o mob testing (ver *Seção 5.2.1*) tornam as funções dentro da equipe menos rígidas. Essas abordagens reúnem toda a equipe, incluindo representantes de negócios, desenvolvedores e testadores, para trabalhar em uma única tarefa simultaneamente, em vez de dividir o trabalho estritamente por funções especializadas.

Os testadores frequentemente assumem a liderança ao ajudar os outros a aplicar práticas focadas na qualidade, como BDD, testes exploratórios e automação de testes. Ao mesmo tempo, eles podem aprender com os desenvolvedores sobre detalhes de implementação para avaliar melhor os riscos e projetar testes. Dessa forma, os testadores preservam sua especialização (conhecimento profundo de testes) e, ao mesmo tempo, desenvolvem habilidades mais amplas que contribuem para a generalização dentro da equipe. O equilíbrio entre especialização e generalização deve ser orientado pelo contexto. Domínios complexos ou ambientes regulamentados podem exigir funções mais especializadas, enquanto startups enxutas podem se beneficiar de generalistas com habilidades abrangentes. Independentemente da estrutura, a comunicação clara e o respeito mútuo entre os membros da equipe são essenciais. Os testadores devem contribuir para as conversas desde o início do projeto para garantir que a qualidade seja incorporada, em vez de apenas avaliada posteriormente.

Em ambientes ágeis com várias equipes (por exemplo, SAFe, LeSS), a abordagem de equipe como um todo vai além das equipes individuais. A coordenação entre equipes em relação a padrões de qualidade, testes de integração e responsabilidade por defeitos nas fronteiras entre equipes torna-se essencial. Os testadores frequentemente participam de comunidades de prática para compartilhar conhecimento entre equipes, consulte (ISTQB-ATLAS, v1.0).

2.1.2 Motivando representantes da área de negócios a participar dos testes

A qualidade é uma responsabilidade compartilhada por toda a equipe. Os representantes de negócios definem a qualidade do produto, mas também devem participar das atividades de teste. Motivá-los a testar começa pelo envolvimento dos representantes de negócios desde o início do processo de teste, especialmente na definição de critérios de aceite e na formulação de testes voltados para os negócios. O envolvimento deles nessas discussões iniciais garante que o que for entregue esteja alinhado com o valor para o cliente e a intenção dos stakeholders.

Os testes não devem ser apresentados como uma fase separada, mas sim como uma atividade colaborativa e contínua. Práticas como desenvolvimento orientado por teste de aceite, desenvolvimento orientado pelo comportamento e especificação por exemplo ajudam a consolidar o papel do representante de negócios no esclarecimento do comportamento por meio de exemplos concretos e testáveis. Essas práticas tornam os testes visíveis e relevantes para as preocupações de negócios, convidando os representantes de negócios a um diálogo significativo com os membros da equipe e outras stakeholders.

Incorporar a testabilidade às histórias do usuário durante o refinamento do backlog garante que os critérios de aceite sejam concretos, em vez de abstratos, vinculados a resultados de testes reais e verificáveis. Trabalhar em estreita colaboração com testadores e desenvolvedores nessas sessões permite que os representantes de negócios vejam como requisitos pouco claros levam à ambiguidade e ao retrabalho, e como exemplos bem formulados levam a uma entrega mais rápida e previsível.

A visibilidade é um fator-chave de motivação. Painéis de controle que mostram cobertura, tendências de defeitos e métricas relacionadas a riscos de qualidade facilitam para que os representantes de negócio compreendam a qualidade atual do produto e onde os testes agregam valor. Métricas de negócios, como satisfação do cliente, custos de suporte ou adoção de recursos, ajudam os representantes de negócio a ver os testes como parte da validação do produto, não apenas da verificação. Essas percepções ajudam a vincular os testes diretamente às decisões sobre o produto e à prontidão para lançamento.

A responsabilidade compartilhada é reforçada na colaboração diária. Durante o planejamento e o refinamento da iteração, os representantes de negócio devem ajudar ativamente a definir o que significa “concluído” para uma determinada história do usuário, incluindo cobertura e critérios de aceite. A presença deles durante atividades de teste, como bug bashes ou testes de usabilidade, garante que os ciclos de feedback permaneçam curtos e baseados no valor comercial, ao mesmo tempo em que aumenta sua motivação ao testemunharem os frutos de seu trabalho.

Por fim, reconhecer a contribuição do representante de negócios para os testes durante retrospectivas e demonstrações reforça seu papel na qualidade e os mantém engajados. Com expectativas claras, acesso a informações relevantes e oportunidades contínuas de influenciar os resultados, os representantes de negócios tendem a ver os testes como parte essencial de suas responsabilidades e uma alavanca poderosa para entregar produtos de sucesso.

2.1.3 Apoiando os desenvolvedores

A abordagem de equipe completa auxilia os desenvolvedores ao incorporar a colaboração focada na qualidade em todo o ciclo de vida de desenvolvimento de software. A interação contínua entre desenvolvedores, testadores e representantes de negócios cria um entendimento comum das metas de qualidade e dos riscos potenciais, levando a melhores decisões técnicas e de negócios. Os desenvolvedores se beneficiam do feedback precoce dos testes, o que reduz os custos de defeitos e possibilita ações corretivas rápidas, especialmente quando práticas como desenvolvimento orientado por teste de aceite e desenvolvimento orientado pelo comportamento são aplicadas.

Os testadores que trabalham em estreita colaboração com os desenvolvedores ajudam a esclarecer os critérios de aceite por meio de exemplos, levando a especificações aprimoradas por exemplo e a um melhor alinhamento entre o código e as expectativas de negócios.

Os testadores podem orientar os desenvolvedores a escrever melhores testes de componentes e testes de integração de componentes para alcançar níveis mais elevados de cobertura de código. À medida que os testes se tornam uma atividade contínua, em vez de um processo realizado em fases finais, os desenvolvedores ganham confiança para refatorar e evoluir a base de código com segurança. Automatizar os testes logo no início do ciclo de vida de desenvolvimento de software permite a detecção mais rápida de defeitos de integração e facilita a implementação de pipelines de integração contínua.

Além disso, a mentalidade de equipe como um todo quebra os silos de funções, criando um ambiente no qual os testadores contribuem para as revisões de código sob a perspectiva de qualidade e risco, e os desenvolvedores participam de sessões de testes exploratórios. Esse ambiente de aprendizagem mútua não apenas eleva a competência geral da equipe em testes, mas também promove uma responsabilidade coletiva pela qualidade do produto. Os desenvolvedores também obtêm insights sobre o comportamento real dos usuários ao participar de testes exploratórios, sessões de teste de usabilidade ou ao analisar métricas de produção, o que os ajuda a construir softwares mais alinhados com o usuário.

Outra forma pela qual a abordagem de equipe inteira apoia os desenvolvedores é envolvendo as equipes de operações, especialistas em segurança e testadores desde o início para abordar considerações de implantação, monitoramento e conformidade ao longo do desenvolvimento. Essa colaboração ajuda os desenvolvedores a projetar soluções que sejam mais fáceis de implantar, observar e proteger em produção, reduzindo retrabalhos em fases finais e problemas operacionais. Ao incorporar essas perspectivas desde o início, os desenvolvedores podem fazer escolhas arquitetônicas que equilibram funcionalidade com características de qualidade. Isso acabará por acelerar a entrega, ao mesmo tempo em que aumenta a confiança na prontidão do sistema para uso no mundo real.

2.2 Testador de Primeiro Contato

Os testadores de primeiro contato (também conhecidos tissue testers) são testadores temporários, externos à equipe, que fornecem feedback rápido e informal sobre um produto ou recurso em desenvolvimento, normalmente utilizados em setores voláteis, como a indústria de videogames. Eles são mais eficazes quando usados nas fases iniciais do design ou da implementação de recursos, antes que as equipes se comprometam com decisões caras de desenvolvimento ou infraestrutura. A ideia é expor brevemente uma nova ideia, interface ou interação a alguém que não esteja familiarizado com ela, a fim de detectar defeitos de usabilidade, comportamentos confusos ou fluxos enganosos.

Os testadores de primeiro contato são normalmente pessoas de fora da equipe ou do projeto (por exemplo, um colega de outra equipe) e devem ser utilizados com o entendimento de que seu feedback pode variar significativamente entre indivíduos e que seu envolvimento é de curto prazo. Depois de terem visto o produto uma vez, eles não são mais verdadeiramente imparciais, de modo que seu valor especial como “olhos novos” diminui (ver *Seção 4.1.4*). Essa metáfora ressalta a natureza leve e transitória da prática de teste. Um testador de primeiro contato não deve ser utilizado depois de ter visto o design em teste, pois suas reações serão influenciadas pelo conhecimento prévio. Nesse ponto, ele perde a perspectiva nova essencial.

Os testadores de primeiro contato são mais adequados para testes exploratórios, testes de usabilidade ou verificação de suposições, onde o feedback rápido é valioso, mas ainda é muito cedo para uma validação rigorosa. Por exemplo, antes de refinar uma interação ou fluxo de tela, um desenvolvedor ou testador pode orientar um testador de primeiro contato para identificar possíveis problemas. É importante não explicar demais e, assim, introduzir viés, pois isso destruirá sua visão imparcial do produto.

O testador de primeiro contato não deve substituir os testes estruturados. Em vez disso, deve precedê-los ou complementá-los. Seu principal valor está em revelar desalinhamentos precoces entre a intenção do design e a compreensão real do usuário. Por serem rápidos de executar e não exigirem o recrutamento de usuários externos, os testes de lenço de papel são úteis em contextos de Desenvolvimento Ágil de Software e DevOps.

3 Gerenciamento de Testes e Melhoria de Processo de Teste - 210 minutos

Palavras-chave

Desenvolvimento ágil de software, testes de caos, controle de teste, monitoramento de teste, planejamento de teste, melhoria de processo de teste, relatório de teste, estratégia de teste, quadrantes de teste

Palavras-chave específicas do domínio

lançamento canário, comunidade de prática, lançamento oculto, Definição de Feito, alternância de recursos, Goal-Question-Metric, métrica

Objetivos de Aprendizagem:

3.1 Planejamento de teste

CTAL-AT-3.1.1 (K2) Resumir como realizar o planejamento de teste no desenvolvimento ágil de software

CTAL-AT-3.1.2 (K4) Descrever uma estratégia de teste do projeto utilizando quadrantes de teste

3.2 Monitoramento de teste e controle de teste

CTAL-AT-3.2.1 (K2) Explicar como realizar o monitoramento de teste e o controle de teste no desenvolvimento ágil de software

3.3 Relatórios de teste

CTAL-AT-3.3.1 (K2) Comparar os diferentes tipos de cobertura que podem ser usados para relatórios de teste no desenvolvimento ágil de software

3.4 Melhoria de processo de teste

CTAL-AT-3.4.1 (K4) Selecionar medidas adequadas de melhoria de processo de teste com base em métricas no desenvolvimento ágil de software

CTAL-AT-3.4.2 (K2) Explicar como realizar a melhoria de processo de teste no desenvolvimento ágil de software

Introdução

Em projetos de desenvolvimento ágil de software, as atividades de teste são integradas em ciclos de teste curtos e iterativos, exigindo um planejamento de teste adaptativo que evolua de acordo com as prioridades do produto e da equipe. A estratégia de teste de um projeto é apoiada pelos quadrantes de teste (Gregory; Crispin, 2019), que são usados para selecionar tipos de teste, níveis de teste, abordagens de teste e técnicas de teste adequados, a fim de alcançar uma cobertura equilibrada das necessidades voltadas para os negócios e para a tecnologia.

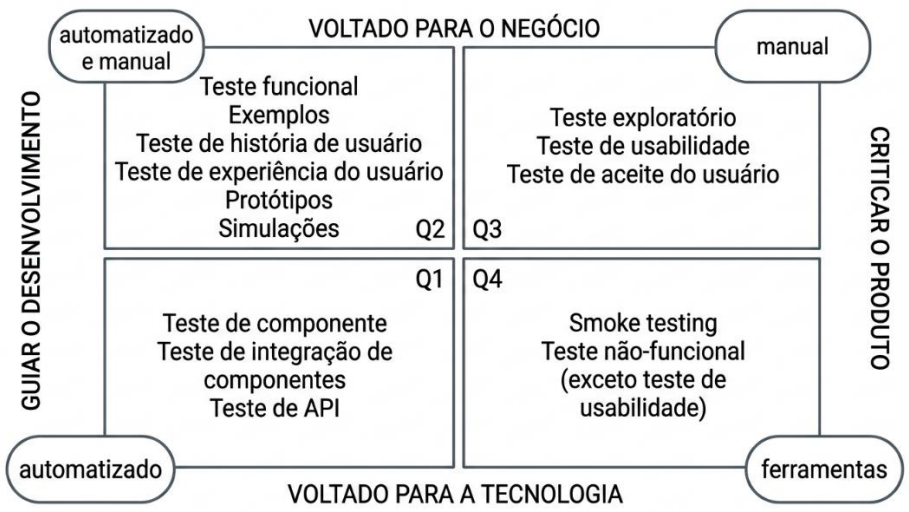

Fig. 1. Modelo dos quadrantes de teste

O quadrante 1 contém testes voltados para a tecnologia que ajudam a orientar o desenvolvimento. Eles verificam pequenos trechos de código. O quadrante 2 contém testes voltados para os negócios que ajudam a orientar o desenvolvimento. Eles validam se o produto se comporta conforme o esperado pela empresa. O quadrante 3 contém testes voltados para os negócios que avaliam criticamente o produto. Eles validam o desempenho do produto para os usuários. O quadrante 4 contém testes voltados para a tecnologia que avaliam criticamente o produto. Eles verificam as características de qualidade do produto.

Os princípios ágeis moldam ainda mais a estratégia de teste, promovendo o planejamento adaptativo, o feedback antecipado e a priorização contínua baseada em riscos para alinhar os testes ao valor de negócios. O monitoramento de teste e o controle de teste são contínuos, utilizando métricas leves e em tempo real para acompanhar o progresso, detectar riscos e ajustar prioridades. Os relatórios de teste fornecem métricas de cobertura significativas e orientadas pelo contexto, como cobertura de requisitos, código e riscos, para oferecer às stakeholders insights diretos e inequívocos sobre o status da qualidade. A melhoria de processo de teste está incorporada ao Agile, utilizando métricas para identificar gargalos, refinar práticas e aumentar a eficiência. Ao iniciar melhorias precocemente e com frequência, as equipes podem responder às mudanças, manter a qualidade e maximizar o valor dos testes em cada iteração.

3.1 Planejamento de Teste

3.1.1 Planejamento de Teste no Desenvolvimento Ágil de Software

No Agile, o planejamento de teste geralmente ocorre em dois níveis principais: planejamento de iteração e planejamento de lançamento. A estratégia de teste do projeto influencia essas atividades de planejamento.

Planejamento de Iteração

Durante o planejamento de iteração, a equipe realiza a revisão do backlog do produto e seleciona as histórias do usuário para a iteração.

Os testadores podem contribuir facilitando sessões de brainstorming de riscos com representantes de negócio e desenvolvedores para identificar riscos e priorizar atividades de teste, definir condições de teste de alto nível e refinar critérios de aceite. O planejamento de iteração inclui a seleção de tipos de teste, níveis de teste, abordagens de teste e técnicas de teste adequadas para a iteração em questão. Os testadores usam seu conhecimento e experiência para ajudar a equipe a fazer as escolhas certas. Os quadrantes de teste podem ser usados para apoiar essa tarefa.

O planejamento da iteração deve levar em consideração o ambiente de teste e as necessidades de dados de teste, a fim de garantir a prontidão da iteração. Como os testes não constituem uma fase separada no Agile e podem ocorrer a qualquer momento durante a iteração, os ambientes de teste e os dados de teste devem estar estáveis e disponíveis desde o início da iteração. O plano de testes geralmente inclui práticas de integração contínua e testes contínuos. Os testes manuais complementam a automação de testes, abordando áreas que exigem julgamento humano, como testes de usabilidade e testes exploratórios.

O planejamento da iteração inclui chegar a um acordo ou revisar a Definição de Feito no nível da equipe, detalhando os critérios de saída para histórias e outros itens do backlog dentro da iteração. Ele também esclarece como a execução de teste, os resultados dos testes e as informações de cobertura contribuem para atender à Definição de Feito para cada item e para o resultado da iteração.

Planejamento de Lançamento

Durante o planejamento de lançamento, o planejamento de teste opera em um nível mais amplo do que no planejamento de iteração. A equipe identifica objetivos de negócios de alto nível e avalia o backlog do produto para determinar o escopo e a sequência dos recursos a serem entregues. Do ponto de vista dos testes, isso envolve avaliar riscos de qualidade em todo o escopo do lançamento, definir objetivos do teste no nível do lançamento e determinar os níveis, tipos de teste, abordagens de teste e técnicas de teste apropriados necessários para mitigar esses riscos.

Os testadores contribuem identificando dependências entre recursos, restrições em ambientes de teste e dados, e pontos de integração que exigem validação específica. Elas ajudam a estimar o esforço total de teste utilizando técnicas baseadas em métricas e na experiência de especialistas, o que contribui para a capacidade da equipe de planejar a capacidade ao longo de várias iterações

A equipe delineia uma estratégia de teste no nível do lançamento que abranja, no mínimo, testes funcionais e não funcionais, como estes serão incorporados ao longo das iterações e como os testes de regressão serão gerenciados. Isso garante o planejamento antecipado para testes baseados na experiência e identifica onde stakeholders externas, como clientes ou especialistas em conformidade, precisam ser envolvidas. O testador, muitas vezes atuando como moderador, pode orientar a equipe por meio de exercícios colaborativos (por exemplo, workshop de mapeamento mental ou sessões de brainstorming de riscos) para elaborar a estratégia.

O planejamento de lançamento inclui um acordo sobre a Definição de Feito no nível do lançamento, incluindo critérios de aceite para o produto integrado. A equipe também define como os resultados dos testes, como informações de cobertura, serão analisados e comunicados para apoiar as avaliações de prontidão para o lançamento.

3.1.2 Estratégia de Teste do Projeto

Uma estratégia de teste de projeto é moldada por vários fatores contextuais, cada um influenciando como as técnicas de teste, os níveis de teste, as abordagens de teste e os tipos de teste são atribuídos aos quadrantes de teste. A complexidade do domínio do produto e a criticidade para os negócios formam a base da estratégia de teste. Em domínios com implicações legais ou de segurança (por exemplo, finanças, saúde, aviação), há maior ênfase nos métodos dos Quadrantes 2 e 4, onde abordagens rastreáveis e estruturadas são preferidas para fornecer evidências auditáveis de conformidade e cobertura (por exemplo, testes não funcionais, como testes de performance, testes de segurança, testes de compatibilidade ou testes de confiabilidade).

Estrutura organizacional e composição da equipe

Elas influenciam a profundidade e a amplitude dos testes. Uma equipe multifuncional com especialidades de teste integradas em experiência do usuário (UX), segurança ou performance permite testes mais aprofundados em todos os quadrantes. Em contrapartida, equipes que dependem de especialistas externos

ou isolados podem subutilizar testes exploratórios e testes não funcionais, a menos que sejam explicitamente planejados e apoiados. Da mesma forma, quando as equipes adotam uma forte mentalidade de responsabilidade pela qualidade por parte de toda a equipe, técnicas de teste baseadas na experiência, como os testes exploratórios do Quadrante 3, são aplicadas continuamente e integradas ao desenvolvimento, em vez de se limitarem ao final das iterações.

Frequência de entrega e estratégia de lançamento

Eles determinam quanto tempo está disponível para testes e afetam diretamente o grau de automação de testes. Ciclos de lançamento curtos e modelos de entrega contínua exigem ampla automação de testes nos Quadrantes 1 e 4. Aqui, a seleção de métodos se inclina para opções de feedback sustentáveis e de baixo custo (por exemplo, testes de componentes, Teste de API, testes de fumaça). Por outro lado, em lançamentos de ciclo mais longo ou baseados em marcos, pode haver mais oportunidades para testes manuais no Quadrante 3 (por exemplo, testes exploratórios, testes de usabilidade aprofundados e teste de aceite de usuário).

Observabilidade

Se a observabilidade for baixa, torna-se difícil confiar no monitoramento passivo implícito, aumentando a necessidade de elementos do Quadrante 2, como simulações e protótipos. Se a testabilidade for alta, as equipes podem introduzir instrumentação e interfaces exclusivas para testes a fim de permitir testes mais direcionados em todos os quadrantes.

Maturidade do produto e frequência de mudanças

Em produtos em estágio inicial ou sistemas passando por prototipagem rápida, a estratégia de teste normalmente favorece mecanismos de feedback de baixo custo e alto valor. Os testes exploratórios do Quadrante 3 têm precedência sobre modelagem formal ou cobertura exaustiva. À medida que o produto alcança a maturidade, o risco de regressão aumenta, exigindo maior investimento em automação de testes sustentável e testes estruturados (por exemplo, testes de componentes do Quadrante 1 ou testes funcionais do Quadrante 2 projetados com técnicas de teste caixa-preta) para mitigar o custo crescente das mudanças. Os Quadrantes 1 e 2 dependem cada vez mais de técnicas de teste baseadas em cobertura e automação de testes orientada por dados para manter a velocidade.

Limitações de ferramentas e maturidade da infraestrutura

Quando pipelines modernos de CI/CD, dublês de teste e ambientes em contêineres estão disponíveis, a automação de testes e a virtualização podem ser plenamente exploradas usando os Quadrantes 1 e 2, enquanto as ferramentas também permitem a aplicação de testes não funcionais do Quadrante 4. Sem esses recursos, a estratégia de teste pode se limitar a testes manuais (por exemplo, testes exploratórios ou testes de aceite de usuário do Quadrante 3) ou depender de ambientes de teste isolados. Isso introduz riscos e exige medidas compensatórias, como maior cobertura exploratória e alternância de recursos, para gerenciar a incerteza na produção.

Apetite por riscos e cultura de liderança

Uma organização avessa ao risco ou orientada por regulamentações pode impor abordagens de teste com muitos processos, priorizando cobertura documentada, condições de teste extensas e métricas quantitativas. Equipes nesses contextos tendem a aplicar técnicas de teste caixa-preta para testes funcionais do Quadrante 2 (por exemplo, análise de valor limite, teste de transição de estado, teste de tabela de decisão). Uma cultura que valoriza a aprendizagem adaptativa e a experimentação apoia testes exploratórios e testes de aceite de usuário do Quadrante 3, bem como simulações do Quadrante 2. Testes eficazes exigem o equilíbrio de todos os aspectos – os Quadrantes 1 e 2 normalmente garantem repetibilidade e confiança, enquanto os Quadrantes 3 e 4 promovem inovação e aprendizagem.

Mecanismos de feedback do cliente e observabilidade da produção

Eles podem estender a estratégia de teste além da entrega. Abordagens de teste como testes A/B, telemetria de uso do cliente e análise de padrões de falha apoiam a validação contínua do valor comercial e da tolerância a falhas operacionais. Essas práticas de teste estendem os Quadrantes 3 e 4 para ambientes de produção, possibilitando ciclos de feedback curtos.

A estratégia de teste deve ser revisada periodicamente, por exemplo, com base em resultados retrospectivos. À medida que um produto e seu contexto evoluem, a estratégia de teste deve adaptar seu

foco, deslocando-se entre os quadrantes de teste, reequilibrando continuamente a combinação de testes roteirizados e não roteirizados, bem como testes manuais e automatizados. Quanto mais dinâmico e contínuo for o modelo de entrega, mais frequente se torna essa reavaliação.

3.2 Monitoramento de Teste e Controle de Teste

No desenvolvimento ágil de software, o monitoramento de testes utiliza métricas leves exibidas por meio de gráficos de burn-down, diagramas de fluxo cumulativo, painéis de controle automatizados e visualizações semelhantes para acompanhar continuamente o progresso dos testes e a qualidade do produto. Essas visualizações são atualizadas com frequência e estão acessíveis a todos os membros da equipe para permitir uma percepção compartilhada da situação e ação imediata quando ocorrem desvios. Normalmente, as métricas se concentram no valor entregue e em fornecer feedback acionável em tempo real, em vez de apenas relatórios periódicos. A ênfase está na compreensão das tendências na descoberta de defeitos, na cobertura da automação de testes, nas descobertas dos testes exploratórios e nos dados de monitoramento de produção, em vez de acompanhar a conformidade com planos de teste predefinidos.

Em equipes ágeis, o controle de teste é exercido de forma colaborativa durante reuniões diárias, revisões de iteração e retrospectivas, onde os dados dos pipelines de CI/CD orientam as decisões. Os ajustes são feitos em tempo real, frequentemente no nível da user story ou do recurso, incluindo a priorização de sessões de testes exploratórios, o refinamento dos critérios de aceite e o foco temporário em áreas de alto risco. Essas atividades são realizadas por toda a equipe, não apenas pelos testadores, garantindo que as mudanças no escopo, no cronograma e nos objetivos de qualidade sejam discutidas e acordadas imediatamente. O feedback da automação de testes, os sinais de monitoramento de produção e os ciclos de feedback do cliente são tratados como gatilhos críticos para as decisões de controle de teste.

O monitoramento de teste e o controle de teste devem ser integrados ao fluxo de trabalho diário, utilizando colaboração multifuncional e feedback contínuo tanto dos ambientes de pré-produção quanto de produção. As decisões são frequentemente reavaliadas com base nas prioridades de negócios em evolução e no estado atual dos incrementos entregues, apoiando o princípio Ágil de responder rapidamente às mudanças, ao mesmo tempo em que se protegem as metas de qualidade.

3.3 Relatórios de Teste

Os relatórios de teste incluem várias atividades, mas esta seção se concentra em várias métricas de cobertura para refletir o progresso do teste, apoiar a tomada de decisões e aumentar a transparência. Esses tipos de cobertura devem estar alinhados com as metas da equipe, apoiar ciclos rápidos de feedback e ser leves o suficiente para serem úteis sem causar sobrecarga desnecessária. Equipes ágeis preferem indicadores de cobertura que promovam um entendimento compartilhado da qualidade e dos riscos do sistema.

A **cobertura de requisitos** é comum, especialmente no contexto de ATDD ou BDD. Ao vincular testes de aceite automatizados a histórias do usuário ou exemplos, as equipes podem demonstrar quais aspectos funcionais foram especificados, desenvolvidos e verificados. Esses critérios de aceite tornam-se unidades rastreáveis de cobertura, ajudando a equipe a comunicar a prontidão e a integridade das histórias do usuário.

A **cobertura de código** pode ser utilizada de forma seletiva, especialmente em ambientes de teste altamente automatizados. Ela deve ser interpretada com cautela e não deve ser considerada um indicador de qualidade. Embora seja útil para identificar caminhos lógicos não testados, uma alta cobertura de código, por si só, não garante que os testes sejam significativos. Equipes ágeis podem combinar isso com a semeadura de falhas ou revisões de qualidade de afirmações para obter uma compreensão mais profunda da efetividade dos testes.

A **cobertura relacionada a testes exploratórios** pode ser relatada usando gerenciamento de teste baseado em sessão ou taxas de conclusão de cartas de teste. Isso é particularmente valioso para testes exploratórios voltados para os negócios, testes de usabilidade ou testes em áreas de risco que carecem de especificações formais. Painéis de controle visuais podem ser usados para representar o que foi explorado e quais cenários ou personas foram testados.

A cobertura de infraestrutura e ambiente, particularmente em contextos de DevOps e entrega contínua, pode ser acompanhada para mostrar quais plataformas, configurações ou APIs foram exercitadas. Ferramentas de monitoramento e observabilidade também aprimoram os relatórios de cobertura operacional, revelando padrões de uso e anomalias na produção.

Em todos os casos, as métricas de cobertura no Agile devem ser discutidas de forma colaborativa, e não apenas consumidas como indicadores abstratos. Elas devem apoiar a tomada de decisões informadas e se adaptar aos contextos organizacionais, do projeto e da equipe, em vez de serem impostas como métricas prescritivas. As métricas de teste são ferramentas para comunicação e melhoria em ciclos de vida de desenvolvimento de software iterativos e adaptativos. As equipes devem resistir à ilusão de completude criada pelas métricas e, em vez disso, fornece informações personalizadas para atender às necessidades das diferentes stakeholders.

3.4 Melhoria de Processo de Teste

3.4.1 Métricas para melhorias de processo de teste

A melhoria de processo de teste baseada em métricas no desenvolvimento ágil de software requer uma interpretação cuidadosa dos indicadores de produto e de processo, alinhando-se aos princípios de melhoria contínua e aproveitando as capacidades e ferramentas da equipe. Exceto por indicadores-chave de desempenho genuínos, as métricas devem servir como mecanismos de feedback, e não como metas de desempenho, para evitar o uso indevido que poderia distorcer o comportamento e levar à otimização local.

Quando o percentual de detecção de defeitos diminui inesperadamente, uma análise da causa-raiz pode revelar testes exploratórios ineficazes ou automação de testes insuficiente. Em resposta, a equipe pode considerar refinar suas técnicas de planejamento de testes, melhorar o gerenciamento de teste baseado em sessão ou investir em ferramentas para observabilidade ou suporte a testes exploratórios.

Se os testes falham repetidamente devido à instabilidade do ambiente, investir no gerenciamento do ambiente de teste, na virtualização de serviços ou em soluções em contêineres pode melhorar a estabilidade e a confiança nos resultados dos testes.

Tempos de ciclo elevados para a resolução de defeitos podem indicar colaboração fraca entre funções ou responsabilidades isoladas. Uma possível melhoria poderia envolver a adoção de abordagens de toda a equipe para a triagem de defeitos, envolvendo testadores mais cedo nas sessões de refinamento e incorporando líderes de teste Ágeis para reforçar a responsabilidade compartilhada pela qualidade. Combinar métricas como tempo médio até a falha e tempo médio de reparo com retrospectivas permite que as equipes reconheçam gargalos nos ciclos de feedback e se adaptem de acordo.

Baixa cobertura em caminhos críticos, conforme relatado por ferramentas de cobertura de código, pode motivar esforços para fortalecer a cobertura automatizada nos níveis de teste de componentes ou de teste de integração de componentes, potencialmente reequilibrando a pirâmide de teste em direção a testes mais granulares e de execução rápida. No entanto, as equipes devem avaliar se esses testes fornecem insights significativos ou simplesmente inflacionam as métricas.

Testes redundantes ou frágeis, indicados por altas taxas de falha com baixo rendimento de defeitos, devem ser refatorados ou substituídos usando padrões de design, como objetos de página ou geradores de dados de teste. Para mais informações sobre padrões de design na automação de testes, consulte (ISTQB-TAE, v2.0).

Uma participação cada vez menor em retrospectivas ou discussões sobre qualidade pode refletir uma questão cultural. Pode ser necessário coaching para facilitar a segurança psicológica e reforçar os testes como uma responsabilidade compartilhada. Nesse contexto, métricas qualitativas, como a percepção de confiança e abertura na colaboração, podem ser tão importantes quanto os indicadores numéricos.

Quando a dívida de testes se acumula, como evidenciado por um backlog crescente de testes instáveis ou pelo tempo de testes de regressão manuais, uma estratégia de testes pode reservar tempo em cada iteração para manter os testes, alternar a responsabilidade pelos testes e aplicar práticas de refatoração de automação de testes. A medição da efetividade da automação de testes deve incluir não apenas taxas de aprovação/reprovação, mas também o esforço de manutenção e os prazos de execução de testes.

Métricas baseadas em monitoramento, como o tempo médio de recuperação ou as taxas de defeitos escapados em produção, podem dar início a melhorias no processo de teste. Um MTTR alto pode indicar alertas de sistema ausentes, restrições de recursos ou testes exploratórios insuficientes de comportamentos operacionais. Isso poderia levar à incorporação de testes de caos ou técnicas de desenvolvimento de software, como lançamentos canários, alternância de recursos ou lançamentos ocultos.

Para o desenvolvimento de pessoas, métricas como frequência de testes em pares, contagem de sessões de aprendizagem sobre defeitos ou índices de contribuição multifuncional podem indicar se o compartilhamento de conhecimento e o crescimento de capacidades estão ocorrendo. A falta de colaboração multifuncional entre funções pode exigir mentoria, rotação de funções ou caminhos de aprendizagem estruturados, como iniciativas de comunidades de prática.

Melhorias relacionadas a ferramentas também podem ser orientadas por métricas de tempo de espera desde o commit do código até o feedback, análise de testes instáveis e taxas de falha de integração. Ferramentas que permitem ciclos de feedback curtos, como painéis de controle, análise automatizada da causa-raiz ou seleção inteligente de testes, podem apoiar diretamente o aprendizado da equipe e permitir respostas mais rápidas.

Qualquer ação de melhoria deve ser mensurável e validada. Para isso, as retrospectivas podem desempenhar um papel vital. As métricas devem ter rastreabilidade até os objetivos do teste e as prioridades de risco, em vez de referências arbitrárias. Modelos como o GQM (Goal-Question-Metric) são úteis para estruturar o uso de métricas de maneira orientada a objetivos. Por fim, as equipes devem manter o pensamento crítico e a sensibilidade contextual para evitar que as métricas se desconectem da missão do teste.

3.4.2 Melhoria de Processo de Teste no Desenvolvimento Ágil de Software

Para a melhoria de processo de teste no desenvolvimento ágil de software, as equipes podem adotar abordagens estruturadas, porém ágeis, que enfatizam a colaboração, a consciência do contexto e a experimentação iterativa:

Realize retrospectivas focadas. Utilizar práticas leves e colaborativas, como retrospectivas específicas para testes, para identificar desafios de teste e oportunidades de aprimoramento. Os esforços de melhoria devem estar fundamentados no contexto atual da equipe, não impostos de fora. Retrospectivas focadas especificamente em testes, distintas das retrospectivas Ágeis gerais, podem fornecer insights direcionados sobre a efetividade das práticas de teste e seu alinhamento com as necessidades comerciais e técnicas (ISTQB-ATLAS, v1.0), além de ajudar a construir um senso de propriedade compartilhada das metas de qualidade.

Estabeleça uma avaliação de linha de base. Realize avaliações de testes Ágeis estruturadas para identificar lacunas na abordagem de teste atual. Ferramentas como autoavaliações ou listas de verificação podem estimular a reflexão sobre dimensões centrais dos testes Ágeis, como colaboração, ciclos de feedback e integração contínua. A partir daí, melhorias direcionadas podem seguir um caminho gradual e iterativo.

Alinhe-se por meio de workshops conjuntos sobre riscos. Inclua representantes dos negócios, dos testes e do desenvolvimento para construir um entendimento comum dos riscos de produto críticos e onde o esforço de teste deve ser mais bem focado. Isso garante que as melhorias apoiem tanto as metas de negócios quanto as da equipe.

Fortaleça os ciclos de feedback de CI/CD. Integre práticas como abordagens de testar primeiro e testes em pares para possibilitar um feedback precoce e acionável. Use testes exploratórios não apenas para descobrir defeitos, mas para descobrir riscos desconhecidos e lacunas nos testes, apoiando o refinamento adaptativo do processo de testes.

Realize pequenos experimentos. Adote técnicas, como gerenciamento de teste baseado em sessão, para explorar mudanças no processo de teste de forma incremental. Os resultados desses experimentos devem ser discutidos abertamente dentro da equipe, utilizando retrospectivas para ajustar a estratégia de teste conforme necessário.

Utilize métricas visíveis com cuidado. Monitore a integridade da automação de testes, as tendências de defeitos e a cobertura em todos os pipelines de CI/CD, mas resista à tentação de comparar equipes entre si.

Em vez disso, concentre-se em métricas que ajudem as equipes a compreender sua própria efetividade e tornem seu trabalho mais visível.

As equipes podem usar práticas como Mapeamento do Fluxo de Valor, ciclo PDCA ou diagramas de loop casual (ver (ISTQB-ATLAS, v1.0)). As melhorias devem levar em conta fatores sociotécnicos, como cultura organizacional e práticas de comunicação. A resistência à mudança (ver: (Prosci, 2026)) deve ser abordada por meio de conversas inclusivas que se baseiem em estratégias de facilitação da mudança.

Acima de tudo, a melhoria contínua nos testes ágeis deve estar enraizada no pensamento crítico, na aprendizagem contínua e na colaboração de toda a equipe. A aplicação de modelos ou processos fixos deve ser cuidadosamente adaptada ao contexto. Cada melhoria deve funcionar como uma hipótese a ser testada por meio da ação, da aprendizagem e da adaptação.

4 Shift Left - 135 minutos

Palavras-chave

desenvolvimento orientado por teste de aceite, desenvolvimento orientado pelo comportamento, qualidade, abordagem "shift left", base de teste, testboarding, testware

Palavras-chave específicas do domínio

viés, mapeamento de exemplo, especificação por exemplo, storyboarding, conversa estruturada, folha de sessão de teste, jornada do usuário, história do usuário, divisão da história do usuário

Objetivos de aprendizagem:

4.1 Usando o shift left para melhorar a qualidade da base de teste

CTAL-AT-4.1.1 (K2) Dar exemplos de como o testware pode ser usado como uma forma de requisitos

CTAL-AT-4.1.2 (K2) Explicar como o storyboarding e o testboarding podem ser usados para aumentar a qualidade da base de teste

CTAL-AT-4.1.3 (K2) Explicar como o mapeamento de exemplo pode ser usado para aumentar a qualidade da base de teste

CTAL-AT-4.1.4 (K2) Dar exemplos de como os vieses podem afetar negativamente a qualidade do produto

CTAL-AT-4.1.5 (K3) Aplicar a divisão de histórias do usuário para obter histórias do usuário com testabilidade

4.2 Shift Left e Engenharia de Requisitos

CTAL-AT-4.2.1 (K2) Explicar como a engenharia de requisitos apoia o shift left

Introdução

O deslocamento para a esquerda enfatiza a incorporação de atividades de qualidade o mais cedo possível no ciclo de vida de desenvolvimento de software, com foco na melhoria da base de teste desde o início. Testware, como critérios de aceite, listas de verificação e scripts de teste automatizados, podem atuar como requisitos dinâmicos ou apoiar e documentar requisitos, garantindo clareza e entendimento comum. Técnicas como storyboarding e testboarding ajudam a visualizar fluxos de trabalho e alinhar perspectivas, enquanto o mapeamento de exemplo refina os critérios de aceite por meio de discussões colaborativas. A consciência dos vieses cognitivos protege contra suposições falhas que podem reduzir a qualidade do produto. Práticas como a segmentação de histórias do usuário garantem que as histórias do usuário sejam pequenas, focadas e com testabilidade. Paralelamente, a engenharia de requisitos eficaz apoia o shift left combinando análise, documentação e validação, e selecionando métodos de elicitação adequados, como entrevistas, workshops ou prototipagem, para capturar as necessidades dos stakeholders. Ao adotar o shift left, a equipe pode detectar anomalias e defeitos mais cedo, reduzir o retrabalho e entregar produtos de maior valor. Ferramentas emergentes de IA e de grandes modelos de linguagem podem apoiar o “shift left” ao analisar requisitos em busca de ambiguidades, sugerir cenários de teste a partir de critérios de aceite ou identificar lacunas na cobertura das histórias do usuário. Os testadores devem avaliar essas ferramentas criticamente, compreendendo suas limitações, e usá-las para complementar, e não substituir, a análise humana colaborativa.

4.1 Usando o Shift Left para Melhorar a Qualidade da Base de Teste

4.1.1 Testware como requisitos

No desenvolvimento ágil de software, testware como histórias de usuário com critérios de aceite, listas de verificação, cenários BDD, casos de teste de nível de aceite, exemplos, cartas de teste, contratos ou scripts de teste automatizados podem ser tratados como uma forma de requisitos quando usados de forma colaborativa e submetidos a revisões e aplicados a cada iteração. Esses produtos de trabalho ajudam as equipes a esclarecer expectativas, promover um entendimento compartilhado e apoiar a validação contínua do comportamento do software. Eles são frequentemente criados durante conversas com as stakeholders do negócio, tornando-os requisitos vivos em vez de documentação estática.

Os requisitos utilizados no desenvolvimento orientado pelo comportamento (BDD), no desenvolvimento orientado por teste de aceite (ATDD) ou na especificação por exemplo (Adzic, 2011) são derivados de conversas entre a equipe e as stakeholders do negócio, e são frequentemente documentados em um formato que pode ser executado como testes. Essas especificações executáveis incorporam os critérios de aceite, servindo tanto como requisitos dinâmicos quanto como testes de regressão duradouros.

Nos testes exploratórios, as cartas de teste baseadas em sessões geralmente surgem de ideias de produto de alto nível ou avaliações de risco. Essas cartas de teste orientam a investigação e ajudam a identificar funcionalidades ausentes ou mal compreendidas. Os registros de sessão de teste gerados durante a sessão de testes exploratórios fornecem insights informais, mas valiosos, sobre requisitos em evolução.

Nos modelos DevOps e de entrega contínua, logs de monitoramento, análises de clientes e ciclos de feedback podem servir como requisitos em evolução, quando interpretados e traduzidos em hipóteses testáveis ou regras de decisão, especialmente quando a mudança é impulsionada pelo comportamento da produção e pelos dados de interação do usuário. As equipes podem capturar essas percepções em casos de teste ou listas de verificação para validação contínua.

Outras formas de testware, como mapas mentais, listas de verificação e modelos visuais (por exemplo, diagramas de fluxo ou diagramas de estado), podem ser usadas durante o planejamento ágil e podem orientar diretamente o trabalho de desenvolvimento. Elas traduzem o entendimento implícito em pontos de referência compartilhados.

4.1.2 Storyboarding e Testboarding

Storyboarding e testboarding (testboarding) são técnicas colaborativas que melhoram a qualidade da base de teste ao aprimorar o entendimento compartilhado e a detecção precoce de ambiguidades, riscos e omissões. Do ponto de vista do teste de software, elas são particularmente efetivas em equipes Ágeis focadas no feedback e na colaboração de toda a equipe.

O storyboarding visualiza as jornadas do usuário apresentando o comportamento esperado como uma sequência de etapas ou cenas (Walker et al., 2013). Essa abordagem narrativa revela suposições, casos de borda e interações que podem não ser óbvias nos requisitos textuais. Ela suscita questões críticas sobre a lógica do fluxo de trabalho, permitindo que os stakeholders descubram inconsistências ou fluxos incompletos antes que o código seja escrito. Como os storyboards se concentram nas interações do usuário, eles ajudam a revelar lacunas na base de teste relacionadas à usabilidade, transições de fluxo e necessidades específicas de cada persona.

O testboarding é uma prática simples que ajuda a visualizar o que precisa ser testado para cada recurso, complementando o storyboarding. Além de validar os critérios de aceite, o testboarding mostra os cenários principais e alternativos, o que leva a um alinhamento precoce entre testadores, desenvolvedores e representantes de negócio em relação à terminologia, ao comportamento do sistema e às prioridades de teste. Esse entendimento compartilhado reduz erros de interpretação durante a modelagem de teste e a execução de teste, aumentando assim a manutenibilidade dos casos de teste.

Ao combinar essas técnicas, as equipes criam um espaço compartilhado para o pensamento exploratório. Os testadores podem fazer perguntas do tipo “e se?” e propor cenários negativos ou alternativos, promovendo melhorias antecipadas na história do usuário e em seus critérios de aceite. Práticas que ajudam a explorar a base de teste antecipadamente melhoram a identificação de risco e reduzem a descoberta tardia de defeitos críticos, permitindo que os testadores priorizem os testes adequadamente.

Essas técnicas são mais poderosas quando usadas de forma iterativa — revisitar storyboards e testboarding durante o refinamento do backlog e o planejamento de iterações expõe mudanças no escopo, nos riscos ou no entendimento. Elas podem servir como documentação viva que evolui junto com o produto e a base de teste, ajudando os testadores a detectar quando os requisitos se desviam ou as metas de qualidade são comprometidas.

4.1.3 Mapeamento de Exemplo

O mapeamento de exemplo (Wynne, 2015) é uma técnica colaborativa que refina histórias do usuário por meio de conversas estruturadas. Ela ajuda as equipes a identificar regras de negócio, exemplos associados (que podem se tornar casos de teste) e a registrar perguntas para aprofundar o entendimento compartilhado. A equipe cria cartões coloridos para representar os diferentes elementos da discussão (por exemplo, amarelo para a história do usuário em revisão, azul para regras de negócios, verde para exemplos que ilustram as regras e vermelho para perguntas). As regras são agrupadas sob uma história do usuário, e os exemplos são agrupados sob as regras correspondentes. A equipe continua criando cartões até estar satisfeita com a clareza do escopo da história do usuário ou até que o tempo limite expire. Quando o tempo acaba, isso pode indicar que a história do usuário é muito complexa ou precisa de refinamento, e a equipe decide os próximos passos.

À medida que a conversa flui, a equipe constrói rapidamente uma representação visual da história do usuário que reflete seu entendimento compartilhado atual. Um número excessivo de cartões de “pergunta” sugere que a equipe ainda tem muito a aprender sobre essa história do usuário. Muitos cartões de “regra”, ou uma única regra com muitos cartões de “exemplo”, sugerem que a história do usuário é grande e complexa e deve ser dividida (ver *Seção 4.1.5*). Nesses casos, a equipe divide as regras e os exemplos entre a história original e a nova parte, cria outro cartão amarelo (história do usuário) para a história do usuário dividida e o coloca no backlog.

Do ponto de vista dos testes, o mapeamento de exemplo fortalece a base de teste ao fundamentar requisitos abstratos em exemplos concretos e acordados. Esses exemplos reduzem a ambiguidade, revelam suposições ocultas e identificam interpretações conflitantes antecipadamente.

No desenvolvimento ágil de software, a base de testes frequentemente evolui durante o refinamento e a implementação. O mapeamento de exemplo melhora a qualidade ao garantir um entendimento comum em

toda a equipe e ao esclarecer os critérios de aceite que orientam diretamente o desenvolvimento orientado por teste de aceite (ATDD) e o desenvolvimento orientado pelo comportamento (BDD). Ele promove a validação antecipada ao revelar lacunas na especificação antes do início da codificação.

O mapeamento de exemplo apoia a criação de cartas de teste para testes exploratórios e ajuda os testadores a identificar condições de teste relevantes e cenários alternativos, já que os exemplos concretos e as perguntas sugerem naturalmente limites testáveis e casos de borda a serem explorados. Quando usado de forma iterativa, torna a base de teste mais resiliente às mudanças, pois a rastreabilidade dos exemplos até as regras e as necessidades do usuário é preservada ao longo do desenvolvimento.

4.1.4 Vieses nos testes ágeis

Um viés é um desvio sistemático da neutralidade ou precisão que afeta a maneira como as pessoas pensam, julgam ou agem, às vezes afastando-as da objetividade ou da imparcialidade. Os vieses podem comprometer significativamente a qualidade do produto ao distorcer a percepção de risco, limitar a diversidade dos testes e enfraquecer os mecanismos de feedback essenciais às práticas de desenvolvimento ágil de software. Nos testes, os vieses podem ter origem em indivíduos, equipes ou na cultura organizacional e influenciar decisões inconscientemente.

Os preconceitos particularmente relevantes para os testes ágeis incluem:

- O viés de confirmação pode levar à concepção e execução de testes que apenas verificam o comportamento esperado, negligenciando possíveis caminhos de falha ou condições incomuns. Isso restringe o escopo da exploração e aumenta o risco de defeitos não detectados, especialmente em sistemas complexos onde suposições implícitas orientam as decisões de desenvolvimento.
- O efeito de ancoragem ocorre quando requisitos iniciais ou expectativas dos stakeholders fixam as equipes em interpretações específicas, ignorando comportamentos alternativos. Por exemplo, se as histórias do usuário iniciais se concentrarem exclusivamente em caminhos favoráveis, os testes podem sub-representar casos de borda, acessibilidade ou questões de internacionalização.
- O viés de conformidade pode prejudicar a qualidade quando opiniões diversas são desencorajadas, os testadores copiam os comportamentos ou crenças uns dos outros em vez de seguirem seu próprio julgamento independente, ou os testadores se submetem às vozes dominantes em equipes multifuncionais. Isso compromete o valor da abordagem de equipe como um todo, que depende da responsabilidade compartilhada pela qualidade e de ciclos críticos de feedback ao longo do ciclo de vida de desenvolvimento de software.

Os vieses podem afetar o projeto dos critérios de aceite, reduzindo a clareza e aumentando a ambiguidade em testes de aceite automatizados ou especificações executáveis. Em práticas como especificação por exemplo e ATDD, deixar de questionar suposições durante a especificação colaborativa pode levar a uma cobertura ineficaz.

O risco de preconceito é aumentado pela abordagem de equipe inteira. Para mitigar o preconceito, as equipes devem promover a diversidade de pensamento, aproveitar os testes exploratórios com testes baseados em sessões e fomentar uma cultura de segurança psicológica que incentive a dissidência e o questionamento. Práticas como reuniões de avaliação após sessões de testes exploratórios, testes em pares, o uso de testadores iniciais e a análise da causa-raiz podem ajudar a revelar a influência do preconceito e fortalecer a consciência coletiva.

4.1.5 Divisão de Histórias do Usuário

A divisão de histórias do usuário é a prática de dividir grandes histórias do usuário em partes menores e mais gerenciáveis que agreguem valor, sejam viáveis dentro de uma iteração e tenham testabilidade. Histórias do usuário devidamente divididas facilitam a entrega contínua e permitem um feedback mais precoce, apoiando assim os princípios Ágeis.

Para aplicar o particionamento de histórias do usuário de forma a apoiar a testabilidade, a equipe começa identificando as diferentes dimensões ao longo das quais uma história do usuário pode ser dividida:

- **Segmentação de fluxo de trabalho** decompõe a história do usuário com base nas tarefas do usuário ou nas etapas de um processo de negócios. Cada segmento deve se concentrar em uma

única tarefa ou interação, idealmente aquela que resulte em um comportamento observável do sistema. Isso facilita a definição de critérios de aceite precisos e resultados esperados.

- **Divisão por complexidade** de dados começa com um único exemplo de caminho ideal usando os dados mais simples e move condições alternativas de dados e casos de borda para histórias do usuário separadas. Isso permite que a equipe verifique a funcionalidade central antecipadamente e construa a cobertura de forma incremental. Cada parte deve incluir variação suficiente para permitir testes significativos sem sobrecarregar o escopo.
- **Divisão por interface** separa a lógica de processamento de back-end da funcionalidade da interface do usuário (UI) quando cada uma pode ser testada independentemente. Isso permite a automação de testes mais cedo e se alinha à camada arquitetônica, especialmente em ambientes de integração contínua. A equipe constrói histórias de usuário de back-end que expõem saídas via APIs antes de trabalhar em histórias de usuário que incluem fluxos de trabalho da UI.
- **Divisão por cenário** começa com o cenário de negócios mais comum e se desenvolve a partir dele, criando fatias adicionais para casos menos frequentes ou mais complexos. A equipe pode usar mapeamento de exemplo ou conversas estruturadas para identificar esses cenários durante o refinamento. Cada fatia de cenário deve ser verificável por meio de exemplos específicos que formam a base para testes voltados para os negócios.
- **Divisão vertical** garante que cada história do usuário inclua uma fatia fina e completa de funcionalidade entre as camadas, do banco de dados à interface do usuário, quando apropriado. Isso ajuda as equipes a entregar valor demonstrável, realizar testes de ponta-a-ponta e verificar a integração entre os componentes. No entanto, deve-se evitar incluir elementos demais em uma única fatia. Se o escopo de ponta-a-ponta se tornar incontrolável, a equipe deve reverter para a divisão por cenário ou fluxo de trabalho.
- **Divisão baseada em restrições** concentra-se primeiro em suposições gerais e, em seguida, considera condições derivadas de restrições conhecidas. Por exemplo, a equipe começa com uma história do usuário que pressupõe que o usuário está autenticado e, em seguida, divide uma história do usuário separada para lidar com falhas de autenticidade. Isso ajuda a isolar as condições de teste e melhora a clareza ao definir limites.

Os critérios de aceite frequentemente determinam o sucesso ou a falha da divisão da história do usuário. Ao dividir, a equipe não pode simplesmente cortar a história do usuário; ela precisa repensar e reescrever os critérios de aceite para que cada divisão faça sentido. Por exemplo, histórias do usuário grandes geralmente têm um conjunto de critérios de aceite amplos que devem ser refinados após a divisão para corresponder ao escopo mais restrito. Os critérios de aceite devem se concentrar em um único resultado específico para o usuário por parte, e deve-se evitar combinar múltiplas variações comportamentais em uma única parte. Ao dividir, os critérios de aceite originais não devem ser copiados e colados em cada parte. Em vez disso, eles devem ser reescritos para cada parte, de modo que cada um tenha testabilidade, possa ser demonstrado e corresponda à sua parte. O conjunto de critérios de aceite entre as partes deve, eventualmente, cobrir a história do usuário completa.

Independentemente da estratégia de divisão, a equipe deve sempre considerar a testabilidade como um critério primário. Isso é feito perguntando se a fatia pode ser validada de forma independente, se critérios de aceite claros podem ser derivados e se ela permite a criação de testes voltados para o negócio. Se alguma dessas condições não for atendida, a história do usuário precisa de refinamento adicional.

4.2 Shift Left e Engenharia de Requisitos

O Shift Left antecipa as atividades de qualidade no ciclo de vida de desenvolvimento de software para detectar anomalias e defeitos precocemente, reduzir retrabalho, encurtar ciclos de entrega e melhorar o alinhamento com as necessidades de negócios. Todas as principais atividades da engenharia de requisitos — elicitação, análise, especificação e validação de requisitos — podem apoiar o Shift Left ao reduzir mal-entendidos, prevenir defeitos e minimizar retrabalhos onerosos mais adiante no ciclo de vida de desenvolvimento de software:

- A elicitação de requisitos apoia o shift left ao capturar as necessidades dos stakeholders de forma precoce e contínua, permitindo que o desenvolvimento e os testes comecem com uma compreensão clara das expectativas dos stakeholders. A elicitação contínua por meio de histórias do usuário, organização do backlog e workshops colaborativos garante que os recursos sejam definidos, com testabilidade e validados, reduzindo defeitos e retrabalho mais tarde na iteração.
- A análise de requisitos apoia o shift left ao identificar ambiguidades, conflitos, riscos e dependências desde o início, garantindo que o desenvolvimento e os testes comecem com requisitos claros, viáveis e priorizados. Ao analisar os requisitos, as equipes podem projetar soluções adequadas, planejar testes e prevenir defeitos antes do início da codificação, reduzindo retrabalhos dispendiosos mais adiante no ciclo de vida de desenvolvimento de software.
- A especificação de requisitos apoia o shift left ao fornecer documentação clara, precisa e testável do que o sistema deve fazer, permitindo que desenvolvedores e testadores planejem e validem o trabalho antecipadamente. Requisitos bem especificados reduzem mal-entendidos, orientam a criação antecipada de testes e ajudam a identificar anomalias antes da implementação, minimizando defeitos e retrabalho a jusante. Testadores ágeis normalmente trabalham com representações textuais de formato livre, como histórias do usuário e épicos. Quando é necessária mais estrutura, eles podem usar modelos de cenário semiformais, como casos de uso, modelos de processos de negócios (por exemplo, BPMN) ou diagramas de transição de estado, dependendo do contexto. Modelos de cenário podem ilustrar fluxos de interação, incluindo casos de borda que podem não ser capturados em descrições de alto nível. Glossários são frequentemente mantidos para garantir um entendimento comum de termos específicos do domínio, especialmente ao definir critérios de aceite.

A validação de requisitos apoia uma abordagem “shift left”, garantindo que os requisitos sejam corretos, inequívocos, completos, consistentes e com testabilidade antes do início do desenvolvimento. A validação precoce por meio de revisões, prototipagem e modelagem ajuda a detectar anomalias e lacunas, reduzindo defeitos, retrabalho e alterações dispendiosas em fases tardias.

5 Abordagens Ágeis e Técnicas de Teste - 285 minutos

Palavras-chave

Desenvolvimento ágil de software, caso de borda, teste exploratório, mob testing, teste em pares, caso de teste, carta de teste, teste heurístico, teste mnemônico, indicador de anomalia, passeio, vibe testing

Palavras-chave específicas do domínio

épico, FEW HICCUPPS, I SLICED UP FUN, RCRCRC, SFDIPOT, TERMS

Objetivos de aprendizagem:

5.1 Teste exploratório

CTAL-AT-5.1.1 (K2) Explicar testes heurísticos

CTAL-AT-5.1.2 (K2) Dar exemplos de testes mnemônicos relacionados aos testes no desenvolvimento ágil de software

CTAL-AT-5.1.3 (K2) Explicar os passeios de teste

CTAL-AT-5.1.4 (K4) Analisar histórias do usuário e epics para criar cartas de teste

CTAL-AT-5.1.5 (K3) Aplicar testes exploratórios para apoiar os testes no desenvolvimento ágil de software

5.2 Teste assistido

CTAL-AT-5.2.1 (K2) Explicar o mob testing

CTAL-AT-5.2.2 (K2) Explicar o teste em pares

CTAL-AT-5.2.3 (K2) Explicar o vibe testing

5.3 Indicadores de anomalias (test smells)

CTAL-AT-5.3.1 (K3) Usar os indicadores de anomalias para avaliar a qualidade dos casos de teste

Introdução

Os testadores devem adaptar suas estratégias de teste e ferramentas de teste para acompanhar a natureza ágil e iterativa dos projetos de desenvolvimento de software. Este capítulo explora um conjunto de técnicas de teste e métodos de colaboração que aumentam a efetividade e a eficiência dos testes no desenvolvimento ágil de software. Eles enfatizam a flexibilidade, o feedback em tempo real e o aprendizado contínuo – valores fundamentais das metodologias de desenvolvimento ágil de software.

5.1 Teste Exploratório

5.1.1 Teste Heurístico

Heurísticas de teste são diretrizes gerais ou regras práticas que ajudam os testadores a tomar decisões informadas, resolver problemas e orientar sua exploração de um sistema em teste. Não há garantia de que sejam corretas em todos os contextos, mas servem como estratégias práticas que frequentemente levam a resultados valiosos quando:

- o tempo ou os recursos são limitados;
- a especificação é incompleta ou pouco clara;
- o testador realiza testes baseados na experiência;
- no teste de regressão é realizada a cobertura total de automação de testes;
- o testador deseja pensar como um usuário ou utilizar um simulador para simular casos de uso do mundo real.

Exemplos de testes heurísticos incluem:

Diretrizes – procedimentos ou etapas recomendados que orientam os testadores sobre ações específicas a serem tomadas ou abordagens a serem seguidas nos testes. Exemplos incluem: “teste cedo e com frequência”, “teste é sobre informação, não apenas aprovado/reprovado”, “sempre considere os limites”, “teste fluxos de trabalho comuns e comportamentos inesperados ou inválidos”, “altere uma variável de cada vez para isolar efeitos ao testar opções, configurações ou entradas”, “teste como os recursos funcionam juntos em sequência, não apenas isoladamente”.

Listas de verificação genéricas – listas reutilizáveis de itens comuns a serem testados, usadas como lembretes ou sugestões para explorar áreas que, de outra forma, poderiam ser esquecidas. As listas de verificação podem se concentrar em várias áreas, como características de qualidade, tipos de teste, níveis de teste, áreas de risco, funções do usuário e ambientes de teste. Exemplos incluem as heurísticas de usabilidade de Nielsen (Nielsen, 1994), os 10 principais riscos de segurança de aplicativos web da OWASP (OWASP, 2025), taxonomias de defeitos como a taxonomia de Beizer (Beizer, 1990) e a taxonomia IEEE 1044 (*IEEE 1044 – Classificação de Anomalias de Software*, 2009).

Regras práticas – “leis” empíricas não escritas ou verdades comumente aceitas. Exemplos incluem: “se algo parece simples demais, provavelmente esconde complexidade”, “se já quebrou antes, provavelmente quebrará de novo”, “defeitos se agrupam”, “se é difícil de testar, provavelmente é difícil de usar”, “o que funciona com um usuário pode falhar com muitos”.

Mnemônicos – recursos de memória que ajudam os testadores a lembrar rapidamente de um conjunto de condições de teste. Exemplos incluem SFDIPOT (ver *Seção 5.1.2*).

Analogias e metáforas – permitem que os testadores comparem o sistema a conceitos familiares e, então, apliquem o raciocínio desses conceitos para gerar condições de teste. Exemplos incluem passeios (ver *Seção 5.1.3*).

5.1.2 Mnemônicos de teste

Mnemônicos são auxiliares de memória, geralmente na forma de acrônimos, abreviações, rimas ou padrões. Mnemônicos não prescrevem rigidamente o que testar, mas ajudam os testadores a lembrar conjuntos complexos de informações e ampliar sua perspectiva durante testes exploratórios e revisões. No entanto, eles exigem conhecimento do domínio, do contexto e dos riscos, bem como pensamento crítico e confiança da equipe para serem eficazes. Em testes de software, testes mnemônicos são comumente usados para:

- orientar os testes exploratórios, fornecendo estrutura para testes de formato livre sem casos de teste rígidos;
- estimular o pensamento criativo e crítico, levando os testadores a pensar de maneiras que talvez não tivessem considerado naturalmente;
- alcançar uma cobertura mais ampla, garantindo que áreas-chave não sejam esquecidas;
- servir como listas de verificação mentais durante o planejamento de teste ou a execução de teste, reduzindo a carga cognitiva ao condensar conceitos complexos em formatos fáceis de lembrar.

O teste de software emprega vários testes mnemônicos, e os testadores podem criar seus próprios conjuntos personalizados. Alguns testes mnemônicos populares incluem:

- SFDIPOT (Structure, Function, Data, Interfaces, Platform, Operations, Time) – apoia a análise geral do sistema em teste (Bolton, 2014). Pode ser usado ao explorar um novo sistema ou planejar cartas de teste.
- RCRCRC (Recent, Core, Risky, Configuration, Repaired, Chronic) – auxilia no teste de regressão, onde cada letra representa uma palavra para ajudar a descobrir condições de teste (Johnson, 2012). Pode ser usado para decidir a cobertura em iterações de desenvolvimento ágil de software ou ciclos de testes de regressão.
- I SLICED UP FUN (Inputs, Store, Location, Interactions/Interruptions, Communication, Ergonomics, Data, Usability, Platform, Function, User scenarios, Network) (Kohl, 2010). Pode ser usado para apoiar testes de aplicativos móveis na geração de condições de teste.
- FEW HICCUPPS (Familiarity, Explainability, World, History, Image, Comparable products, Claims, Users' desires, Product, Purpose, Statutes) – auxilia na identificação e aplicação de oráculos de teste (Bolton, 2012). Pode ser usado para identificar defeitos por meio da observação de diferentes tipos de inconsistências.
- TERMS (Tools and Technology, Execution, Requirements and Risks, Maintenance, Security) (Gareev, 2019). Ajuda a lembrar diferentes fatores que influenciam o sucesso da automação de testes.

5.1.3 Passeios de teste

Os passeios de teste utilizam metáforas de passeios turísticos para orientar os testes exploratórios de um sistema. O conceito visa ajudar os testadores a explorar o software, da mesma forma que turistas exploram um novo lugar, utilizando diferentes tipos de "passeios" para identificar diversos tipos de problemas. Cada passeio se concentra em uma perspectiva específica, orientando o testador a examinar a aplicação a partir de um ponto de vista particular. Os passeios são projetados para serem informais, leves e criativos, tornando-os ideais para testes exploratórios, testes baseados em sessão ou como geradores de condições de teste.

Exemplos de passeios de teste, adaptados de bairros da cidade (Whittaker, 2009), incluem:

Distrito comercial. Para o software, isso se refere aos recursos principais. Essa área é delimitada pelos códigos de inicialização e desligamento e contém os recursos e funções com os quais os usuários interagem no software.

Distrito histórico. No caso de software, isso se refere ao código legado e ao histórico de funções ou recursos com defeitos. Assim como na história real, o código legado é frequentemente mal compreendido, e muitas suposições são feitas quando ele é incluído, modificado ou utilizado.

Distrito turístico. Muitas cidades têm distritos que os turistas visitam principalmente. Moradores locais e residentes tendem a evitar essas áreas. O software reflete isso, com usuários novatos sendo atraídos por recursos e funções que usuários experientes já não precisam.

Bairro de entretenimento. Nas cidades, esses bairros oferecem relaxamento e entretenimento. O software, assim como as cidades, também possui recursos de apoio. Esses passeios complementam os passeios por outros bairros e preenchem as lacunas de um plano de teste abrangente.

Bairro hoteleiro. Nas cidades, esses bairros são um local para descanso. Nos testes, é um local para o testador de software se afastar da funcionalidade principal e dos recursos populares e testar algumas das funções secundárias e de apoio que muitas vezes são negligenciadas ou sub-representadas em um plano de teste.

Bairros perigosos são arriscados, mas ainda assim atraem um certo tipo de turista. Os passeios por bairros perigosos são essenciais para os testadores, pois revelam áreas de vulnerabilidade que poderiam ser altamente problemáticas para os usuários se permanecessem no produto. Essa área se concentra principalmente no teste negativo e emprega técnicas de teste como adivinhação de erros ou ataques de falha.

5.1.4 Criação de Carta de Teste

As cartas de teste estão em estreita sintonia com os princípios do desenvolvimento ágil de software e oferecem uma abordagem estruturada, porém flexível, para os testes exploratórios (ver *Seção 5.1.5*). Uma carta de teste é uma declaração curta e focada que descreve a finalidade, o escopo de teste e os objetivos do teste. Ela atua como uma declaração de missão orientadora ou hipótese testável para o testador durante a sessão de teste. As cartas de teste podem ser criados com base em várias fontes, como histórias do usuário, critérios de aceite, riscos de produto ou metas de iteração. No mínimo, uma carta de teste contém:

- Missão/objetivos do teste – foco de alto nível ou metas testáveis para a sessão (por exemplo, “Validar o login do usuário sob carga”)
- Escopo – áreas específicas de interesse dentro do sistema em teste, nível de teste, condições de teste, técnicas de teste a serem utilizadas, critérios de saída, prioridades, o que será coberto e o que não será coberto e pode incluir outras informações, tais como:
- Ator – usuário pretendido do sistema
- Propósito – um objetivo que o ator deseja alcançar
- Configuração – o que precisa estar pronto para iniciar os testes exploratórios (por exemplo, ambiente de teste)
- Prioridade – importância relativa desta carta de teste, com base na prioridade da história do usuário associada ou no nível de risco
- Referência – especificações (por exemplo, história do usuário), riscos ou outras fontes de informação
- Dados – quaisquer dados necessários para executar a carta de teste
- Atividades – uma lista de ideias sobre o que o ator pode querer fazer com o sistema, o que seria interessante testar e quais técnicas de teste poderiam ser utilizadas
- Notas do oráculo de teste – como avaliar o produto para determinar os resultados corretos do teste (por exemplo, capturar o que acontece na tela e comparar com o que está escrito no manual do usuário)
- Variações – ações e avaliações alternativas para complementar as ideias descritas em atividades
- Limitações – por exemplo, o que o produto nunca deve fazer
- Restrições e riscos – por exemplo, regulamentos, regras e normas utilizadas

Para derivar cartas de teste a partir de histórias de usuário e épicos, os testadores ágeis as analisam para identificar os objetivos do teste e o foco da missão. Eles extraem a intenção da história do usuário ou do épico (por exemplo, quem é o ator e qual é seu propósito) e a traduzem em uma declaração de missão para

a sessão de teste. Histórias de usuário relacionadas podem ser agrupadas para avaliar o comportamento integrado.

Os testadores ágeis identificam o que precisa ser aprendido, verificado ou explorado durante a sessão de teste. Eles podem usar a técnica 5W1H, também conhecida como método Kipling, para responder exaustivamente às perguntas existentes e gerar ideias que possam contribuir para a resolução de um problema:

- Qual recurso está sendo entregue?
- Quem são os usuários?
- Quando é usado?
- Onde é usado?
- Por que esse recurso é importante?
- Como ela se integra a outros recursos?

Essas perguntas ajudam a definir o escopo de teste, a configuração e a prioridade. Várias heurísticas de teste (ver *Seção 5.1.1*, *Seção 5.1.2* e *Seção 5.1.3*) podem ser úteis para responder a essas perguntas e gerar condições de teste, dados de teste e atividades de teste a serem incluídas na carta de teste.

Os critérios de aceite definem o que deve ser verdadeiro para que uma história do usuário seja aceita. Eles podem ser traduzidos diretamente em condições de teste. Embora os critérios de aceite orientem o comportamento esperado, eles também destacam áreas a serem exploradas além do “caminho ideal”: O que acontece se os critérios de aceite não forem atendidos? Que casos de borda ou condições de teste negativas surgem? Os critérios de aceite são claros ou possivelmente ambíguos? Isso ajuda os testadores a desenvolver condições de teste tanto para testes positivos quanto negativos. Os critérios de aceite frequentemente implicam ou especificam requisitos de dados, como valores de entrada válidos, pré-condições ou formatos ou intervalos específicos.

Os critérios de aceite também podem servir como notas de oráculo de teste simplificadas, pois definem comportamentos esperados, estabelecem limites de aprovação/reprovação e se alinham às expectativas dos representantes de negócio. No entanto, os testadores devem lembrar que os critérios de aceite não são oráculos de teste exaustivos — eles não capturam todos os aspectos de qualidade. Testes exploratórios ainda são necessários para descobrir defeitos fora do escopo dos critérios de aceite. Testadores ágeis devem tratar critérios de aceite ausentes, vagos ou ambíguos como oportunidades para testes exploratórios, a fim de descobrir variações não declaradas e casos de borda.

5.1.5 Realização de testes exploratórios

Em ambientes de desenvolvimento ágil de software, onde os requisitos mudam com frequência, os testes exploratórios oferecem a flexibilidade e a capacidade de resposta que muitas vezes faltam às abordagens de testes roteirizados. Os testes exploratórios também são importantes em projetos de desenvolvimento ágil de software devido ao tempo limitado disponível para análise de testes e aos detalhes limitados das histórias do usuário.

Os testes exploratórios podem ser aplicados em vários pontos do ciclo de vida de desenvolvimento ágil de software, incluindo:

- durante a execução da iteração (para testar rapidamente novos recursos ou alterações antes ou paralelamente aos testes roteirizados)
- em revisões/demonstrações de iteração (para explorar o produto informalmente enquanto se coleta feedback)
- após grandes alterações (para detectar defeitos de integração ou regressão que os testes roteirizados possam deixar passar)
- quando os critérios de aceite são vagos ou mínimos (para esclarecer as expectativas e identificar casos de borda), os testes exploratórios geralmente se baseiam em cartas de teste e são realizados em sessões de teste (Ghazi et al., 2017). Uma sessão de teste é um período com duração definida e

ininterrupta (geralmente de 60 a 120 minutos). Essas sessões de teste ajudam a organizar os esforços de teste e fornecem estrutura em uma abordagem de teste com flexibilidade. Uma sessão de teste concentra-se em uma carta de teste específica e é frequentemente documentada com planilhas de sessão de teste, observações e conclusões. A estrutura típica de uma sessão de teste exploratório é a seguinte:

- Preparação pré-sessão. A carta de teste é definida, o ambiente de teste é preparado e o tempo da sessão de teste é estabelecido.
- Fase de execução de teste. O testador aprende como o produto funciona, explora e avalia o produto. O testador segue a intenção da carta de teste, mas se adapta à medida que novas informações são descobertas. Ele usa heurísticas e testes mnemônicos para simular condições de teste. Como o teste exploratório não segue um roteiro, a documentação adequada é essencial para a reprodutibilidade, o relatório de teste e a melhoria. Os métodos padrão de anotação incluem planilhas de sessão, notas em formato livre, capturas de tela, gravações de tela e mapas mentais. Ferramentas de gravação de sessão também podem ser utilizadas. As informações documentadas geralmente incluem cobertura (incluindo cobertura de riscos), notas de avaliação, registro do comportamento real do sistema e anomalias detectadas.
- Revisão pós-sessão. O testador exploratório discute os resultados da sessão de teste com as stakeholders, incluindo:
    - a carta de teste versus a realidade (o testador seguiu a carta de teste, descobriu áreas inesperadas e atingiu as metas da sessão?);
    - defeitos encontrados;
    - questões levantadas (por exemplo, requisitos pouco claros ou incertezas técnicas);
    - próximos passos (são necessários novos planos de teste ou testes de regressão?).

Ao realizar testes exploratórios, o testador não está limitado de forma alguma no que diz respeito a abordagens de teste, técnicas de teste ou tipos de teste. Ele pode usar técnicas de teste caixa-preta e técnicas de teste caixa-branca. No entanto, devido à natureza dos testes exploratórios, os testes baseados na experiência e as heurísticas de teste são os mais comumente usados (ver *Seção 5.1.1*, *Seção 5.1.2* e *Seção 5.1.3*).

Os testadores exploratórios precisam aplicar oráculos de teste de forma flexível e contextual, muitas vezes contando com sua experiência, conhecimento do produto e colaboração da equipe. No teste exploratório, os oráculos de teste costumam ser:

- consistentes com as histórias do usuário ou critérios de aceite (ver *Seção 5.1.4*)
- expressam o senso comum ou as expectativas do usuário
- consistentes com recursos semelhantes ou versões anteriores
- seguem padrões ou diretrizes (por exemplo, padrões de acessibilidade ou de interface do usuário)

As categorias de oráculos de teste que apoiam os testes exploratórios podem incluir o teste mnemônico FEW HICCUPPS (ver *Seção 5.1.2*).

5.2 Testes Assistidos

5.2.1 Mob Testing

O mob testing (também conhecido como teste em conjunto) é uma abordagem de teste colaborativa na qual um pequeno grupo (normalmente de 5 a 8 pessoas) testa o mesmo sistema simultaneamente. Baseia-se no conceito de swarming, em que vários membros da equipe se reúnem para resolver um determinado problema de teste. Essa colaboração em tempo real combina diversas habilidades, perspectivas e feedback imediato para melhorar a qualidade, a velocidade e a criatividade dos testes.

A configuração do mob testing inclui uma tela compartilhada, um computador e um quadro branco (no caso de trabalho remoto, o ambiente pode ser virtual, em vez de físico). Ela utiliza as seguintes funções:

- Moderador – coordenador da sessão que supervisiona o andamento da sessão, garante o cumprimento das funções, gerencia o tempo e promove uma participação equilibrada.
- Navegador – o testador principal que toma as decisões finais sobre as ações, incorporando insights e orientações do grupo.
- Piloto – digitador que implementa as instruções coletivas da equipe e, normalmente, não assume a liderança na estratégia, garantindo que as ideias passem primeiro pela mente de outra pessoa.
- Mob – um grupo de observadores que monitora o progresso e contribui com insights conforme necessário.

O mob testing consiste em duas partes: o mobbing e a retrospectiva. A sessão de mobbing começa com as pessoas dispostas em círculo, com o piloto ao teclado. As pessoas se revezam a cada certo tempo (geralmente a cada 4 minutos); o navegador se torna o piloto, o piloto se torna a primeira pessoa do grupo, e assim por diante.

O moderador fica sentado atrás e não se alterna com o resto do mob. Se for necessário que ele intervenha, pode pausar o mob e assumir qualquer função necessária, exceto a de piloto. O mob se alterna com frequência, forçando todos a prestarem atenção. A sessão de mobbing geralmente dura 2 horas.

Após o término da sessão de mobbing, segue-se a retrospectiva para aprimorar o processo para sessões futuras. Normalmente, o moderador coleta silenciosamente as observações de cada participante em post-its. A equipe então agrupa itens semelhantes para revelar padrões e oportunidades de melhoria contínua.

5.2.2 Teste em Pares

O teste em pares é uma abordagem de teste na qual duas pessoas trabalham juntas em uma única estação de trabalho (ou em um ambiente virtual, no caso de trabalho remoto ou de uma equipe distribuída) para testar o mesmo objeto de teste em tempo real (por exemplo, testador + testador, testador + desenvolvedor, testador + analista de produto). Normalmente, no teste em pares, uma pessoa opera o computador (o condutor), e a outra observa, realiza revisões, toma notas e traça estratégias (o navegador). O teste em pares inspira-se na programação em pares, uma prática popularizada pela programação extrema (XP).

O teste em pares promove o compartilhamento de conhecimento, incentivando o aprendizado a partir da expertise de cada um e das estratégias de teste, e é particularmente útil na integração de novos membros da equipe. Trabalhar em pares ajuda a descobrir defeitos mais rapidamente devido à sinergia colaborativa que cria. Isso também melhora a cobertura ao combinar perspectivas diversas. O feedback em tempo real ajuda a refinar as condições de teste, o que leva a casos de teste mais criativos e eficazes. Também pode melhorar a comunicação, reduzindo mal-entendidos sobre requisitos ou funcionalidades. O teste em pares mantém os testadores engajados e reduz a probabilidade de omissão de etapas de teste. O ciclo de feedback imediato pode até mesmo desencadear correções instantâneas de defeitos (que ainda precisam ser relatadas adequadamente), especialmente em pares de desenvolvedor-testador.

O teste em pares é particularmente valioso quando a colaboração, o aprendizado e o feedback rápido são essenciais. Em particular, ele é útil em:

- teste exploratório
- teste de recursos complexos ou novos
- teste funcional (por exemplo, UI + API)
- orientação de testadores juniores ou treinamento cruzado de membros da equipe

O teste em pares não está isento de desafios. É demorado e exige muito esforço, já que duas pessoas trabalham na mesma tarefa, o que pode parecer menos eficiente do que trabalhar separadamente. Além disso, os testadores podem ter estilos de trabalho, níveis de assertividade ou formas de pensar diferentes. Isso pode ser uma vantagem, mas também pode levar a atritos, desconforto e redução da produtividade. Uma pessoa pode dominar a sessão (por exemplo, sempre conduzindo ou tomando decisões), enquanto falar constantemente, pensar em voz alta e justificar decisões pode ser mentalmente exaustivo.

Para superar esses desafios:

- o teste em pares deve ser usado estrategicamente – por exemplo, apenas para recursos complexos ou de alto risco
- deve-se praticar a comunicação aberta e a escuta ativa
- os papéis devem ser claramente definidos (condutor e navegador) e alternados regularmente para evitar padrões fixos de comportamento
- devem ser estabelecidas metas claras antes do início de cada sessão de teste em pares

5.2.3 Vibe Testing

O vibe testing é uma abordagem de teste emergente e informal, assistida por IA, que ainda não foi totalmente padronizada. Ela se desenvolveu paralelamente ao crescimento do código gerado por IA, comumente chamado de “vibe coding”. Normalmente, os desenvolvedores escrevem o código manualmente, criam casos de teste e seguem fluxos de trabalho estruturados de controle da qualidade. Com a codificação vibe, no entanto, os desenvolvedores descrevem o comportamento exigido pela aplicação em linguagem natural, normalmente como prompts para LLMs (Large Language Models), que então geram o código, muitas vezes reduzindo a necessidade de revisão humana detalhada. Essa mudança no desenvolvimento exige uma abordagem de teste — chamada de vibe testing — para verificar se o produto está alinhado com a intenção do usuário.

O vibe testing enfatiza a validação com foco na intenção, concentrando-se em testar o que o aplicativo deve fazer, em vez de apenas o que ele faz atualmente, em vez de depender de scripts de teste estáticos. Em vez de especificar meticulosamente cada caso de teste com antecedência, os testadores interagem com o aplicativo como usuários reais fariam, contando com testes exploratórios e casos de teste gerados por IA. O vibe testing acelera o processo de teste, torna os testes mais adaptáveis a aplicativos construídos com IA em rápida evolução, reduz a dependência da criação manual de testes e se alinha estreitamente ao estilo de design intuitivo e de alto nível da vibe coding. No entanto, ele também apresenta desafios. Como os desenvolvedores podem não compreender ou realizar a revisão totalmente do código subjacente, há um risco de defeitos ocultos, vulnerabilidades de segurança, alucinações de IA ou comportamentos mal interpretados. O vibe testing mitiga esse risco atuando como uma salvaguarda, garantindo que o aplicativo ofereça a funcionalidade pretendida, em vez de apenas o que a IA interpretou.

Por exemplo, suponha que o desenvolvedor tenha usado um LLM para gerar código com o prompt “Crie uma página de login com campos de e-mail e senha. Os usuários devem ser redirecionados para o painel de controle após o login bem-sucedido.” O testador usa prompts ou exploração manual para testar o comportamento do sistema da perspectiva do usuário.

Exemplos de prompts incluem: “Teste o login com formato de e-mail inválido e confirme se ele não prossegue para o painel de controle” ou “Após o login, confirme se o usuário vê o painel de controle correto com base na função (por exemplo, administrador versus usuário comum).” O prompt original do desenvolvedor não especificava regras de validação de entrada, tratamento de entradas incorretas ou lógica do painel de controle baseada em função. Essas são expectativas comuns, mas, a menos que explicitamente declaradas, o LLM pode ignorá-las. O vibe testing visa eliminar essas lacunas potenciais, testando o que o aplicativo deve fazer no mundo real, em vez de presumir que o LLM entendeu tudo corretamente a partir de um prompt vago.

5.3 Indicadores de Anomalias (test smells)

Um indicador de anomalia no teste é um sintoma de um teste mal projetado. Embora nem sempre indique um problema, sugere riscos potenciais e aumento dos custos de manutenção. Assim como os indícios de problema no código no desenvolvimento de software devem ser abordados para manter a qualidade do código, os indicadores de anomalias nos testes devem ser abordados para manter uma estratégia de teste eficaz e confiável. Este syllabus descreve apenas indicadores de anomalias em casos de teste manuais. Para casos de teste automatizados, consulte (Aljedaani et al., 2021).

Os seguintes indicadores de anomalias estão agrupados em categorias, cada uma com uma descrição do sintoma e uma solução recomendada:

Sinais de dependência e complexidade

Testes Interdependentes

- Indicador de anomalia: O sucesso de um teste depende de outro teste ser executado primeiro ou de deixar dados residuais.
- Solução: Torne os testes independentes, autônomos e executáveis em qualquer ordem.

Dependências Ocultas

- Indicador de anomalia: O caso de teste depende de alterações de dados ou configuração que não estão especificadas nas pré-condições.
- Solução: Torne as dependências explícitas e controladas.

Chamadas

- Indicador de anomalia: Dependência de chamadas extensas a outros casos de teste ou da passagem de parâmetros entre eles.
- Solução: Evite aninhar chamadas e encadear casos de teste; em vez disso, mantenha os testes simples e use nomenclatura e documentação claras.

Decisão

- Indicador de anomalia: Uso de lógica de decisão nas etapas de teste, como oferecer ações alternativas e múltiplos resultados esperados.
- Solução: Torne os casos de teste determinísticos e sem ramificações, dividindo-os em casos de teste separados, movendo a lógica de decisão para as pré-condições e evitando o uso de “ou” nos resultados esperados.

Sinais de alerta em resultados esperados

Resultados Esperados Não Claros

- Indicador de anomalia: Os procedimentos de teste listam ações, mas não definem claramente os critérios de aprovação/reprovação.
- Solução: Adicione resultados esperados, usando um formato estruturado que mantenha as expectativas visíveis e rastreáveis. Evite linguagem subjetiva e estabeleça rastreabilidade em relação aos requisitos ou critérios de aceite.

Mais um Passo

- Indicador de anomalia: Os resultados esperados incluem etapas de teste adicionais, frequentemente usando verbos de ação como “verificar”, “confirmar” e “observar”.
- Solução: Mantenha as ações e os resultados esperados claramente separados da execução de teste, movendo a verificação para uma etapa própria, e evite usar verbos de ação nos resultados esperados.

Etapas de teste ambíguas

- Indicador de anomalia: As instruções são vagas ou abertas a interpretações.
- Solução: Use linguagem específica e objetiva com termos e dados concretos. Defina claramente as entradas, evite palavras subjetivas e faça referência a elementos específicos da interface do usuário ou rótulos.

Hotstepper

- Indicador de anomalia: As etapas de teste listam cada ação separadamente, mesmo quando a ação não tem efeito.
- Solução: Junte as etapas que não têm efeito específico.

Etapas em massa

- Indicador de anomalia: Muitas ações estão incluídas em uma única etapa de teste.
- Solução: Use pré-condições quando apropriado e siga a regra “uma ação = uma etapa”, garantindo que os resultados esperados correspondentes sejam adicionados para cada ação.

Problemas de nível de detalhe

Caso de teste excessivamente longo ou detalhado

- Indicador de anomalia: Casos de teste com muitas etapas de teste que abrangem vários recursos ou casos de teste que incluem o caminho de navegação completo antes de cada pequena ação.
- Solução: Divida em casos de teste menores, modularize partes reutilizáveis, remova detalhes de navegação repetidos e use pré-condições de forma eficaz.

Caso de Teste "Deus"

- Indicador de anomalia: Um único caso de teste é usado para testar vários cenários alternativos ou até mesmo todo o sistema.
- Solução: Projete casos de teste pequenos, focados e independentes, organizados em suítes de teste.

Dados de Teste Codificados

- Sinal de alerta: Cada caso de teste inclui explicitamente todos os seus dados de teste.
- Solução: Use descrições de dados de alto nível quando apropriado e utilize parâmetros para separar os dados de teste do caso de teste.

Ausência de indícios

Suposições de Ambiente

- Indicador de anomalia: O caso de teste pressupõe condições específicas do ambiente de teste sem documentá-las.
- Solução: Documente explicitamente as pré-condições do ambiente e os requisitos de configuração.

Ausência de limpeza ou desmontagem

- Indicador de anomalia: O caso de teste não especifica como os dados de teste são tratados após a conclusão do teste.
- Solução: Defina procedimentos explícitos de limpeza ou desmontagem.

Sinais de excesso

Paraíso da Invalidação

- Indicador de anomalia: Cada tipo diferente de entrada inválida é tratado por um caso de teste separado.
- Solução: Combine o tratamento de entradas inválidas em um único caso de teste usando parâmetros.

Duplicação por Copiar e Colar

- Indicador de anomalia: Vários casos de teste diferem apenas em alguns valores de dados de teste.

- Solução: Refatore em um único caso de teste reutilizável e parametrizado para evitar a duplicação, garantindo ao mesmo tempo a rastreabilidade aos requisitos para realizar a medição da cobertura.

Problemas de formatação e linguagem

Formatação Desestruturada

- Indicador de anomalia: As etapas de teste não estão numeradas e não possuem títulos.
- Solução: Use modelos.

Erros Ortográficos

- Indicador de anomalia: Os casos de teste contêm erros ortográficos e palavras ou frases incompletas.
- Solução: Execute um corretor ortográfico.

Clique-Empurre-Pressione

- Indicador de anomalia: A terminologia e as formulações usadas nos casos de teste são inconsistentes.
- Solução: Use um glossário de expressões padrão.

6 Automação de Testes e Ferramentas de Teste - 30 minutos

Palavras-chave

Desenvolvimento ágil de software, automação de testes

Objetivos de aprendizagem:

6.1 Automação de testes no desenvolvimento ágil de software

CTAL-AT-6.1.1 (K2) Distinguir entre diferentes abordagens de automação de testes aplicáveis ao desenvolvimento ágil de software

6.2 Ferramentas de teste no desenvolvimento ágil de software

CTAL-AT-6.2.1 (K2) Dar exemplos de ferramentas de teste úteis em testes ágeis

Introdução

A automação de testes pode fornecer feedback rápido e confiável em ciclos de teste curtos. A automação de testes complementa, mas não substitui os testes exploratórios e os testes manuais. As ferramentas de teste auxiliam na colaboração, no rastreamento de requisitos, no monitoramento de desempenho e na geração de relatórios de teste. Ferramentas que se integram facilmente ao desenvolvimento ágil de software melhoram a visibilidade, aceleram o feedback e ajudam toda a equipe a compartilhar a responsabilidade pela qualidade.

6.1 Automação de Testes no Desenvolvimento Ágil de Software

A decisão de automatizar é motivada tanto pelo risco quanto pelo valor. A automação de testes é adequada para tarefas repetitivas, determinísticas e de alto valor em termos de redução de risco, especialmente quando falhas podem afetar funcionalidades críticas para os negócios. Testes que nunca falham oferecem pouco valor e trazem benefícios mínimos quando automatizados. Da mesma forma, testes cujo comportamento muda frequentemente são candidatos inadequados para a automação de testes, pois os custos de manutenção muitas vezes superam os benefícios.

Considera-se uma prática recomendada decidir, durante o planejamento ou refinamento da iteração, quais histórias do usuário automatizar e em que medida. Isso facilita uma estimativa mais precisa do tempo necessário para alcançar o aceite total da história do usuário, resultando em maior previsibilidade da entrega da iteração.

O desenvolvimento ágil de software adota diferentes abordagens de automação de testes dependendo do contexto, das habilidades da equipe, da criticidade do sistema e da frequência de entrega. Uma abordagem comum de teste é automatizar testes que forneçam feedback rápido e validação repetível das partes mais valiosas e estáveis do sistema. Isso normalmente inclui testes de componentes, testes de API e um conjunto cuidadosamente definido de testes de ponta-a-ponta, de acordo com a pirâmide de teste (ver: (ISTQB-CTFL, v4.0.1)). Testes exploratórios e testes pontuais (ou seja, testes realizados apenas uma vez para testar uma condição de teste em um determinado momento) são normalmente executados manualmente porque dependem da percepção humana, são impulsionados por comportamentos emergentes ou porque os custos de manutenção seriam muito altos. As mesmas considerações geralmente se aplicam aos testes de usabilidade.

O momento da automação de testes também é uma consideração fundamental. Em uma abordagem de testar primeiro, os testes são automatizados antecipadamente, antes ou enquanto o código é escrito. Isso permite que os testes orientem o desenvolvimento e forneçam verificação contínua. Em contrapartida, a automação de testes de regressão é frequentemente uma atividade que evolui durante ou após a iteração, dependendo de quando a funcionalidade se estabiliza.

Os testes ágeis integram a automação de testes ao longo da iteração, começando com a verificação da compilação na integração contínua, continuando com testes exploratórios apoiados por ferramentas de automação de testes e concluindo com suítes de testes de regressão automatizadas que confirmam a estabilidade do sistema antes do lançamento.

A automação de testes bem-sucedida no desenvolvimento ágil de software requer uma estratégia em várias camadas que se concentre em automatizar o que fornece feedback consistente e valioso, selecionar os testes certos para automatizar e integrar a automação de testes progressivamente ao longo do ciclo de vida de desenvolvimento de software.

A pirâmide de automação de testes Ágil, introduzida por Mike Cohn, promove uma estratégia de testes equilibrada ao priorizar testes de unidade rápidos e de baixo nível, apoiando-os com uma camada menor de testes de serviço e de integração, e mantendo apenas um conjunto mínimo de testes de interface do usuário de ponta-a-ponta, permitindo feedback rápido, custos reduzidos e maior estabilidade de CI/CD.

Outros syllabi do ISTQB oferecem conhecimento mais aprofundado sobre automação de testes (consulte: (ISTQB-TAE, v2.0), (ISTQB-TAS, v1.0)).

6.2 Ferramentas de Teste no Desenvolvimento Ágil de Software

As ferramentas de teste utilizadas nos testes ágeis abrangem várias categorias e dão suporte tanto à natureza iterativa quanto à colaborativa do desenvolvimento ágil. As equipes ágeis se beneficiam de ferramentas que facilitam a comunicação, o feedback contínuo e a automação de testes em todo o ciclo de vida de desenvolvimento de software. A seleção de ferramentas de teste deve estar alinhada com a estratégia de teste, as camadas de automação de testes e os objetivos de feedback, em vez de se basear em conveniência ou tendências.

Uma variedade de ferramentas oferece suporte às atividades de testes ágeis e à colaboração em toda a equipe, cada uma servindo a um propósito distinto dentro do processo de desenvolvimento e entrega de software:

As ferramentas de gerenciamento e acompanhamento de tarefas permitem que a equipe planeje, priorize e acompanhe histórias de usuários, tarefas de teste e defeitos, promovendo a transparência e um entendimento comum do progresso e dos riscos da equipe. Essas ferramentas são essenciais para manter o backlog, gerenciar defeitos e gerenciar as cartas de teste.

Ferramentas de comunicação e compartilhamento de informações apoiam a colaboração em tempo real, especialmente em equipes distribuídas. Elas aprimoram a interação durante sessões de refinamento, planejamento de iterações, revisões de testes e retrospectivas. Essas ferramentas ajudam os testadores a compartilhar insights, discutir defeitos e comunicar o feedback do cliente rapidamente por toda a equipe.

Ferramentas de modelagem de teste, implementação de testes e execução de testes apoiam a criação e a execução tanto de testes automatizados quanto de testes manuais. Isso inclui frameworks de automação de testes para testes de unidade, testes de API e automação de testes de interface gráfica. Elas também possibilitam práticas de especificação por exemplo, como ATDD e BDD, facilitando a colaboração entre os membros da equipe em relação aos critérios de aceite.

Ferramentas de integração contínua permitem a integração frequente de código e a detecção rápida de falhas, mantendo o software sempre pronto para implantação. Essas ferramentas são essenciais no desenvolvimento ágil de software, onde ciclos curtos de feedback apoiam a entrega iterativa e ajudam a mitigar riscos de regressão antecipadamente.

As ferramentas de gerenciamento de configuração garantem a consistência entre os ambientes de teste, gerenciando código, scripts de teste e alterações na infraestrutura. Essas ferramentas são frequentemente integradas a pipelines de CI/CD para apoiar as metas de DevOps e entrega contínua.

Ferramentas de testes exploratórios oferecem suporte ao gerenciamento de teste baseado em sessão, ao acompanhamento de cartas de teste, ao registro de planilhas de sessões de teste e à captura de telas. Essas ferramentas ajudam a capturar condições de teste e resultados do teste com o mínimo de sobrecarga de documentação.

Ferramentas de monitoramento e análise fornecem insights de nível de produção por meio do rastreamento de padrões de uso, taxas de defeitos e métricas de desempenho. Elas são especialmente úteis em equipes ágeis que praticam o monitoramento como teste e naquelas que aproveitam o feedback de usuários reais para refinar o comportamento do produto após a implantação.

Ferramentas de IA podem gerar automaticamente casos de teste a partir de alterações de código, requisitos ou padrões de comportamento do usuário; prever falhas de compilação usando alterações de código e dados históricos; analisar histórias de usuários ou requisitos para detectar ambiguidades, critérios de aceite ausentes ou terminologia inconsistente; e criar dados de teste realistas que abranjam casos de borda.

7 Referências

Normas

IEEE 1044 – Classificação de Anomalias de Software, 2009. Norma. Instituto de Engenheiros Eletricistas e Eletrônicos.

ISO 29119-6 Engenharia de software e sistemas – Testes de software: Parte 6: Diretrizes para o uso da ISO/IEC/IEEE 29119 (todas as partes) em projetos ágeis, 2021. Relatório técnico. Organização Internacional de Normalização.

Documentos ISTQB®

ISTQB-ATLAS, ISTQB® , 2023. *Testador Certificado – Liderança em Testes Ágeis em Escala: Syllabus*. Versão 1.0.

ISTQB-CTFL, ISTQB® , 2024. *Testador Certificado Nível Básico: Syllabus*. Versão 4.0.1.

ISTQB-TAE, ISTQB® , 2024. *Testador Certificado em Engenharia de Automação de Testes: Syllabus*. Versão

2.0. ISTQB-TAS, ISTQB® , 2024. *Testador Certificado em Estratégia de Automação de Testes: Syllabus*.

Versão 1.0.

Referências do Glossário

Referências para a terminologia utilizada neste documento:

- Agile Alliance: https://agilealliance.org/agile101/agile-glossary
- IEEE/Pascal para engenharia de software: https://pascal.computer.org/sev_display/index.action
- IREB-CPRE para Engenharia de Requisitos: https://cpre.ireb.org/en/downloads-and-resources/glossário
- Glossário ISTQB®: https://glossary.istqb.org/

Livros

ADZIC, Gojko, 2011. *Specification by Example: How Successful Teams Deliver the Right Software*. Manning.

ANDERSON, D.J.; REINERSTEN, D.G., 2010. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.

ANDERSON, L.W.; KRATHWOHL, D.R., 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman. isbn 9780801319037.

BECK, K.; ANDRES, C., 2004. *Extreme Programming Explained: Embrace Change, 2nd Ed.* AddisonWesley.

BEIZER, Boris, 1990. *Software Testing Techniques*. Van Nostrand Reinhold: Boston MA.

GREAVES, K.; LAING, S., 2019. *Growing Agile: A Coach's Guide to Agile Testing*. Leanpub.

GREGORY, J.; CRISPIN, L., 2019. *Agile Testing Condensed: A Brief Introduction*. Leanpub.

WHITTAKER, J.A., 2009. *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*. Addison-Wesley Professional.

Artigos

ALJEDAANI, Wajdi; PERUMA, Anthony; ALJOHANI, Ahmed; ALOTAIBI, Mazen; MKAOUER, Mohamed Wiem; OUNI, Ali; NEWMAN, Christian D.; GHALLAB, Abdullatif; LUDI, Stephanie, 2021. Test Smell Detection Tools: A Systematic Mapping Study, pp. 170–180. Available from doi: 10.1145/3463274.3463335.

GHAZI, Ahmad Nauman; GARIGAPATI, Ratna Pranathi; PETERSEN, Kai, 2017. Checklists to Support Test Charter Design in Exploratory Testing, pp. 251–258. isbn 978-3-319-57633-6.

NIELSEN, Jakob, 1994. Enhancing the explanatory power of usability heuristics. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 152–158.

WALKER, R.; CENYDD, L. ap; POP, S.; MILES, H. C.; HUGHES, C. J.; TEAHAN, W. J.; ROBERTS, J. C., 2013. Storyboarding for visual analytics. *Information Visualization*. Vol. 14, pp. 27–50.

Páginas da Web

Agile Alliance, [n.d.]. Available also from: https://agilealliance.org/agile101/agile-glossary.

BECK, K.; BEEDLE, M.; BENNEKUM, A. van; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D., 2001. *Agile Manifesto* [online]. [visited on 2025-08-30]. Available from: https://agilemanifesto.org/.

BOLTON, M., 2012. *FEW HICCUPS* [online]. [visited on 2025-07-27]. Available from: https://developsense.com/blog/2012/07/few-hiccupps.

BOLTON, M., 2014. *How Models Change* [online]. [visited on 2025-07-27]. Available from: https://developsense.com/blog/2014/07/how-models-change.

GAREEV, A., 2019. *The TERMS for Test Automation* [online]. [visited on 2025-07-27]. Available from: https://www.automation-beyond.com/2019/04/10/the-terms-for-test-automation.

GREGORY, J., 2021. *Testing from a holistic point of view* [online]. [visited on 2025-09-05]. Available from: https://janetgregory.ca/testing-from-a-holistic-point-of-view/.

IEEE/Pascal for software engineering, [n.d.]. Available also from: https://pascal.computer.org/sev_display/index.action.

IREB-CPRE for Requirements Engineering, [n.d.]. Available also from: https://cpre.ireb.org/en/downloadsand-resources/glossary.

JOHNSON, K., 2012. *Software Testing Heuristics and Mnemonics* [online]. [visited on 2025-07-27]. Available from: https://www.slideshare.net/slideshow/kn- johnson- 2012- heuristics- mnemonics/15019733.

KOHL, J., 2010. *Test Mobile Applications with I SLICED UP FUN!* [online]. [visited on 2025-07-27]. Available from: https://www.kohl.ca/articles/ISLICEDUPFUN.pdf.

OWASP, 2025. *Top Ten Web Application Security Risks* [online]. 2025-07-27. [visited on 2025-07-27]. Available from: https://owasp.org/www-project-top-ten.

PROSCI, 2026. *The Prosci ADKAR Model* [online]. 2026-01-04. [visited on 2026-01-04]. Available from: https://www.prosci.com/methodology/adkar.

SUTHERLAND, J.; SCHWABER, K., 2020. *Scrum Guide* [online]. [visited on 2025-08-30]. Available from: https://scrumguides.org/index.html.

WYNNE, M., 2015. *Introdução ao Mapeamento de Exemplo* [online]. [consultado em 07/10/2025]. Disponibilidade: https://cucumber.io/blog/bdd/example-mapping-introduction.

As referências anteriores apontam para informações disponíveis na Internet e em outros locais. Embora essas referências tenham sido verificadas no momento da publicação deste syllabus, o ISTQB®não pode ser responsabilizado caso a disponibilidade das referências tenha diminuído.

8 Leitura complementar

GREGORY, J.; CRISPIN, L., 2024. *Holistic Testing: Weave Quality into Your Product*. Leanpub.

HENDRICKSON, Elisabeth, 2013. *Explore It!* Pragmatic Bookshelf.

ISO 29119-1 Software and systems engineering – Software testing: Part 1: General concepts, 2022. Standard. International Organization for Standardization.

ISO/IEC 20246: Software and systems engineering – Work product reviews, 2017. Standard. International Organization for Standardization.

ISO/IEC 25010: Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model, 2023. Standard. International Organization for Standardization.

ISO/IEC 25019: Systems and software Quality Requirements and Evaluation (SQuaRE) – Quality-in-use model, 2023. Standard. International Organization for Standardization.

ISO/IEC/IEEE 29119-3: Software testing, Part 3: Test documentation, 2021. Standard. International Organization for Standardization.

ISO/IEC/IEEE 29119-4: Software testing, Part 4: Test techniques, 2021. Standard. International Organization for Standardization.

ISO/IEC/IEEE 29119-5: Software testing, Part 5: Keyword-driven testing, 2016. Standard. International Organization for Standardization.

ISTQB-AI, ISTQB®, 2021. *Certified Tester AI Testing: Syllabus*. Version 1.0.

ISTQB-ITP, ISTQB®, 2011. *Certified Tester Expert Level Improving the Test Process: Syllabus*. Version 1.0.

ISTQB-MBT, ISTQB®, 2024. *Certified Tester Model-Based Tester: Syllabus*. Version 1.1.

ISTQB-PT, ISTQB®, 2018. *Certified Tester Performance Testing: Syllabus*. Version 1.0.

ISTQB-TM, ISTQB®, 2024. *Certified Tester Advanced Level Test Management: Syllabus*. Version 3.0.

ISTQB-TTA, ISTQB®, 2021. *Certified Tester Advanced Level Technical Test Analyst: Syllabus*. Version 4.0.

ISTQB-UT, ISTQB®, 2018. *Certified Tester Usability Testing: Syllabus*. Version 1.0.

9 Apêndice A – Lista de Abreviações

Abreviação	Definição
IA	Inteligência Artificial
API	Application Programming Interface
ATDD	Desenvolvimento Orientado por Teste de Aceite
BDD	Desenvolvimento Orientado pelo Comportamento
BPMN	Modelo e Notação de Processos de Negócios
CI/CD	Integração Contínua/Entrega Contínua
GQM	Meta-Pergunta-Métrica
LLM	Large Language Model
MTTR	Tempo Médio de Recuperação
OWASP	Projeto Aberto de Segurança de Aplicações Web
TDD	Desenvolvimento Orientado por Teste
IU	Interface do Usuário
XP	Programação Extrema

10 Apêndice B – Termos específicos do domínio

Nome do termo	Definição
viés	Um desvio sistemático da objetividade que pode afetar os objetivos do teste, as atividades ou os resultados do teste.
lançamento canário	Estratégia de implantação na qual uma nova versão de um sistema é lançada para um subconjunto limitado de usuários para avaliação antes de uma implementação mais ampla para todos os usuários.
comunidade de prática	Um grupo de profissionais que compartilham um interesse, paixão ou desafio comum, se reúnem regularmente para compartilhar conhecimento, aprender com as experiências uns dos outros e aprimorar habilidades para resolver problemas e inovar em seu domínio específico.
lançamento oculto	Técnica na qual uma funcionalidade nova ou alterada é implantada no ambiente de produção sem ficar visível ou com acessibilidade limitada aos usuários finais, a fim de validar o comportamento, a performance ou os riscos em condições reais antes do lançamento completo.
Definição de Feito	Uma descrição formal do estado do incremento quando este atende aos requisitos de qualidade exigidos para o produto.
épico	Uma descrição de uma necessidade dos stakeholders que é tipicamente maior do que o que pode ser implementado em uma única iteração.
mapeamento de exemplo	Uma técnica colaborativa no desenvolvimento ágil de software que utiliza regras, exemplos e perguntas para esclarecer requisitos e apoiar a modelagem de teste.
ativador de recurso	Um mecanismo para modificar o comportamento do sistema sem alterar o código.
FEW HICCUPPS	Um teste heurístico para Familiarity, Explainability, World, History, Image, Comparable products, Claims, Users' desires, Product, Purpose, Statutes, para apoiar a identificação de oráculos de teste.
Iteração de endurecimento	Uma iteração, normalmente realizada no final de uma série de iterações de desenvolvimento, na qual o foco principal é estabilizar o produto, corrigindo defeitos pendentes, melhorando o desempenho, concluindo atividades de integração e testes de sistema e preparando o produto para lançamento.
I SLICED UP FUN	Um teste heurístico para Inputs, Store, Location, Interactions/Interruptions, Communication, Ergonomics, Data, Usability, Platform, Function, User scenarios, Network.
iteração	Um ciclo de duração fixa e com prazo definido, no qual um conjunto de atividades planejadas é executado para produzir uma versão potencialmente lançável de um produto.
métrica	Uma escala de medição e método utilizado para medição.
RCRCRC	Um teste heurístico para Recent, Core, Risky, Configuration, Repaired, Chronic.
brainstorming de riscos	Uma técnica estruturada de brainstorming na qual os stakeholders identificam colaborativamente riscos potenciais, a fim de apoiar testes baseados em riscos.
SFDIPOT	Um teste heurístico para Structure, Function, Data, Interfaces, Platform, Operations, Time.
especificação por exemplo	Uma técnica de desenvolvimento na qual a especificação é definida por exemplos.
storyboarding	Uma representação visual de uma sequência de interações do usuário ou comportamentos do sistema, usada para esclarecer requisitos, apoiar a modelagem de teste e comunicar cenários de forma eficaz.

conversa estruturada	Uma discussão facilitada e direcionada que segue uma estrutura definida ou um conjunto de perguntas para garantir um entendimento comum, foco nos objetivos e troca eficaz de informações entre os stakeholders.
TERMS	Um teste heurístico para Tools and Technology, Execution, Requirements and Risks, Maintenance, Security.
registro de sessão de teste	Um documento utilizado em testes exploratórios para registrar a execução e os resultados de uma sessão de teste com duração definida, incluindo notas sobre as atividades de teste realizadas, observações, cobertura de teste alcançada, defeitos encontrados e itens de acompanhamento.
jornada do usuário	Uma representação da sequência de etapas que um usuário realiza para atingir um objetivo específico ao interagir com um sistema, incluindo a experiência do usuário, suas percepções e respostas emocionais em cada etapa.
história do usuário	Uma breve narrativa que descreve uma necessidade da perspectiva do usuário, juntamente com o benefício esperado quando essa necessidade for atendida.
divisão de histórias do usuário	A prática de dividir uma história do usuário grande ou complexa em histórias do usuário menores, gerenciáveis e com testabilidade.

11 Apêndice C – Objetivos de aprendizagem/Nível cognitivo de conhecimento

Os objetivos de aprendizagem específicos aplicáveis a este syllabus são apresentados no início de cada capítulo. Cada tópico do syllabus será analisado de acordo com o respectivo objetivo de aprendizagem.

Os objetivos de aprendizagem começam com um verbo de ação correspondente ao seu nível cognitivo de conhecimento, conforme listado abaixo.

Nível 1: Lembrar (K1)

O candidato irá lembrar, reconhecer e recordar um termo ou conceito.

Verbos de ação: Recordar, reconhecer.

Exemplos
Recordar os conceitos da pirâmide de teste.
Reconhecer os objetivos típicos dos testes.

Nível 2: Compreender (K2)

O candidato é capaz de selecionar as razões ou explicações para afirmações relacionadas ao tema e pode resumir, comparar, classificar e dar exemplos para o conceito de teste.

Verbos de ação: Classificar, comparar, diferenciar, distinguir, explicar, dar exemplos, interpretar, resumir

Exemplos	Notas
Classifique as ferramentas de teste de acordo com sua finalidade e as atividades de teste que elas suportam.	
Comparar os diferentes níveis de teste.	Pode ser usado para procurar semelhanças, diferenças ou ambos.
Diferenciar teste de depuração.	Procure diferenças entre conceitos.
Distinguir entre riscos de projeto e riscos de produto.	Permite que dois (ou mais) conceitos sejam classificados separadamente.
Explique o impacto do contexto no processo de teste.	
Dê exemplos de por que o teste é necessário.	
Inferir a causa-raiz dos defeitos a partir de um determinado perfil de falhas.	
Resuma as atividades do processo de revisão do produto de trabalho.	

Nível 3: Aplicar (K3)

O candidato é capaz de executar um procedimento quando confrontado com uma tarefa familiar, ou selecionar o procedimento correto e aplicá-lo a um determinado contexto.

Verbos de ação: Aplicar, implementar, preparar, usar

Exemplos	Notas
Aplicar a análise de valor limite para derivar casos de teste a partir de requisitos dados.	Deve consultar um procedimento/técnica/processo etc.

Implementar métodos de coleta de métricas para atender aos requisitos técnicos e de gestão.	
Prepare testes de instalabilidade para aplicativos móveis.	
Use a rastreabilidade para monitorar o progresso do teste quanto à integridade e consistência com os objetivos do teste, a estratégia de teste e o plano de teste.	Pode ser usado em um LO que deseja que o candidato seja capaz de usar uma técnica ou procedimento. Semelhante a "aplicar".

Nível 4: Analisar (K4)

O candidato é capaz de separar as informações relacionadas a um procedimento ou técnica em suas partes constituintes para melhor compreensão e pode distinguir entre fatos e inferências. A aplicação típica é analisar um documento, software ou situação de projeto e propor ações apropriadas para resolver um problema ou tarefa.

Verbos de ação: analisar, desconstruir, esboçar, priorizar, selecionar.

Exemplos	Notas
Analisar uma determinada situação de projeto para determinar quais técnicas de teste caixa-preta ou baseadas na experiência devem ser aplicadas para atingir objetivos específicos.	Avaliável apenas em combinação com um objetivo de medição da análise. Deve ter a forma "Analisar xxxx para xxxx" (ou similar).
Priorizar casos de teste em uma determinada suíte de teste para execução com base nos riscos de produto.	
Selecionar os níveis de teste e tipos de teste apropriados para verificar um determinado conjunto de requisitos.	Necessário quando a seleção requer análise como requisito.

Nível 5: Avaliar (K5)

O candidato pode fazer julgamentos com base em critérios e padrões. Ele detecta inconsistências ou falácias dentro de um processo ou produto, determina se um processo ou produto tem consistência interna e detecta a efetividade de um procedimento à medida que este está sendo implementado (por exemplo, determinar se as conclusões de um cientista decorrem dos dados observados).

Verbos de ação: Avaliar, criticar, avaliar, recomendar

Exemplos	Notas
Avaliar uma organização de testes usando o TPI Next ou o TMMi.	
Criticar a adequação das atividades de teste em uma organização para o contexto específico.	
Avaliar uma organização para determinar as opções de posicionamento adequado da equipe de testes.	Deve se restringir à avaliação, não incluindo a solução resultante (que seria K6)
Recomendar medidas para promover o aceite das mudanças pelas pessoas envolvidas.	

Nível 6: Criar (K6)

O candidato reúne elementos para formar um todo coerente ou funcional. A aplicação típica é reorganizar elementos em um novo padrão ou estrutura, elaborar um procedimento para realizar alguma tarefa ou inventar um produto (por exemplo, construir habitats para um propósito específico).

Verbos de ação: Criar, projetar, desenvolver, planejar

Exemplos	Notas
Criar um plano de melhoria de testes considerando questões de gestão de mudanças com etapas e ações apropriadas.	
Projete uma estrutura organizacional para um determinado escopo de um programa de melhoria de processo de teste	
Desenvolver um processo de gerenciamento de defeitos para uma organização de testes, incluindo o fluxo de trabalho de relatórios de defeitos e a comunicação	
Planeje e realize entrevistas de avaliação utilizando um modelo específico baseado em processo ou conteúdo.	

Os níveis cognitivos dos objetivos de aprendizagem baseiam-se em (L. Anderson et al., 2001).

12 Apêndice D – Matriz de rastreabilidade dos Resultados de Negócio com Objetivos de Aprendizagem

Esta seção lista a rastreabilidade entre os Resultados de Negócios e os Objetivos de Aprendizagem do Nível Avançado – Testador Ágil.

Resultados de Negócios: Testador Ágil de Nível Avançado			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT_BO1	Colaborar em equipes multifuncionais, estando familiarizado com os princípios e práticas básicas do desenvolvimento ágil de software		6							
CTAL-AT_BO2	Adaptar a experiência e o conhecimento em testes existentes aos valores e princípios ágeis			8						
CTAL-AT_BO3	Apoiar a equipe Ágil no planejamento de teste				6					
CTAL-AT_BO4	Aplicar abordagens relevantes de desenvolvimento ágil de software e técnicas de teste para garantir que os testes ofereçam cobertura adequada					10				
CTAL-AT_BO5	Auxiliar as stakeholders da empresa na definição de histórias do usuário, cenários, requisitos e critérios de aceite compreensíveis e com testabilidade, conforme apropriado						6			
CTAL-AT_BO6	Criar e implementar várias abordagens de teste para o desenvolvimento ágil de software							10		
CTAL-AT_BO7	Apoiar e contribuir para a automação de testes em um projeto de desenvolvimento ágil de software								2	

Resultados de Negócios: Testador Ágil de Nível Avançado			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT_BO8	Trabalhar com — e compartilhar informações com — outros membros da equipe utilizando estilos e canais de comunicação eficazes									9
LO exclusivo	Objetivo de Aprendizagem	Nível K								
1	Desafios da estratégia de teste e abordagem de teste - 60 minutos									
1.1	Tipos de teste									
CTAL-AT-1.1.1	Comparar os tipos de teste a serem realizados durante e após uma iteração	K2		X	X					
1.2	Teste de ponta-a-ponta	K2								
CTAL-AT-1.2.1	Explique quando o teste de ponta-a-ponta deve ser realizado	K2		X	X					
1.3	Testes formais e testes holísticos									
CTAL-AT-1.3.1	Compare as vantagens e desvantagens dos testes formais e dos testes holísticos	K2		X						
1.4	Abordagens de testes de regressão									
CTAL-AT-1.4.1	Diferenciar entre abordagens de teste de regressão	K2		X	X			X		

Resultados de Negócios: Testador Ágil de Nível Avançado			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
2	Pessoas e Equipes - 60 minutos									
2.1	Abordagem de Equipe Inteira									
CTAL-AT-2.1.1	Comparar generalização e especialização dentro de uma equipe	K2	X							X
CTAL-AT-2.1.2	Dê exemplos de como motivar representantes de negócio a realizar atividades de teste	K2	X							X
CTAL-AT-2.1.3	Resuma como a abordagem de equipe inteira pode auxiliar a equipe de desenvolvimento	K2	X							X
2.2	Testadores de tecidos									
CTAL-AT-2.2.1	Explique como e quando usar testadores de tecido	K2				X				
3	Gerenciamento de teste e melhoria de processo de teste - 210 minutos									
3.1	Planejamento de Teste									
CTAL-AT-3.1.1	Resumir como realizar o planejamento de teste no desenvolvimento ágil de software	K2	X	X	X					
CTAL-AT-3.1.2	Descrever uma estratégia de teste de projeto utilizando quadrantes de teste	K4		X				X		
3.2	Monitoramento de teste e controle de teste									

Resultados de Negócios: Testador Ágil de Nível Avançado			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT-3.2.1	Explique como realizar o monitoramento de teste e o controle de teste no desenvolvimento ágil de software	K3			X					
3.3	Relatórios de testes									
CTAL-AT-3.3.1	Compare os diferentes tipos de cobertura que podem ser usados para relatórios de teste no desenvolvimento ágil de software	K2		X		X				
3.4	Melhoria de Processo de Teste									
CTAL-AT-3.4.1	Selecionar medidas adequadas de melhoria de processo de teste com base em métricas do desenvolvimento ágil de software	K4	X	X						X
CTAL-AT-3.4.2	Explique como realizar a melhoria de processo de teste no desenvolvimento ágil de software	K2	X		X					X
4	Shift Left - 135 minutos									
4.1	Usando o shift left para melhorar a qualidade da base de teste									
CTAL-AT-4.1.1	Dê exemplos de como o testware pode ser usado como uma forma de requisitos	K2					X			X
CTAL-AT-4.1.2	Explique como o storyboarding e o testboarding podem ser usados para aumentar a qualidade da base de teste	K2					X			X
CTAL-AT-4.1.3	Explique como o mapeamento de exemplo pode ser usado para aumentar a qualidade da base de teste	K2					X			X

Resultados de Negócios: Testador Ágil de Nível Avançado			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT-4.1.4	Dê exemplos de como os vieses podem afetar negativamente a qualidade do produto	K2					X			
CTAL-AT-4.1.5	Aplicar a divisão de histórias do usuário para obter histórias do usuário com testabilidade	K3					X			
4.2	Shift Left e Engenharia de Requisitos									
CTAL-AT-4.2.1	Explique como a engenharia de requisitos apoia o shift left	K2					X			
5	Abordagens Ágeis e Técnicas de Teste - 285 minutos									
5.1	Teste exploratório									
CTAL-AT-5.1.1	Explicar testes heurísticos	K2				X		X		
CTAL-AT-5.1.2	Dê exemplos de testes mnemônicos relacionados aos testes no desenvolvimento ágil de software	K2				X		X		
CTAL-AT-5.1.3	Explique o que são test tours	K2				X		X		
CTAL-AT-5.1.4	Análise histórias do usuário e épicos para criar cartas de teste	K4								X
CTAL-AT-5.1.5	Aplicar testes exploratórios para apoiar os testes no desenvolvimento ágil de software	K3				X		X		
5.2	Testes Assistidos									

Resultados de Negócios: Testador Ágil de Nível Avançado			BO1	BO2	BO3	BO4	BO5	BO6	BO7	BO8
CTAL-AT-5.2.1	Explique o que é o mob testing	K2				X		X		
CTAL-AT-5.2.2	Explique o teste em pares	K2				X		X		
CTAL-AT-5.2.3	Explique o vibe testing	K2				X		X		
5.3	Indicadores de Anomalias (test smells)									
CTAL-AT-5.3.1	Use indicadores de anomalias para avaliar a qualidade dos casos de teste	K3				X		X		
6	Automação de testes e ferramentas de teste - 30 minutos									
6.1	Automação de testes no desenvolvimento ágil de software									
CTAL-AT-6.1.1	Distinguir entre diferentes abordagens de automação de testes aplicáveis ao desenvolvimento ágil de software	K2							X	
6.2	Ferramentas de teste no desenvolvimento ágil de software									
CTAL-AT-6.2.1	Dê exemplos de ferramentas de teste úteis em testes ágeis	K2							X	

13 Apêndice E – Notas de lançamento

Esta é uma versão principal do syllabus, portanto, não são fornecidas alterações detalhadas em relação à versão anterior. Em vez disso, como a diferença entre a v1.0 e a v2.0 é significativa, a natureza geral e o motivo das alterações são descritos abaixo.

Como o Agile é agora um método padrão de desenvolvimento ágil de software, os fundamentos do Agile não são mais descritos nos objetivos de aprendizagem. Em vez disso, esses fundamentos são abordados no Capítulo 1: Introdução ao Agile. Parte do conteúdo do syllabus do testador Agile v1.0 foi transferida para o syllabus do Nível Básico v4.0. Portanto, esse conteúdo não é abordado no syllabus atual do Agile Tester v2.0. Isso se aplica à abordagem de testar primeiro, pirâmide de teste, quadrantes de teste Ágeis, testes independentes, criação colaborativa de histórias do usuário, retrospectivas e estimativa do esforço de teste. Parte desse conteúdo é aprofundada neste syllabus (por exemplo, quadrantes de teste Ágeis no nível K4).

Novos tópicos foram introduzidos (como mnemônicos, heurísticas, test tours, vieses, métricas de cobertura para relatórios, testes holísticos, testadores de tecido, testes em grupo, testes de percepção, storyboarding e mapeamento de exemplo), e parte do conteúdo discutido no syllabus do Nível Básico 4.0 foi aprofundado (por exemplo, testes exploratórios e a abordagem de equipe completa).

14 Marcas registradas

ISTQB®é uma marca registrada do International Software Testing Qualifications Board. TMMi®é uma marca registrada da TMMi Foundation. TPI Next®é uma marca registrada da Sogeti, Países Baixos.