Certified Tester Advanced Level

Technical Test Analyst Syllabus

CTAL-TTA

versão 4.0

International Software Testing Qualifications Board

BSTQB

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

Direitos autorais

Copyright Notice© International Software Testing Qualifications Board (doravante denominado ISTQB®). ISTQB® é uma marca registrada da International Software Testing Qualifications Board.

Copyright© 2021, os autores para a atualização 2021 Adam Roman, Armin Born, Christian Graf, Stuart Reid

Copyright© 2019, os autores da atualização de 2019 Graham Bath (vice-presidente), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (presidente), Erik van Veenendaal.

Todos os direitos reservados. Os autores transferem os direitos autorais para o ISTQB®. Os autores (como atuais detentores dos direitos autorais) e o ISTQB® (como futuro detentor dos direitos autorais) concordaram com as seguintes condições de uso:

Extratos, para uso não-comercial, deste documento podem ser copiados se a fonte for reconhecida. Qualquer Provedor de Treinamento Credenciado pode usar este syllabus como base para um curso de treinamento se os autores e o ISTQB® forem reconhecidos como a fonte e os direitos autorais do syllabus e desde que qualquer anúncio de tal curso de treinamento possa mencionar o syllabus somente após o Credenciamento oficial do material de treinamento ter sido recebido de um Conselho Membro reconhecido pelo ISTQB®.

Qualquer indivíduo ou grupo de indivíduos pode usar este syllabus como base para artigos e livros, se os autores e o ISTQB® forem reconhecidos como a fonte e os proprietários dos direitos autorais do syllabus.

Qualquer outro uso deste syllabus é proibido sem antes obter a aprovação por escrito do ISTQB®.

Qualquer Conselho Membro reconhecido pelo ISTQB® pode traduzir este syllabus desde que reproduza o Aviso de Direitos Autorais acima mencionado na versão traduzida do syllabus.

©International Software Testing Qualifications Board

Histórico da Revisão

Versão	Data	Observações
v4.0	30/06/2021	Lançamento no GA para a versão v4.0
V4.0	28/04/2021	Rascunho atualizado com base no feedback da Beta Review.
2021 v4.0 Beta	01/03/2021	Rascunho atualizado com base no feedback da Alpha Review.
2021 v4.0 Alfa	07/12/2020	Rascunho para Revisão Alpha atualizado para: • Melhorar o texto todo • Remover a subseção associada ao K3 TTA-2.6.1 (2.6 Basis Path Testing) e remover o LO • Remover a subseção associada a K2 TTA-3.2.4 (3.2.4 Gráficos de Chamada) e remover LO • Reescrever a subseção associada ao TTA-3.2.2 (3.2.2 Análise do fluxo de dados) e torná-lo um K3 • Rewrite section associated with TTA-4.4.1 and TTA-4.4.2 (4.4. Teste de Confiabilidade) • Reescrever o capítulo associado ao TTA-4.5.1 e TTA-4.5.2 (4.5 Teste de performance) • Adicionar o capítulo 4.9 sobre Perfis Operacionais. • Reescrever o capítulo associado ao TTA-2.8.1 (2.7 Seleção do capítulo de Técnicas de Teste de White-Box) • Reescrever o TTA-3.2.1 para incluir a complexidade ciclomática (sem impacto nos Qs de exame) Reescrever TTA-2.4.1 (MC/DC) para torná-lo consistente com outros LOs de caixa-branca (sem impacto nos Qs de exame)
2019 v1.0	18/10/2019	Lançamento da GA para a versão 2019
2012	19/10/2012	Lançamento da GA para a versão 2012

Histórico da versão BSTQB

Versão	Data	Observações
1	14/06/2023	Padronização do layout com o ISTQB

©International Software Testing Qualifications Board

Índice

©International Software Testing Qualifications Board

©International Software Testing Qualifications Board

©International Software Testing Qualifications Board

Agradecimentos

A versão de 2019 deste documento foi produzida pela equipe *International Software Testing Qualifications Board Advanced Level Working Group*: Graham Bath (vice-presidente), Rex Black, Judy McKay, Kenji Onoshi, Mike Smith (presidente), Erik van Veenendaal

As seguintes pessoas participaram na revisão, comentário e votação da versão de 2019 deste syllabus:

Dani Almog	Andrew Archer	Rex Black
Armin Born	Sudeep Chatterjee	Tibor Csöndes
Wim Decoutere	Klaudia Dusser-Zieger	Melinda Eckrich-Brájer
Peter Foldhazi	David Frei	Karol Frühauf
Jan Giesen	Attila Gyuri	Matthias Hamburg
Tamás Horváth	N. Khimanand	Jan te Kock
Attila Kovács	Claire Lohr	Rik Marselis
Marton Matyas	Judy McKay	Dénes Medzihradszky
Petr Neugebauer	Ingvar Nordström	Pálma Polyák
Meile Posthuma	Stuart Reid	Lloyd Roden
Adam Roman	Jan Sabak	Péter Sótér
Benjamin Timmermans	Stephanie van Dijck	Paul Weymouth

Este documento foi produzido pela equipe *International Software Testing Qualifications Board Advanced Level Working Group*: Armin Born, Adam Roman, Stuart Reid.

A versão atualizada v4.0 deste documento foi produzida pela equipe *International Software Testing Qualifications Board Advanced Level Working Group*: Armin Born, Adam Roman, Christian Graf, Stuart Reid.

As seguintes pessoas participaram da revisão, comentário e votação da versão v4.0 atualizada deste syllabus:

Adél Vécsey-Juhász	Jane Nash	Pálma Polyák
Ágota Horváth	Lloyd Roden	Paul Weymounth
Benjamin Timmermans	Matthias Hamburg	Péter Földházi Jr.
Erwin Engelsma	Meile Posthuma	Rik Marselis
Gary Mogyorodi	Nishan Portoyan	Sebastian Małyska
Geng Chen	Joan Killeen	Tal Pe'er
Gergely Ágnecz	Ole Chr. Hansen	Wang Lijuan
		Zuo Zhenlei

A equipe de central agradece à equipe de revisão e aos Conselhos Nacionais por suas sugestões e contribuições.

Este documento foi formalmente divulgado pela Assembleia Geral do *ISTQB®* em 30 de junho de 2021.

©International Software Testing Qualifications Board

0 Introdução

0.1 Objetivo deste Syllabus

Este syllabus forma a base para a Qualificação Internacional de Teste de Software no Nível Avançado para o Analista Técnico de Teste. O ISTQB® fornece este syllabus como se segue:

1. Aos **Conselhos Nacionais**, para traduzir para seu idioma local e para credenciar os provedores de treinamento. Os Conselhos Nacionais podem adaptar o syllabus às suas necessidades linguísticas particulares e modificar as referências para se adaptarem às publicações locais.
2. Para os **Provedores de Exame**, derivar perguntas de exame em sua língua local adaptadas aos objetivos de aprendizagem do syllabus.
3. Para os **Provedores de Treinamento**, para produzir material didático e determinar os métodos de ensino apropriados.
4. Aos **Candidatos à Certificação**, para preparar-se para o exame (como parte de um curso de treinamento ou independentemente).
5. À **Comunidade Internacional** de Engenharia de Software e Sistema, para promover a profissão de teste de software e sistemas, e como base para livros e artigos.

O ISTQB® pode permitir que outras entidades utilizem este syllabus para outros fins, desde que busquem e obtenham permissão prévia por escrito.

0.2 A Certificação de Nível Avançado em Testes de Software

A qualificação de Nível Avançado é composta de três syllabus separados relacionados com as seguintes funções:

- Test Manager
- Test Analyst
- Technical Test Analyst

O ISTQB® Technical Test Analyst Advanced Level Overview é um documento separado [CTAL_TTA_OVIEW] que inclui as seguintes informações:

- Resultados de negócio
- Matriz de rastreabilidade entre os resultados de negócio e os objetivos de aprendizado

0.3 Objetivos de Aprendizagem e Níveis Cognitivos

Os Objetivos de Aprendizagem apoiam os Resultados de Negócio e são usados para criar o exame para alcançar a Certificação avançada Technical Test Analyst.

Os níveis de conhecimento dos objetivos específicos de aprendizado nos níveis K2, K3 e K4 são mostrados no início de cada capítulo e são classificados da seguinte forma:

- **K2**: Entender

©International Software Testing Qualifications Board

- **K3**: Aplicar
- **K4**: Analisar

0.4 Expectativas de experiência

Alguns dos objetivos de aprendizado para o Analista Técnico de Teste assumem que a experiência básica está disponível nas seguintes áreas:

- Conceitos gerais de programação
- Conceitos gerais de arquiteturas de sistemas

0.5 O Exame

O exame *Advanced Level Technical Test Analyst* será baseado neste syllabus. As respostas às perguntas do exame podem exigir o uso de materiais baseados em mais de um capítulo deste syllabus. Todos os capítulos são examináveis, exceto a introdução e os apêndices. Normas, livros e outros syllabi do ISTQB® são incluídos como referências, mas seu conteúdo não é examinável além do que está resumido neste syllabus.

O formato do exame é de múltipla escolha com 45 perguntas. Para passar no exame, pelo menos 65% do total de pontos deve ser obtido.

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

0.6 Requisitos de entrada para o exame

A certificação *Certified Tester Foundation Level* deve ser obtida antes de se fazer o exame de certificação *Advanced Level Technical Test Analyst*.

0.7 Credenciamento de cursos

Um Conselho Membro do ISTQB® pode credenciar provedores de treinamento cujo material didático siga este syllabus. Os prestadores de treinamento devem obter diretrizes de credenciamento do Conselho de Membros ou do órgão que realiza o credenciamento. Um curso credenciado é reconhecido como estando em conformidade com este syllabus e é permitido ter um exame ISTQB® como parte do curso.

0.8 Nível de detalhe do Syllabus

O nível de detalhes nesse syllabus permite cursos e exames internacionalmente consistentes. Para atingir este objetivo, o syllabus consiste em:

- Objetivos gerais de instruções descrevendo a intenção do Analista Técnico de Teste;
- Uma lista de termos que os estudantes devem ser capazes de lembrar;

©International Software Testing Qualifications Board

- Objetivos de aprendizagem para cada área de conhecimento, descrevendo o resultado da aprendizagem cognitiva a ser alcançado;
- Uma descrição dos conceitos-chave, incluindo referências a fontes como literatura ou normas aceitas.

O conteúdo do syllabus não é uma descrição de toda a área de conhecimento; ele reflete o nível de detalhe a ser coberto nos cursos de treinamento de Nível Avançado. Ele se concentra no material que pode ser aplicado a qualquer projeto de software, utilizando qualquer ciclo de vida de desenvolvimento de software. O syllabus não contém nenhum objetivo de aprendizado específico relacionado a nenhum modelo de desenvolvimento de software em particular, mas discute como estes conceitos se aplicam no desenvolvimento de software Ágil, em outros tipos de modelos de desenvolvimento de software iterativos e incrementais, e em modelos sequenciais de desenvolvimento de software.

0.9 Como este Syllabus está organizado

Há seis capítulos com conteúdo examinável. O cabeçalho de nível superior para cada capítulo especifica o tempo mínimo para o capítulo; o tempo não é fornecido para subcapítulos. Para cursos de treinamento credenciados, o syllabus exige um mínimo de **20 horas de instrução**, distribuídas pelos seis capítulos da seguinte forma:

- Capítulo 1: As tarefas do Analista Técnico de Teste em testes baseados em risco (30 minutos)
- Capítulo 2: Técnicas de Teste de White-Box (300 minutos)
- Capítulo 3: Análise estática e dinâmica (180 minutos)
- Capítulo 4: Características de qualidade para testes técnicos (345 minutos)
- Capítulo 5: Revisões (165 minutos)
- Capítulo 6: Ferramentas de teste e automação (180 minutos)

©International Software Testing Qualifications Board

1 Tarefas do Analista Técnico de Teste em testes baseados em risco [30 min]

Palavras-chave

avaliação do risco, identificação do risco, mitigação do risco, risco do produto, risco do projeto, testes baseados no risco.

Objetivos de aprendizagem

1.2 Tarefas de testes baseados em risco

TTA-1.2.1 (K2) Resumir os fatores genéricos de risco que o Analista Técnico de Teste normalmente precisa considerar.

TTA-1.2.2 (K2) Resumir as atividades do Analista Técnico de Testes dentro de uma abordagem baseada em riscos para atividades de testes.

©International Software Testing Qualifications Board

1.1 Introdução

O Gerente de Teste tem a responsabilidade geral de estabelecer e gerenciar uma estratégia de testes baseada em riscos. O Gerente de Teste normalmente solicitará o envolvimento do Analista Técnico de Teste para garantir que a abordagem baseada em riscos seja implementada corretamente.

Os Analistas Técnicos de Teste trabalham dentro da estrutura de teste baseado em risco estabelecida pelo Gerente de Teste para o projeto. Eles contribuem com seu conhecimento dos riscos técnicos do produto que são inerentes ao projeto, tais como os riscos relacionados à segurança, confiabilidade do sistema e performance. Eles também devem contribuir para a identificação e tratamento dos riscos do projeto associados aos ambientes de teste, tais como a aquisição e configuração de ambientes de teste para testes de performance, confiabilidade e segurança.

1.2 Tarefas de Testes Baseados em Risco

Os Analistas Técnicos de Teste estão ativamente envolvidos nas seguintes tarefas de testes baseados em riscos:

- Identificação de riscos.
- Avaliação de risco.
- Mitigação de riscos.

Estas tarefas são executadas iterativamente durante todo o projeto para lidar com riscos emergentes e mudanças de prioridades, e para avaliar e comunicar regularmente a situação de risco.

1.2.1 Identificação de riscos

Ao recorrer a amostra mais ampla possível de stakeholders, é mais provável que o processo de identificação de riscos detecte o maior número possível de riscos significativos. Como os Analistas Técnicos de Teste possuem habilidades técnicas únicas, eles são particularmente adequados para conduzir entrevistas com especialistas, trocar ideias com colegas de trabalho e analisar suas experiências para determinar onde se encontram as prováveis áreas de risco do produto. Em particular, os Analistas Técnicos de Teste trabalham em estreita colaboração com outros stakeholders, tais como desenvolvedores, arquitetos, engenheiros de operações, proprietários de produtos, escritórios de suporte local, especialistas técnicos e técnicos de service desk, para determinar as áreas de risco técnico que impactam o produto e o projeto. O envolvimento de outros stakeholders garante que todas as opiniões sejam consideradas, e são normalmente facilitadas pelos Gerentes de Testes.

Os riscos que podem ser identificados pelo Analista Técnico de Teste são normalmente baseados nas características de qualidade do produto [ISO 25010] listadas no Capítulo 4 deste syllabus.

1.2.2 Avaliação de risco

Enquanto a identificação de riscos se refere à identificação do maior número possível de riscos pertinentes, a avaliação de riscos é o estudo para categorizar e determinar a probabilidade e o impacto associados a cada risco identificado.

©International Software Testing Qualifications Board

A probabilidade de risco de um produto é geralmente interpretada como a probabilidade da ocorrência da falha no sistema em teste. O Analista Técnico de Teste contribui para a compreensão da probabilidade de risco técnico do produto, enquanto o Analista de Teste contribui para a compreensão do potencial impacto no negócio caso o problema ocorra.

Os riscos do projeto que se tornam problemas podem ter impacto no sucesso geral do projeto. Tipicamente, os seguintes fatores genéricos de risco do projeto precisam ser considerados:

- Conflitos entre os stakeholders quanto aos requisitos técnicos;
- Problemas de comunicação resultantes da distribuição geográfica da organização de desenvolvimento;
- Ferramentas e tecnologia (incluindo habilidades relevantes);
- Tempo, recursos e pressão gerencial;
- Falta de garantia de qualidade anterior;
- Altas taxas de mudança nos requisitos técnicos.

Os riscos do produto que se tornam problemas podem resultar em maior número de defeitos. Normalmente, os seguintes fatores genéricos de risco do produto precisam ser considerados:

- Complexidade da tecnologia;
- Complexidade do código;
- Quantidade de mudanças no código fonte (inserções, exclusões, modificações);
- Grande número de defeitos encontrados relacionados às características técnicas de qualidade (histórico de defeitos);
- Interface técnica e questões de integração.

Dadas as informações disponíveis de risco, o Analista Técnico de Teste propõe uma probabilidade inicial de risco de acordo com as diretrizes estabelecidas pelo Gerente de Teste. O valor inicial pode ser modificado pelo Gerente de Teste quando todas as opiniões dos stakeholders tiverem sido consideradas. O impacto do risco é normalmente determinado pelo Analista de Testes.

1.2.3 Mitigação de riscos

Durante o projeto, os Analistas Técnicos de Teste influenciam como os testes respondem aos riscos identificados. Isto geralmente envolve o seguinte:

- Projetar casos de teste para aqueles riscos que tratam de áreas de alto risco e ajudando a avaliar o risco residual
- Reduzir o risco executando os casos de teste projetados e colocando em ação medidas apropriadas de mitigação e contingência, como indicado no plano de teste
- Avaliar os riscos com base em informações adicionais coletadas à medida que o projeto se desdobra, e usar essas informações para implementar medidas de mitigação destinadas a diminuir a probabilidade desses riscos

O Analista Técnico de Teste frequentemente cooperará com especialistas em áreas como segurança e performance para definir medidas de mitigação de risco e elementos da estratégia de teste. Informações adicionais podem ser obtidas nos syllabi de especializações do ISTQB®, como o *Security Testing* [CT_SEC_SYL] e o *Performance Testing* [CT_PT_SYL].

©International Software Testing Qualifications Board

2 Técnicas de Teste Caixa-Branca [300 min]

Palavras-chave

condição atômica, fluxo de controle, nível de integridade de segurança, técnica de teste caixa-branca, teste API, teste de condição múltipla, teste de condição/decisão modificada, teste de decisão, teste de instrução.

Objetivos de aprendizagem

2.2 Testes de instrução

TTA-2.2.1 (K3) Conceber casos de teste para um determinado objeto de teste, aplicando testes de instrução para alcançar um nível definido de cobertura.

2.3 Teste de decisão

TTA-2.3.1 (K3) Conceber casos de teste para um determinado objeto de teste, aplicando a técnica de teste de decisão para alcançar um nível definido de cobertura.

2.4 Teste de Condição/Decisão Modificada

TTA-2.4.1 (K3) Conceber casos de teste para um determinado objeto de teste aplicando a técnica de teste de condição/decisão modificada para alcançar cobertura total de condição/decisão modificada (MC/DC).

2.5 Teste de condições múltiplas

TTA-2.5.1 (K3) Conceber casos de teste para um determinado objeto de teste aplicando a técnica de teste de condições múltiplas para alcançar um nível definido de cobertura.

2.6 Teste do caminho de base (foi removido da versão v4.0 deste syllabus)

TTA-2.6.1 removido da versão v4.0 deste syllabus.

2.7 Testes API

TTA-2.7.1 (K2) Entender a aplicabilidade dos testes API e os tipos de defeitos encontrados.

2.8 Selecionando uma técnica de teste caixa-branca

TTA-2.8.1 (K4) Selecionar uma técnica de teste caixa-branca apropriada, de acordo com uma determinada situação de projeto.

©International Software Testing Qualifications Board

2.1 Introdução

Este capítulo descreve as técnicas de teste caixa-branca. Estas técnicas se aplicam a códigos e outras estruturas com um fluxo de controle, tais como gráficos de fluxo de processos de negócio.

Cada técnica específica permite que os casos de teste sejam derivados sistematicamente e se concentrem em um aspecto particular da estrutura. Os casos de teste gerados por estas técnicas satisfazem critérios de cobertura que são estabelecidos como um objetivo e são medidos em relação a eles. Atingir a cobertura total (ou seja, 100%) não significa que todo o conjunto de testes esteja completo, mas sim que a técnica utilizada não sugere mais nenhum teste adicional útil para a estrutura em consideração.

As entradas de teste são geradas para garantir que um caso de teste exerça uma determinada parte do código (p. ex., uma instrução ou resultado de decisão). Determinar as entradas de teste que farão com que uma determinada parte do código seja exercida pode ser um desafio, especialmente se a parte do código a ser exercida estiver no final de um longo subcaminho de fluxo de controle com várias decisões sobre ele. Os resultados esperados são identificados com base em uma fonte externa a esta estrutura, como uma exigência ou especificação de projeto ou outra base de teste.

As seguintes técnicas são consideradas neste syllabus:

- Teste de instrução
- Testes de decisão
- Testes de condição/decisão modificados
- Teste de condições múltiplas
- Testes API

O Foundation Syllabus [CTFL_SYL] introduz os testes de instrução e testes de decisão. O teste de instrução concentra-se no exercício das instruções executáveis no código, enquanto o teste de decisão nos resultados da decisão.

A condição/decisão modificada e as técnicas de condições múltiplas listadas acima são baseadas em previsões de decisão contendo condições múltiplas e encontram tipos similares de defeitos. Não importa quão complexo possa ser um predicado de uma decisão, ele será avaliado como VERDADEIRO ou FALSO, o que determinará o caminho percorrido pelo código. Um defeito é detectado quando o caminho pretendido não é seguido porque um predicado de decisão não é avaliado como esperado.

Consulte [ISO 29119] para mais detalhes sobre a especificação, e exemplos, destas técnicas.

2.2 Testes de Instrução

O teste da instrução exercita as instruções executáveis no código. A cobertura é medida como o número de instruções executadas pelos testes dividido pelo número total de instruções executáveis no objeto de teste, normalmente expresso como uma porcentagem.

©International Software Testing Qualifications Board

Aplicabilidade

A obtenção de uma cobertura total das instruções deve ser considerada como um mínimo para todos os códigos a serem testados, embora isso nem sempre seja possível na prática.

Limitações/Dificuldades

A obtenção da totalidade da cobertura de instrução deve ser considerada como um mínimo para todos os códigos a serem testados, embora isto nem sempre seja possível na prática devido às restrições de tempo e/ou esforço disponíveis. Mesmo as altas porcentagens de cobertura de instrução podem não detectar certos defeitos na lógica do código. Em muitos casos, não é possível atingir 100% de cobertura de instrução devido ao código ser inalcançável. Embora o código inalcançável geralmente não seja considerado uma boa prática de programação, ele pode ocorrer, por exemplo, se uma instrução de mudança deve ter um caso padrão, mas todos os casos possíveis são tratados explicitamente.

2.3 Testes de Decisão

Os testes de decisão exercem os resultados da decisão no código. Para isso, os casos de teste seguem os fluxos de controle de um ponto de decisão (p. ex., para uma instrução IF, há um fluxo de controle para o resultado verdadeiro e um para o resultado falso; para uma instrução CASE, pode haver vários resultados possíveis; para uma instrução LOOP, há um fluxo de controle para o resultado verdadeiro da condição do loop e um para o resultado falso).

A cobertura é medida como o número de resultados de decisão exercidos pelos testes dividido pelo número total de resultados de decisão no objeto de teste, normalmente expresso como uma porcentagem. Observe que um único caso de teste pode resultar em vários resultados de decisão.

Em comparação com a condição/decisão modificada e as técnicas de condições múltiplas descritas abaixo, os testes de decisão consideram a decisão como um todo e avaliam apenas os resultados VERDADEIRO e FALSO, independentemente da complexidade de sua estrutura interna.

Os testes de decisão são frequentemente utilizados de forma intercambiável com os testes de decisão, pois os mesmos testes podem cobrir todos os desvios e todos os resultados de decisão. Os testes de decisão testam os desvios do código, onde um desvio é normalmente considerado como uma fronteira do gráfico do fluxo de controle. Para programas onde não há decisões em seu código, a definição de cobertura de decisão resulta em uma cobertura de 0/0, que é indefinida, não importando quantos testes sejam executados; enquanto um único desvio de um ponto de entrada até um ponto de saída (assumindo um ponto de entrada e saída) resultará em uma cobertura de 100% dos desvios. Para resolver esta diferença entre as duas medidas, a ISO 29119-4 exige que pelo menos um teste seja executado em um código sem decisões para atingir 100% de cobertura de decisão, de modo que as coberturas de decisão e desvio sejam 100% equivalentes para quase todos os programas. Muitas ferramentas de teste que fornecem medidas de cobertura, incluindo as utilizadas para testar sistemas relacionados à segurança, empregam uma abordagem semelhante.

Aplicabilidade

Este nível de cobertura deve ser considerado quando o código a ser testado for importante ou mesmo crítico (ver as tabelas no capítulo 2.8.2 para sistemas relacionados com a segurança). Esta

©International Software Testing Qualifications Board

técnica pode ser usada para código e para qualquer modelo que envolva pontos de decisão, como modelos de processos de negócio.

Limitações/Dificuldades

Os testes de decisão não consideram os detalhes de como uma decisão com múltiplas condições é tomada e pode falhar na detecção de defeitos causados por combinações dos resultados das condições.

2.4 Teste de Condição/Decisão Modificada

Em comparação com os testes de decisão, que consideram a decisão como um todo e avaliam os resultados VERDADEIROS e FALSOS, os testes de condição/decisão modificados consideram como uma decisão é tomada quando inclui múltiplas condições (quando uma decisão é composta de apenas uma condição atômica, é simplesmente um teste de decisão).

Cada predicado de decisão é composto por uma ou mais condições atômicas, cada uma das quais avalia a um valor booleano. Estas são logicamente combinadas para determinar o resultado da decisão. Esta técnica verifica se cada uma das condições atômicas afeta de forma independente e correta o resultado da decisão global.

Esta técnica proporciona um nível de cobertura mais forte do que a cobertura de declarações e decisões quando há decisões contendo múltiplas condições. Assumindo N condições atômicas únicas e mutuamente independentes, a cobertura de condição/decisão modificada (MC/DC) para uma decisão pode normalmente ser alcançado exercitando a decisão N+1 vezes. O teste de condição/decisão modificada requer pares de testes que mostram que uma mudança de um único resultado de condição atômica pode afetar independentemente o resultado de uma decisão. Observe que um único caso de teste pode exercitar várias combinações de condições e, portanto, nem sempre é necessário executar N+1 casos de teste separados para alcançar MC/DC.

Aplicabilidade

Essa técnica é utilizada nas indústrias aeroespacial e automotiva, e em outros setores industriais para sistemas críticos de segurança. É utilizada quando se testa um software onde uma falha pode causar uma catástrofe. O teste de condição/decisão modificada pode ser um meio-termo razoável entre o teste de decisão e o teste de condições múltiplas (devido ao grande número de combinações a serem testadas). É mais rigoroso do que o teste de decisão, mas requer muito menos condições de teste a serem exercitadas do que testes de condições múltiplas quando há várias condições atômicas na decisão.

Limitações/Dificuldades

A execução de MC/DC pode ser complicada quando há múltiplas ocorrências da mesma variável em uma decisão com múltiplas condições; quando isso ocorre, as condições podem ser "acopladas". Dependendo da decisão, pode não ser possível variar o valor de uma condição de tal forma que só isso faça com que o resultado da decisão mude. Uma abordagem para tratar desta questão é especificar que somente condições atômicas desacopladas são testadas usando testes de condição/decisão modificados. A outra abordagem é analisar cada decisão em que ocorre o acoplamento.

©International Software Testing Qualifications Board

Alguns compiladores e/ou interpretadores são projetados de forma que exibam comportamento de curto-circuito ao avaliar uma instrução de decisão complexa no código. Ou seja, o código de execução pode não avaliar uma expressão inteira se o resultado da avaliação puder ser determinado após avaliar apenas uma parte da expressão. Por exemplo, se avaliar a decisão "A e B", não há razão para avaliar B se A já tiver sido avaliado como FALSO. Nenhum valor de B pode alterar o resultado, portanto o código pode economizar tempo de execução ao não avaliar B. O curto-circuito pode afetar a capacidade de atingir a MC/DC, uma vez que alguns testes requeridos podem não ser executados. Normalmente, é possível configurar o compilador para desligar a otimização do curto-circuito para os testes, mas isto pode não ser permitido para aplicações críticas de segurança, onde o código testado e o código entregue devem ser idênticos.

2.5 Teste de Condições Múltiplas

Em raros casos, pode ser necessário testar todas as combinações possíveis de condições atômicas que uma decisão pode conter. Esse nível de teste é chamado de teste de condições múltiplas. Assumindo N condições atômicas únicas e mutuamente independentes, a cobertura total de condições múltiplas para uma decisão pode ser alcançada exercitando-a 2N vezes. Note que um único caso de teste pode exercitar várias combinações de condições e, portanto, nem sempre é necessário executar casos de teste 2N separados para atingir 100% de cobertura de condições múltiplas.

A cobertura é medida como o número de combinações de condições atômicas executadas sobre todas as decisões no objeto de teste, normalmente expresso como uma porcentagem.

Aplicabilidade

Esta técnica é usada para testar software de alto risco e software incorporado que se espera que funcionem de forma confiável sem travamentos por longos períodos.

Limitações/Dificuldades

Como o número de casos de teste pode ser derivado diretamente de uma tabela verdade contendo todas as condições atômicas, este nível de cobertura pode ser facilmente determinado. Entretanto, o número absoluto de casos de teste necessários para testes de condições múltiplas torna o teste de condição/decisão modificada mais viável para situações em que existem várias condições atômicas em uma decisão.

Se o compilador usar curto-circuito, o número de combinações de condições que podem ser exercitadas muitas vezes será reduzido, dependendo da ordem e agrupamento de operações lógicas que são realizadas nas condições atômicas.

2.6 Teste do Caminho de Base

Este capítulo foi retirado da versão v4.0 deste syllabus.

©International Software Testing Qualifications Board

2.7 Testes de API

Uma interface de programação de aplicativos (API) é uma interface definida que permite a um programa chamar outro sistema de software, o que lhe fornece um serviço, como o acesso a um recurso remoto. Os serviços típicos incluem serviços web, serviço de ônibus executivo, bancos de dados, mainframes e UIs da Web.

O teste de API é um tipo de teste em vez de uma técnica. Em certos aspectos, o teste de API é bastante semelhante ao teste de uma interface gráfica de usuário (GUI). O foco é a avaliação dos valores de entrada e dos dados retornados.

Os testes negativos são frequentemente cruciais quando se trata de APIs. Programadores que usam APIs para acessar serviços externos a seu próprio código podem tentar usar interfaces API de maneiras para as quais não foram destinados. Isso significa que o manuseio robusto de erros é essencial para evitar uma operação incorreta. Testes combinatórios de muitas interfaces podem ser necessários porque as APIs são frequentemente usadas em conjunto com outras APIs, e porque uma única interface pode conter vários parâmetros, onde os valores destes podem ser combinados de muitas maneiras.

As APIs são frequentemente pouco acopladas, resultando na possibilidade muito real de perda de transações ou falhas de tempo. Isso requer testes minuciosos dos mecanismos de recuperação e de tentativas de recuperação. Uma organização que fornece uma interface API deve assegurar que todos os serviços tenham uma disponibilidade muito alta; isso geralmente requer testes rigorosos de confiabilidade pelo editor da API, bem como suporte de infraestrutura.

Aplicabilidade

Os testes de API estão se tornando mais importantes para testar sistemas de sistemas à medida que os sistemas individuais se tornam distribuídos ou usam processamento remoto como uma forma de descarregar algum trabalho para outros processadores. Exemplos incluem:

- Chamadas de sistemas operacionais;
- Arquiteturas orientadas a serviços (SOA);
- Chamadas de procedimento remoto (RPC);
- Serviços Web.

O software encapsulado resulta na divisão de um programa de software em vários contêineres que se comunicam uns com os outros usando mecanismos como os listados acima. Os testes de API também devem visar essas interfaces.

Limitações/Dificuldades

Testar uma API diretamente geralmente requer um Analista Técnico de Teste para utilizar ferramentas especializadas. Como normalmente não há uma interface gráfica direta associada a uma API, podem ser necessárias ferramentas para configurar o ambiente inicial, ordenar os dados, invocar a API e determinar o resultado.

Cobertura

O teste de API é uma descrição de um tipo de teste; não indica nenhum nível específico de cobertura. No mínimo, o teste de API deve incluir a realização de chamadas à API com valores de entrada

©International Software Testing Qualifications Board

realistas e entradas inesperadas para verificação do tratamento de exceções. Testes de API mais completos podem garantir que entidades que chamam sejam exercidas pelo menos uma vez, ou que todas as funções possíveis sejam chamadas pelo menos uma vez.

A *Representational State Transfer* (RESTful) é um estilo arquitetônico. Os serviços RESTful Web permitem que os sistemas solicitem acesso aos recursos da web usando um conjunto uniforme de operações sem estado. Existem vários critérios de cobertura para as APIs RESTful web, o padrão de fato, para integração de software [Web-7]. Eles podem ser divididos em dois grupos: critérios de cobertura de entrada e critérios de cobertura de saída. Entre outros, os critérios de entrada podem exigir a execução de todas as possíveis operações de API, o uso de todos os parâmetros de API possíveis, e a cobertura de sequências de operações de API. Entre outros, os critérios de saída podem exigir a geração de todos os códigos de status corretos e errôneos, e a geração de respostas contendo recursos que exibam todas as propriedades (ou todos os tipos de propriedades).

Tipos de Defeitos

Os tipos de defeitos que podem ser encontrados testando APIs são bastante díspares. Problemas de interface são comuns, assim como problemas de manuseio de dados, de tempo, perda de transações, duplicação de transações e problemas no manuseio de exceções.

2.8 Seleção da técnica de Teste Caixa-Branca

A técnica de teste caixa-branca selecionada é normalmente especificada em termos de um nível de cobertura específico, o que é alcançado através da aplicação da técnica de teste. Por exemplo, uma exigência para atingir 100% de cobertura de instrução normalmente levaria ao uso do teste de instrução. Entretanto, as técnicas de teste caixa-preta são normalmente aplicadas primeiro, a cobertura é então medida, e a técnica de teste caixa-branca só é usada se o nível de cobertura específico do teste caixa-branca não tiver sido alcançado. Em algumas situações, o teste caixa-branca pode ser usado menos formalmente para fornecer uma indicação de onde a cobertura pode precisar ser aumentada (p. ex., criando testes adicionais onde os níveis de cobertura de teste caixa-branca são particularmente baixos). Os testes de instrução seriam normalmente suficientes para essa medição informal da cobertura.

Ao especificar um nível de cobertura de teste caixa-branca necessário, é uma boa prática especificá-lo apenas a 100%. A razão para isto é que se forem necessários níveis de cobertura mais baixos, então isto normalmente significa que as partes do código que não são testadas são as partes mais difíceis de se testar, e estas partes são normalmente, também, as mais complexas e propensas a erros. Assim, ao solicitar e alcançar, por exemplo, uma cobertura de 80%, pode significar que o código que inclui a maioria dos defeitos detectáveis necessita ser testada. Por este motivo, quando os critérios de cobertura de teste caixa-branca são especificados em normas, eles são quase sempre especificados a 100%. Definições estritas dos níveis de cobertura tornaram às vezes este nível de cobertura impraticável. Entretanto, aquelas dadas na ISO 29119-4 permitem que itens de cobertura inviáveis sejam descontados dos cálculos, tornando assim a cobertura de 100% uma meta alcançável.

Ao especificar a cobertura de teste caixa-branca exigida para um objeto de teste, é necessário especificar um único critério de cobertura (p. ex., não é necessário exigir tanto uma cobertura de instrução 100%, como 100% MC/DC). Com critérios de saída a 100% é possível relacionar alguns critérios de saída em uma hierarquia de subconjuntos, onde os critérios de cobertura são mostrados

©International Software Testing Qualifications Board

para incluir outros critérios de cobertura em um contexto mais amplo. Diz-se que um critério de cobertura é integrado a outro se, para todos os componentes e suas especificações, cada conjunto de casos de teste que satisfaz o primeiro critério também satisfaz o segundo. Por exemplo, a cobertura de decisão integra a cobertura de instrução porque se a cobertura de decisão for alcançada (até 100%), então a cobertura de instrução é garantida a 100%. Para as técnicas de teste caixa branca cobertas neste syllabus, podemos dizer que a cobertura de decisão e de decisão integram a cobertura de instrução, a cobertura de decisão e de desvio MC/DC, e a cobertura de múltiplas condições integram MC/DC (se considerarmos a cobertura de decisão e de decisão como sendo a mesma a 100%, então podemos dizer que elas se integram uma à outra).

Observe que ao determinar os níveis de cobertura de teste caixa-branca a serem alcançados para um sistema, é bastante normal definir níveis diferentes para diferentes partes do sistema. Isto ocorre porque diferentes partes de um sistema contribuem de forma diferente para o risco. Por exemplo, em um sistema aviônico, aos subsistemas associados ao entretenimento de voo seria atribuído um nível de risco menor do que aqueles associados ao controle de voo. As interfaces de teste são comuns para todos os tipos de sistemas e normalmente são necessárias para todos os níveis de integridade de sistemas relacionados à segurança (ver capítulo 2.8.2 para mais informações sobre níveis de integridade). O nível de cobertura exigido para testes de API normalmente aumentará com base no risco associado (p. ex., o nível mais alto de risco associado a uma interface pública pode exigir testes de API mais rigorosos).

A seleção da técnica de teste caixa-branca a ser utilizada é geralmente baseada na natureza do objeto de teste e em seus riscos levantados. Se o objeto de teste for considerado como sendo de segurança (ou seja, uma falha pode causar danos às pessoas ou ao meio ambiente), então as normas regulamentares são aplicáveis e definirão os níveis de cobertura de teste caixa-branca necessários (ver capítulo 2.8.2). Se o objeto de teste não estiver relacionado à segurança, então a escolha dos níveis de cobertura de teste caixa-branca a serem atingidos é mais subjetiva, mas ainda assim deve ser amplamente baseada nos riscos levantados, como descrito no capítulo 2.8.1.

2.8.1 Sistemas não relacionados à segurança

Os seguintes fatores (nenhuma ordem de prioridade em particular) são normalmente considerados ao selecionar técnicas de teste caixa-branca para sistemas não relacionados à segurança:

- **Contrato**: Se o contrato exigir um determinado nível de cobertura a ser alcançado, e este não for alcançado o resultado será potencialmente uma violação do contrato.
- **Cliente**: Se o cliente solicitar um determinado nível de cobertura, por exemplo, como parte do planejamento do teste, e este não for alcançado haverá problemas com o cliente.
- **Norma regulatória**: Para alguns setores industriais (p. ex., financeiro), aplica-se uma norma regulatória que define os critérios necessários de cobertura de teste caixa-branca para sistemas de missão crítica. Ver capítulo 2.8.2 para cobertura de normas regulatórias para sistemas relacionados com a segurança.
- **Estratégia de teste**: Se a estratégia de teste da organização especificar os requisitos para a cobertura do código caixa-branca, o não alinhamento com a estratégia da organização pode arriscar a censura da alta administração.

©International Software Testing Qualifications Board

- **Estilo de codificação**: Se o código for escrito sem condições múltiplas dentro das decisões, então seria um desperdício exigir níveis de cobertura de teste caixa-branca, tais como MC/DC e cobertura de condições múltiplas.
- **Informação histórica de defeitos**: Se os dados históricos sobre a eficácia de alcançar um determinado nível de cobertura sugerir o que seria apropriado usar para este objeto de teste, seria arriscado ignorar os dados disponíveis. Note que tais dados podem estar disponíveis dentro do projeto, organização ou indústria.
- **Habilidades e experiência**: Se os testadores disponíveis para realizar os testes não forem suficientemente experientes e habilitados em uma determinada técnica caixa-branca, isso pode ser mal compreendido e pode introduzir riscos desnecessários se essa técnica for selecionada.
- **Ferramentas**: A cobertura de teste caixa-branca só pode ser medida na prática através do uso de ferramentas de cobertura. Se tais ferramentas não estiverem disponíveis e que suportem uma determinada medida de cobertura, então a seleção dessa medida a ser alcançada introduziria um alto nível de risco.

Ao selecionar o teste caixa-branca para sistemas não relacionados à segurança, o Analista Técnico de Testes tem mais liberdade para recomendar a cobertura apropriada caixa-branca para sistemas não relacionados à segurança do que para sistemas relacionados à segurança. Tais escolhas são tipicamente um compromisso entre os riscos percebidos e o custo, recursos e tempo necessários para tratar esses riscos através de testes caixa-branca. Em algumas situações, outros tratamentos, que podem ser implementados por outras abordagens de teste de software ou de outra forma (p. ex., diferentes abordagens de desenvolvimento), podem ser mais apropriados.

2.8.2 Sistemas relacionados à segurança

Quando o software que está sendo testado faz parte de um sistema relacionado à segurança, então um padrão regulatório que define os níveis de cobertura necessários a serem atingidos normalmente terá que ser usado. Tais padrões normalmente exigem uma análise de risco a ser realizada para o sistema e os riscos resultantes são usados para atribuir níveis de integridade a diferentes partes do sistema. Os níveis de cobertura exigida são definidos para cada um dos níveis de integridade.

A IEC 61508 (segurança funcional de sistemas programáveis, eletrônicos, relacionados à segurança [IEC 61508]) é uma norma internacional guarda-chuva usada para tais propósitos. Em teoria, ela poderia ser usada para qualquer sistema relacionado à segurança, porém algumas indústrias criaram variantes específicas (p. ex., a ISO 26262 [ISO 26262] se aplica a sistemas automotivos) e algumas indústrias criaram suas próprias normas (p. ex., DO-178C [DO178C] para software aerotransportado). Informações adicionais sobre a ISO 26262 são fornecidas no syllabus do ISTQB® *Automotive Software Tester* [CT_AuT_SYL].

A IEC 61508 define quatro níveis de integridade de segurança (SILs), cada um deles definido como um nível relativo de redução de risco proporcionado por uma função de segurança, correlacionado com a frequência e a gravidade dos perigos percebidos. Quando um objeto de teste executa uma função relacionada à segurança, quanto maior o risco de falha significa que o objeto de teste deve ter maior confiabilidade. A tabela a seguir mostra os níveis de confiabilidade associados aos SILs. Observe que o nível de confiabilidade da SIL 4 para o caso de operação contínua é extremamente alto, já que corresponde a um tempo médio entre falhas (MTBF) maior que 10.000 anos.

©International Software Testing Qualifications Board

IEC 6150 SIL	Operação Contínua (probabilidade de uma falha perigosa por hora)	Sob Demanda (probabilidade de falha sob demanda)
1	$\geq 10^{-6}$ para $<10^{-5}$	$\geq 10^{-2}$ para $<10^{-1}$
2	$\geq 10^{-7}$ para $<10^{-6}$	$\geq 10^{-3}$ para $<10^{-2}$
3	$\geq 10^{-8}$ para $<10^{-7}$	$\geq 10^{-4}$ para $<10^{-3}$
4	$\geq 10^{-9}$ para $<10^{-8}$	$\geq 10^{-5}$ para $<10^{-4}$

As recomendações para os níveis de cobertura de teste caixa-branca associados a cada SIL são mostradas na tabela a seguir. Quando uma entrada é mostrada como "Altamente recomendada", na prática, alcançar esse nível de cobertura é normalmente considerado obrigatório. Em contraste, quando uma entrada é mostrada apenas como 'Recomendada', muitos profissionais consideram isso como opcional e evitam alcançá-la fornecendo uma justificativa adequada. Assim, um objeto de teste atribuído à SIL 3 é normalmente testado para atingir 100% de cobertura de decisão (ele atinge 100% de cobertura de instrução automaticamente, como mostrado pelo pedido de subsumo).

IEC 61508 SIL	100% Cobertura de Instrução	100% Cobertura de Decisão	100% MC/DC
1	Recomendada	Recomendada	Recomendada
2	Altamente recomendada	Recomendada	Recomendada
3	Altamente recomendada	Altamente recomendada	Recomendada
4	Altamente recomendada	Altamente recomendada	Altamente recomendada

Observe que os SILs acima e os requisitos sobre níveis de cobertura da IEC 61508 são diferentes na ISO 26262, e diferentes na DO-178C.

©International Software Testing Qualifications Board

3 Análise Estática e Dinâmica [180 min]

Palavras-chave

análise de fluxo de controle, análise de fluxo de dados, análise dinâmica, análise estática, anomalia, complexidade ciclomática, par de definição-utilização, ponteiro selvagem, vazamento de memória.

Objetivos de aprendizagem

3.2 Análise Estática

TTA-3.2.1 (K3) Usar a análise de fluxo de controle para detectar se o código tem alguma anomalia de fluxo e para medir a complexidade ciclomática.

TTA-3.2.2 (K3) Usar análise de fluxo de dados para detectar se o código tem alguma anomalia de fluxo de dados.

TTA-3.2.3 (K3) Propor formas de melhorar a capacidade de manutenção do código através da aplicação de análise estática.

3.3 Análise Dinâmica

TTA-3.3.1 (K3) Aplicar análise dinâmica para atingir uma meta específica.

©International Software Testing Qualifications Board

3.1 Introdução

A análise estática (ver capítulo 3.2) é uma forma de teste que é realizada sem a execução do software. A qualidade do software é avaliada por uma ferramenta ou por uma pessoa com base em sua forma, estrutura, conteúdo ou documentação. Esta visão estática do software permite uma análise detalhada sem a necessidade de criar os dados e condições prévias necessárias para executar os casos de teste.

Além da análise estática, as técnicas de testes estáticos também incluem diferentes formas de revisão. As que são relevantes para o Analista Técnico de Teste são abordadas no Capítulo 5.

A análise dinâmica (ver capítulo 3.3) requer a execução real do código e é usada para encontrar defeitos que são mais facilmente detectados quando o código é executado (p. ex., vazamentos de memória). A análise dinâmica, assim como a análise estática, pode depender de ferramentas ou pode depender de uma pessoa que monitore o sistema de execução, observando tais ponteiros como um rápido aumento do uso de memória.

3.2 Análise Estática

O objetivo da análise estática é detectar defeitos reais ou potenciais no código e na arquitetura do sistema e melhorar sua capacidade de manutenção.

3.2.1 Análise do Fluxo de Controle

A análise do fluxo de controle é a técnica estática na qual as etapas seguidas por um programa são analisadas através do uso de um gráfico de fluxo de controle, geralmente com o uso de uma ferramenta. Há uma série de anomalias que podem ser encontradas em um sistema que utiliza esta técnica, incluindo loops que são mal projetados (p. ex., ter múltiplos pontos de entrada ou que não terminam), alvos ambíguos de chamadas de funções em certos idiomas, sequenciamento incorreto de operações, código que não pode ser alcançado, funções não chamadas etc.

A análise do fluxo de controle pode ser usada para determinar a complexidade ciclomática, que é um número inteiro positivo que representa o número de caminhos independentes em um gráfico fortemente conectado.

A complexidade ciclomática é geralmente usada como um indicador da complexidade de um componente. A teoria de Thomas McCabe [McCabe76] cita que quanto mais complexo é o sistema, mais difícil será mantê-lo e mais defeitos ele conterá. Muitos estudos notaram esta correlação entre a complexidade e o número de defeitos contidos. Qualquer componente que seja medido com maior complexidade deve ser revisto para uma possível refatoração, por exemplo, a divisão em múltiplos componentes.

3.2.2 Análise do Fluxo de Dados

A análise do fluxo de dados cobre uma variedade de técnicas que reúnem informações sobre o uso de variáveis em um sistema. O ciclo de vida de cada variável é investigado (ou seja, onde ela é

©International Software Testing Qualifications Board

declarada, definida, usada e destruída), uma vez que potenciais anomalias podem ser identificadas se essas ações forem usadas fora de sequência [Beizer90].

Uma técnica comum classifica o uso de uma variável como uma das três ações atômicas:

- quando a variável é definida, declarada ou inicializada (p. ex., x:=3).
- quando a variável é usada ou lida (p. ex., se x > temp).
- quando a variável é morta, destruída, ou sai do escopo (p. ex., text_file_1. close, variável de controle de loop (i) na saída do loop).

As sequências de tais ações que indicam potenciais anomalias incluem:

- definição seguida de outra definição ou matar sem uso interventivo.
- definição sem matar posteriormente (p. ex., levando a um possível vazamento de memória para variáveis alocadas dinamicamente).
- usar ou matar antes da definição.
- usar ou matar após a variável já ter sido morta.

Dependendo da linguagem de programação, algumas dessas anomalias podem ser identificadas pelo compilador, mas uma ferramenta de análise estática separada pode ser necessária para identificar as anomalias de fluxo de dados. Por exemplo, a redefinição sem intervenção é permitida na maioria das linguagens de programação e pode ser programada deliberadamente, mas seria sinalizada por uma ferramenta de análise de fluxo de dados como sendo uma possível anomalia que deveria ser verificada.

O uso de caminhos de fluxo de controle para determinar a sequência de ações de uma variável, pode levar ao relato de potenciais anomalias que não podem ocorrer na prática. Por exemplo, as ferramentas de análise estática nem sempre podem identificar se um caminho de fluxo de controle é viável, pois alguns caminhos são determinados apenas com base nos valores atribuídos às variáveis em tempo de execução. Há também uma classe de problemas de análise de fluxo de dados que são difíceis de identificar pelas ferramentas, quando os dados analisados fazem parte de estruturas de dados com variáveis dinamicamente atribuídas, tais como registros e matrizes. As ferramentas de análise estática também têm dificuldade em identificar potenciais anomalias de fluxo de dados quando as variáveis são compartilhadas entre linhas de controle simultâneas em um programa, pois a sequência de ações sobre os dados torna-se difícil de prever.

Ao contrário da análise de fluxo de dados, que é um teste estático, o teste de fluxo de dados é um teste dinâmico no qual são gerados casos de teste para exercitar pares de definição-utilização no código do programa. O teste de fluxo de dados usa alguns dos mesmos conceitos da análise de fluxo de dados, que estes pares de definição-utilização são caminhos de fluxo de controle entre uma definição e um uso subsequente de uma variável em um programa.

3.2.3 Usando a Análise Estática para melhorar a manutenibilidade

A análise estática pode ser aplicada de várias maneiras para melhorar manutenibilidade do código, da arquitetura e de websites.

Código mal escrito, não comentado e não estruturado tende a ser mais difícil de ser mantido. Pode exigir mais esforço dos desenvolvedores para localizar e analisar defeitos no código, e a modificação

©International Software Testing Qualifications Board

do código para corrigir um defeito ou adicionar uma característica provavelmente resultará na introdução de mais defeitos.

A análise estática é utilizada para verificar a conformidade com as normas e diretrizes de codificação; quando é identificado um código incompatível, ele pode ser atualizado para melhorar sua capacidade de manutenção. Estas normas e diretrizes descrevem as práticas de codificação e projeto necessárias, tais como convenções de nomenclatura, comentários, recuo e modularização. Observe que as ferramentas de análise estática geralmente levantam avisos ao invés de detectar defeitos. Essas advertências (p. ex., em nível de complexidade) podem ser fornecidas mesmo que o código esteja sintaticamente correto.

Os projetos modulares geralmente resultam em códigos mais fáceis de manter. As ferramentas de análise estática apoiam o desenvolvimento do código modular das seguintes maneiras:

- Eles pesquisam códigos repetidos. Essas seções de código podem ser candidatas a refatoração em componentes (embora a sobrecarga de tempo de execução imposta pelas chamadas a componentes possa ser um problema para sistemas em tempo real).
- Eles geram métricas que são valiosos indicadores de modularização de códigos. Estes incluem medidas de conexão e coesão. Um sistema que tenha boa capacidade de manutenção tem mais probabilidade de ter uma medida baixa de conexão (o grau em que os componentes dependem uns dos outros durante a execução) e uma medida alta de coesão (o grau em que um componente é autocontido e focado em uma única tarefa).
- Eles indicam, em código orientado a objetos, onde os objetos derivados podem ter muita ou pouca visibilidade para as classes pais.
- Eles destacam áreas em código ou arquitetura com um alto nível de complexidade estrutural.

A manutenção de um website também pode ser suportada usando ferramentas de análise estática. Aqui o objetivo é verificar se a estrutura em forma de árvore do site está bem balanceada ou se há um desequilíbrio que levará a:

- Tarefas de teste mais difíceis
- Aumento da carga de trabalho de manutenção

Além de avaliar a capacidade de manutenção, ferramentas de análise estática também podem ser aplicadas ao código usado para implementar websites para verificar a possível exposição a vulnerabilidades de segurança, tais como injeção de código, segurança de cookies, scripts entre sites, adulteração de recursos e injeção de código SQL. Mais detalhes são fornecidos no capítulo 4.3 e no syllabus *Security Testing* [CT_SEC_SYL].

3.3 Análise Dinâmica

3.3.1 Visão Geral

A análise dinâmica é utilizada para detectar falhas onde os sintomas só são visíveis quando o código é executado. Por exemplo, a possibilidade de vazamentos de memória pode ser detectada pela análise estática (encontrar código que aloca, mas nunca libera a memória), mas um vazamento de memória é facilmente aparente com a análise dinâmica.

©International Software Testing Qualifications Board

As falhas que não são imediatamente reproduzíveis (intermitentes) podem ter consequências significativas no esforço de teste e na capacidade de liberar ou usar produtivamente o software. Tais falhas podem ser causadas por vazamentos de memória ou de recursos, uso incorreto de indicadores e outras corrupções (p. ex., da pilha do sistema) [Kaner02]. Devido à natureza dessas falhas, que podem incluir a piora gradual da performance do sistema ou mesmo falhas do sistema, as estratégias de teste devem considerar os riscos associados a tais defeitos e, quando apropriado, realizar análises dinâmicas para reduzi-los (tipicamente usando ferramentas). Como essas são as falhas mais caras para encontrar e corrigir, é recomendável iniciar a análise dinâmica no início do projeto.

A análise dinâmica pode ser aplicada para realizar o seguinte:

- Impedir a ocorrência de falhas detectando vazamentos de memória (ver capítulo 3.3.2) e ponteiros perdidos (ver capítulo 3.3.3);
- Analisar falhas do sistema que não podem ser facilmente reproduzidas;
- Avaliar o comportamento da rede;
- Melhorar a performance do sistema usando perfis de código para fornecer informações sobre o comportamento do sistema em tempo de execução, que podem ser usadas para fazer mudanças informadas.

A análise dinâmica pode ser realizada em qualquer nível de teste e requer habilidades técnicas e de sistema para:

- Especificar os objetivos do teste de análise dinâmica;
- Determinar o tempo adequado para iniciar e interromper a análise;
- Analisar os resultados.

Ferramentas de análise dinâmica podem ser usadas mesmo que o Analista Técnico de Teste tenha habilidades técnicas mínimas; as ferramentas usadas geralmente criam registros abrangentes que podem ser analisados por aqueles com as habilidades técnicas e analíticas necessárias.

3.3.2 Detecção de vazamentos de memória

Um vazamento de memória ocorre quando áreas de memória (RAM) são alocadas a um programa, mas não são liberadas posteriormente quando não são mais necessárias. Esta área de memória não está disponível para reutilização. Quando isto ocorre frequentemente ou em situações de pouca memória, o programa pode ficar sem memória utilizável. Historicamente, a manipulação da memória era de responsabilidade do programador. Qualquer área de memória alocada dinamicamente tinha que ser liberada pelo programa de alocação para evitar um vazamento de memória. Muitos ambientes de programação modernos incluem "coletor de lixo" automático ou semiautomático onde a memória alocada é liberada após o uso sem a intervenção direta do programador. O isolamento de vazamentos de memória pode ser muito difícil nos casos em que a memória alocada deve ser liberada pelos coletores automáticos de lixo.

Os vazamentos de memória normalmente causam problemas após algum tempo - quando uma quantidade significativa de memória vazou e ficou indisponível. Quando o software é instalado recentemente, ou quando o sistema é reiniciado, a memória é realocada, e assim os vazamentos de memória não são perceptíveis; o teste é um exemplo onde a alocação, frequente de memória, pode

©International Software Testing Qualifications Board

impedir a detecção de vazamentos de memória. Por estas razões, os efeitos negativos de vazamentos de memória podem ser notados primeiro quando o programa está em produção.

O principal sintoma de um vazamento de memória é o tempo de resposta do sistema em constante deterioração, que pode resultar em falha do sistema. Embora tais falhas possam ser resolvidas reiniciando o sistema, isto pode nem sempre ser prático ou mesmo possível para alguns sistemas.

Muitas ferramentas de análise dinâmica identificam áreas no código onde ocorrem vazamentos de memória para que eles possam ser corrigidos. Monitores simples de memória também podem ser usados para obter uma impressão geral de que a memória disponível está diminuindo com o tempo, embora ainda seja necessária uma análise de acompanhamento para determinar a causa exata do declínio.

3.3.3 Detectando ponteiros perdidos

Ponteiros perdidos dentro de um programa são indicadores que não são mais precisos e não devem ser usados. Por exemplo, um ponteiro perdido pode ter "perdido" o objeto ou função para o qual deveria estar apontando ou não aponta para a área de memória pretendida (p. ex., aponta para uma área que está além dos limites alocados de uma matriz). Quando um programa usa ponteiros perdidos, uma variedade de consequências pode ocorrer, incluindo as seguintes:

- O programa pode ter a performance esperada. Este pode ser o caso em que o ponteiro perdido acessa a memória que atualmente não é utilizada pelo programa e está supostamente "livre" e/ou contém um valor razoável.
- O programa pode falhar. Neste caso, o ponteiro perdido pode ter causado o uso incorreto de uma parte da memória que é crítica para a execução do programa (p. ex., o sistema operacional).
- O programa não funciona corretamente porque os objetos requeridos pelo programa não podem ser acessados. Nestas condições, o programa pode continuar a funcionar, embora uma mensagem de erro possa ser emitida.
- Os dados no local da memória podem ser corrompidos pelo ponteiro e valores incorretos usados posteriormente (isto também pode representar uma ameaça à segurança).

Observe que mudanças no uso da memória do programa (p. ex., uma nova construção após uma mudança de software) podem desencadear qualquer uma das quatro consequências listadas acima. Isto é particularmente crítico onde inicialmente o programa funciona como esperado, apesar do uso de ponteiros perdido, e depois cai inesperadamente (talvez até mesmo na produção) após uma mudança de software. As ferramentas podem ajudar a identificar os ponteiros perdidos à medida que são usados pelo programa, independentemente de seu impacto na execução do programa. Alguns sistemas operacionais têm funções incorporadas para verificar violações de acesso à memória durante o tempo de execução. Por exemplo, o sistema operacional pode lançar uma exceção quando uma aplicação tenta acessar um local de memória que está fora da área de memória permitida dessa aplicação.

©International Software Testing Qualifications Board

3.3.4 Análise de eficiência de performance

A análise dinâmica não é útil apenas para detectar falhas e localizar os defeitos associados. Com a análise dinâmica da performance do programa, as ferramentas ajudam a identificar gargalos de eficiência da performance e geram uma ampla gama de métricas de performance que podem ser usadas pelo desenvolvedor para ajustar a performance do sistema. Por exemplo, podem ser fornecidas informações sobre o número de vezes que um componente é chamado durante a execução. Os componentes que são frequentemente chamados seriam provavelmente candidatos a um aumento de performance. Muitas vezes a regra de Pareto se aplica aqui: um programa gasta uma parte desproporcional (80%) de seu tempo de execução em um pequeno número (20%) de componentes [Andrist20].

A análise dinâmica da performance do programa geralmente é feita durante a realização de testes do sistema, embora também possa ser feita ao testar um único subsistema em fases anteriores de testes usando uma estrutura de teste. Mais detalhes são fornecidos no syllabus *Performance Testing* [CT_PT_SYL].

©International Software Testing Qualifications Board

4 Características de qualidade para testes técnicos [345 min]

Palavras-chave

responsabilidade, adaptabilidade, analisabilidade, autenticidade, disponibilidade, capacidade, coexistência, compatibilidade, confidencialidade, tolerância a falhas, instalabilidade, integridade, manutenção, maturidade, modificabilidade, modularidade, não repúdio, perfil operacional, eficiência de performance, portabilidade, característica de qualidade, recuperabilidade, confiabilidade, modelo de crescimento de confiabilidade, substituibilidade, utilização de recursos, reusabilidade, segurança, testabilidade, comportamento temporal

Objetivos de aprendizagem

4.2 Questões gerais de planejamento

TTA-4.2.1 (K4) Para um determinado cenário, analisar os requisitos não-funcionais e escrever as respectivas seções do plano de teste
TTA-4.2.2 (K3) Dado um risco particular do produto, definir o(s) tipo(s) de teste não-funcional(ais) mais apropriado(s)
TTA-4.2.3 (K2) Entender e explicar os estágios do ciclo de vida de desenvolvimento de software de uma aplicação onde os testes não-funcionais devem ser aplicados tipicamente
TTA-4.2.4 (K3) Para um determinado cenário, defina os tipos de defeitos que você esperaria encontrar utilizando os diferentes tipos de testes não-funcionais

4.3 Teste de Segurança

TTA-4.3.1 (K2) Explicar as razões para incluir testes de segurança em uma abordagem de teste
TTA-4.3.2 (K2) Explicar os principais aspectos a serem considerados no planejamento e especificação dos testes de segurança

4.4 Teste de Confiabilidade

TTA-4.4.1 (K2) Explicar as razões para incluir testes de confiabilidade em uma abordagem de teste
TTA-4.4.2 (K2) Explicar os principais aspectos a serem considerados no planejamento e especificação dos testes de confiabilidade

4.5 Teste de Performance

TTA-4.5.1 (K2) Explicar as razões para incluir testes de performance em uma abordagem de teste
TTA-4.5.2 (K2) Explicar os principais aspectos a serem considerados no planejamento e especificação de testes de performance

4.6 Teste de Manutenibilidade

TTA-4.6.1 (K2) Explicar as razões para incluir testes de manutenção em uma abordagem de teste

4.7 Teste de Portabilidade

TTA-4.7.1 (K2) Explicar as razões para incluir testes de portabilidade em uma abordagem de teste

4.8 Teste de Compatibilidade

TTA-4.8.1 (K2) Explicar as razões para incluir testes de coexistência em uma abordagem de teste

©International Software Testing Qualifications Board

4.1 Introdução

Em geral, o Analista Técnico de Teste foca os testes em "como" o produto funciona, em vez dos aspectos funcionais do "o que" ele faz. Estes testes podem ser realizados em qualquer nível de teste. Por exemplo, durante os testes de componentes de sistemas embutidos e em tempo real, é importante realizar benchmarking de eficiência de performance e uso de recursos de teste. Durante os testes de aceite operacional e testes de sistema, é apropriado testar os aspectos de confiabilidade, tais como a recuperabilidade. Os testes neste nível visam testar um sistema específico, ou seja, combinações de hardware e software. O sistema específico em teste pode incluir vários servidores, clientes, bancos de dados, redes e outros recursos. Independentemente do nível de teste, os testes devem ser realizados de acordo com as prioridades de risco e os recursos disponíveis.

Tanto testes dinâmicos quanto testes estáticos, incluindo revisões (ver capítulos 2, 3 e 5), podem ser aplicados para testar as características de qualidade não-funcionais descritas neste capítulo.

A descrição das características de qualidade do produto fornecida na ISO 25010 é usada como um guia para as características e suas subcaracterísticas. Estas são mostradas na tabela abaixo, juntamente com uma indicação de quais características/subcaracterísticas são cobertas pelos syllabi *Test Analyst* e *Technical Test Analyst*.

Características	Subcaracterísticas	TA	TTA
Adequação funcional	Correção funcional, adequação funcional, completude funcional	X	
Confiabilidade	Maturidade, tolerância a falhas, capacidade de recuperação, disponibilidade		X
Usabilidade	Reconhecimento de adequação, capacidade de aprendizagem, operabilidade, estética da interface do usuário, proteção de erros do usuário, acessibilidade	X	
Eficiência de performance	Comportamento do tempo, utilização de recursos, capacidade		X
Manutenibilidade	Analisabilidade, modificabilidade, testabilidade, modularidade, reusabilidade		X
Portabilidade	Adaptabilidade, instalabilidade, substituibilidade	X	X
Segurança	Confidencialidade, integridade, não rejeição, responsabilidade, autenticidade		X
Compatibilidade	Coexistência		X
	Interoperabilidade	X	

Observe que no Anexo A é fornecida uma tabela que compara as características descritas na agora cancelada norma ISO 9126-1 (como usada na versão 2012 deste syllabus) com as da última norma ISO 25010.

Para todas as características e subcaracterísticas de qualidade discutidas nesta seção, os riscos típicos devem ser reconhecidos para que uma abordagem de teste apropriada possa ser formada e documentada. Os testes de características de qualidade requerem atenção especial ao tempo de

©International Software Testing Qualifications Board

ciclo de vida, ferramentas necessárias, padrões exigidos, disponibilidade de software e documentação e experiência técnica. Sem o planejamento de uma abordagem para lidar com cada característica e suas necessidades únicas de teste, o testador pode não ter um planejamento adequado, preparação e tempo de execução de teste integrados no cronograma.

Alguns destes testes, por exemplo, testes de performance, requerem planejamento extensivo, equipamento dedicado, ferramentas específicas, habilidades de testes especializados e, na maioria dos casos, uma quantidade significativa de tempo. Os testes das características e subcaracterísticas de qualidade devem ser integrados ao cronograma geral de testes com recursos adequados alocados para o esforço.

Enquanto o Gerente de Teste se preocupará em compilar e relatar as informações resumidas da métrica relativa às características e subcaracterísticas de qualidade, o Analista de Testes ou o Analista Técnico de Testes (de acordo com a tabela acima) reunirá as informações para cada métrica.

Medidas de características de qualidade coletadas em testes de pré-produção pelo Analista Técnico de Teste podem formar a base para Acordos de Nível de Serviço (SLAs) entre o fornecedor e os stakeholders (p. ex., clientes, operadores) do sistema de software. Em alguns casos, os testes podem continuar a ser executados após o software ter entrado em produção, geralmente por uma equipe ou organização separada. Isto geralmente é observado nos para testes de performance e testes de confiabilidade, que podem mostrar diferentes resultados no ambiente de produção e no ambiente de testes.

4.2 Questões gerais de planejamento

A falha no planejamento de testes não funcionais pode colocar o sucesso de um projeto em risco considerável. O Analista Técnico de Teste pode ser solicitado pelo Gerente de Teste para identificar os principais riscos para as características relevantes de qualidade (ver tabela no capítulo 4.1) e abordar quaisquer questões de planejamento associadas aos testes propostos. Estas informações podem ser usadas na criação do plano de testes principal.

Os seguintes fatores gerais são considerados na execução dessas tarefas:

- Requisitos dos stakeholders
- Requisitos do ambiente de teste
- Aquisição e treinamento da ferramenta necessária
- Considerações organizacionais
- Considerações sobre a segurança dos dados.

4.2.1 Requisitos dos stakeholders

Os requisitos não funcionais são muitas vezes mal especificados ou mesmo inexistentes. Na fase de planejamento, os Analistas Técnicos de Teste devem ser capazes de obter níveis de expectativa relacionados às características técnicas de qualidade dos stakeholders afetadas e avaliar os riscos que estas representam.

Uma abordagem comum é assumir que, se o cliente estiver satisfeito com a versão existente do sistema, ele continuará satisfeito com novas versões, desde que os níveis de qualidade alcançados

©International Software Testing Qualifications Board

sejam mantidos. Isto permite que a versão existente do sistema possa ser usada como referência. Essa pode ser uma abordagem particularmente útil a ser adotada para algumas das características de qualidade não-funcionais, tais como eficiência de performance, onde os stakeholders podem ter dificuldade em especificar seus requisitos.

É aconselhável obter múltiplos pontos de vista ao capturar requisitos não-funcionais. Eles devem ser solicitados aos stakeholders, tais como clientes, proprietários de produtos, usuários, pessoal de operações e pessoal de manutenção. Se os principais stakeholders forem excluídos, é provável que alguns requisitos não sejam cumpridos. Para mais detalhes sobre a captura de requisitos, consulte o syllabus *Test Manager* [CTAL_TM_SYL].

No desenvolvimento ágil de software, requisitos não-funcionais podem ser declarados como histórias de usuários ou adicionados à funcionalidade especificada em casos de uso como restrições não-funcionais.

4.2.2 Requisitos do ambiente de teste

Muitos testes técnicos (p. ex., testes de segurança, testes de confiabilidade, testes de eficiência de performance) requerem um ambiente de teste semelhante ao da produção para fornecer medidas realistas. Dependendo do tamanho e complexidade do sistema em teste, isto pode ter um impacto significativo no planejamento e financiamento dos testes. Como o custo de tais ambientes pode ser alto, as seguintes opções podem ser consideradas:

- Usar o ambiente de produção
- Usar uma versão reduzida do sistema, cuidando para que os resultados dos testes obtidos sejam suficientemente representativos ao sistema de produção
- Usar recursos baseados na nuvem como alternativa para adquirir os recursos diretamente
- Usar ambientes virtualizados

O tempo de execução dos testes deve ser planejado cuidadosamente, e é bem provável que alguns testes só possam ser executados em momentos específicos (p. ex., em tempos de baixa utilização).

4.2.3 Aquisição e treinamento necessário de ferramentas

As ferramentas fazem parte do ambiente de teste. As ferramentas comerciais ou simuladores são particularmente relevantes para a eficiência da performance e certos testes de segurança. Os Analistas Técnicos de Teste devem estimar os custos e prazos envolvidos para aquisição, aprendizado e implementação das ferramentas. Onde ferramentas especializadas devem ser usadas, o planejamento deve levar em conta as curvas de aprendizado para novas ferramentas e o custo de contratação de especialistas externos.

O desenvolvimento de um simulador complexo pode representar um projeto de desenvolvimento por direito próprio e deve ser planejado como tal. Em particular, os testes e a documentação da ferramenta desenvolvida devem ser contabilizados no cronograma e no plano de recursos. Orçamento e tempo suficientes devem ser planejados para atualizar e testar novamente o simulador à medida que o produto simulado muda. O planejamento dos simuladores a serem usados em aplicações críticas para a segurança deve levar em conta os testes de aceite e a possível certificação do simulador por um órgão independente.

©International Software Testing Qualifications Board

4.2.4 Considerações organizacionais

Os testes técnicos podem envolver a medição do comportamento de vários componentes em um sistema completo (p. ex., servidores, bancos de dados, redes). Se esses componentes forem distribuídos por vários locais e organizações, o esforço necessário para planejar e coordenar os testes pode ser significativo. Por exemplo, certos componentes de software podem estar disponíveis apenas para testes do sistema em um determinado momento do dia, ou as organizações podem oferecer suporte para testes apenas por um número limitado de dias. A falta de confirmação de que os componentes e o pessoal do sistema (isto é, conhecimento "emprestado") de outras organizações estejam disponíveis "de plantão" para fins de teste pode resultar em uma grave interrupção dos testes programados.

4.2.5 Segurança de dados e proteção de dados

Medidas específicas de segurança implementadas para um sistema devem ser consideradas na fase de planejamento do teste para garantir que todas as atividades de teste sejam possíveis. Por exemplo, o uso da criptografia de dados pode dificultar a criação de dados de teste e a verificação dos resultados.

As políticas e leis de proteção de dados podem impedir a geração de quaisquer dados de teste necessários com base em dados de produção (p. ex., dados pessoais, dados de cartão de crédito). Tornar os dados de teste anônimos é uma tarefa não trivial que deve ser planejada como parte da implementação do teste.

4.3 Teste de Segurança

4.3.1 Razões para considerar os testes de segurança

Os testes de segurança avaliam a vulnerabilidade de um sistema às ameaças que comprometem a política de segurança do sistema. A seguir está uma lista de ameaças potenciais que devem ser exploradas durante os testes de segurança:

- Cópia não autorizada de aplicações ou dados;
- Controle de acesso não autorizado (p. ex., capacidade de executar tarefas para as quais o usuário não tem direitos). Direitos, acessos e privilégios do usuário são o foco deste teste. Estas informações devem estar disponíveis nas especificações do sistema;
- Software que exibe efeitos colaterais indesejados ao executar a função pretendida. Por exemplo, um "*Media Player*" que reproduz corretamente o áudio, mas o faz escrevendo arquivos para armazenamento temporário não criptografado, exibe um efeito colateral que pode ser explorado por piratas de software;
- Código inserido em uma página web que pode ser exercido por usuários subsequentes (cross-site scripting ou XSS). Este código pode ser malicioso;
- Estouro de buffer (buffer overrun) que pode ser causado pela inserção de strings em um campo de entrada de interface de usuário que são maiores do que o código pode manipular

©International Software Testing Qualifications Board

corretamente. Uma vulnerabilidade de estouro de buffer representa uma oportunidade para executar instruções de código maliciosas;
- *Denial of Service*, que impede os usuários de interagir com uma aplicação (p. ex., sobrecarregando um servidor web com solicitações "incômodas");
- A interceptação, imitação e/ou alteração e retransmissão subsequente de comunicações (p. ex., transações com cartão de crédito) por terceiros, de modo que um usuário permanece inconsciente da presença desse terceiro (ataque "man-in-the-middle");
- Quebrar os códigos de criptografia usados para proteger dados confidenciais;
- Bombas lógicas (às vezes chamadas de Ovos de Páscoa – *Easter Eggs*), que podem ser inseridas maliciosamente no código e que só são ativadas sob certas condições (p. ex., em uma data específica). Quando as bombas lógicas são ativadas, elas podem executar atos maliciosos, como a eliminação de arquivos ou a formatação de discos.

4.3.2 Planejamento de Testes de Segurança

Em geral, os seguintes aspectos são de particular relevância quando se planejam testes de segurança:

- Como os problemas de segurança podem ser introduzidos durante a arquitetura, o desenho e a implementação do sistema, os testes de segurança podem ser programados para os níveis de componente, integração e teste do sistema. Devido à natureza variável das ameaças à segurança, os testes de segurança também podem ser agendados regularmente após o sistema ter entrado em produção. Isto é particularmente verdadeiro para arquiteturas dinâmicas abertas, como a Internet das Coisas (IoT), onde a fase de produção é caracterizada por muitas atualizações dos elementos de software e hardware utilizados.
- As abordagens de teste propostas pelo Analista Técnico de Teste podem incluir revisões da arquitetura, desenho e código, e a análise estática do código com ferramentas de segurança. Eles podem ser eficazes para encontrar problemas de segurança que são facilmente esquecidos durante os testes dinâmicos.
- O Analista Técnico de Teste pode ser chamado para planejar e executar certos "ataques" de segurança (ver abaixo), que requerem planejamento e coordenação cuidadosos com os stakeholders (incluindo especialistas em testes de segurança). Outros testes de segurança podem ser realizados em cooperação com desenvolvedores ou analistas de teste (p. ex., testar direitos, acesso e privilégios de usuários).
- Um aspecto essencial do planejamento de testes de segurança é a obtenção de aprovações. Para o Analista Técnico de Teste, isso significa garantir que foi obtida permissão explícita do Gerente de Teste para realizar os testes de segurança planejados. Qualquer teste adicional não planejado realizado pode parecer um ataque real e a pessoa que realiza esses testes pode estar em risco de ação legal. Sem nada por escrito para demonstrar intenção e autorização, pode ser difícil de explicar de forma convincente a desculpa "Estávamos realizando um teste de segurança".
- Todo o planejamento do teste de segurança deve ser coordenado com o responsável pela segurança da informação da organização, se ela possuir essa função.

©International Software Testing Qualifications Board

- Deve-se observar que as melhorias que podem ser feitas na segurança de um sistema podem afetar sua eficiência ou confiabilidade de performance. Após realizar melhorias na segurança, é aconselhável considerar a necessidade de realizar testes de eficiência ou confiabilidade de performance (ver caopítulos 4.4 e 4.5).

Normas individuais podem ser aplicadas ao realizar o planejamento de testes de segurança, tais como [IEC 62443-3-2], que se aplica à automação industrial e sistemas de controle.

O syllabus *Security Testing* [CT_SEC_SYL] inclui mais detalhes sobre os elementos-chave do plano de testes de segurança.

4.3.3 Especificação do Teste de Segurança

Testes particulares de segurança podem ser agrupados [Whittaker04] de acordo com a origem do risco de segurança. Isso inclui:

- **Relacionada à Interface do usuário** - acesso não autorizado e entradas maliciosas;
- **Relacionado ao sistema de arquivos** - acesso a dados confidenciais armazenados em arquivos ou repositórios;
- **Relacionado ao sistema operacional** - armazenamento de informações sensíveis como senhas em forma não criptografada na memória, que podem ser expostas quando o sistema é travado por entradas maliciosas
- **Relacionado à software externo** - interações que podem ocorrer entre componentes externos que o sistema utiliza. Eles podem estar no nível da rede (p. ex., pacotes ou mensagens incorretas passadas) ou no nível do componente de software (p. ex., falha de um componente de software no qual o software se baseia).

As subcaracterísticas de segurança da ISO 25010 [ISO25010] também fornecem uma base a partir da qual os testes de segurança podem ser especificados. Estes se concentram nos seguintes aspectos de segurança:

- Confidencialidade
- Integridade
- Não-repúdio
- Prestação de contas
- Autenticidade

A seguinte abordagem [Whittaker04] pode ser usada para desenvolver testes de segurança:

- Reunir informações que possam ser úteis para especificação dos testes, como nomes de funcionários, endereços físicos, detalhes sobre as redes internas, números IP, identidade do software ou hardware utilizado e versão do sistema operacional.
- Realizar uma varredura de vulnerabilidade usando ferramentas amplamente disponíveis. Tais ferramentas não são usadas diretamente para comprometer o(s) sistema(s), mas para identificar a vulnerabilidade resultante, ou que podem resultar em uma violação da política de segurança. Vulnerabilidades específicas também podem ser identificadas usando informações e *checklists* como as fornecidas pelo *National Institute of Standards and Technology* (NIST) [Web-1] e o *Open Web Application Security Project*™ (OWASP) [Web-4].

©International Software Testing Qualifications Board

- Desenvolver "planos de ataque" (ou seja, um plano de ações de teste destinadas a comprometer a política de segurança de um sistema específico) utilizando as informações coletadas. Várias entradas através de várias interfaces (p. ex., interface de usuário, sistema de arquivos) precisam ser especificadas nos planos de ataque para detectar os mais graves defeitos de segurança. Os vários "ataques" descritos em [Whittaker04] são uma fonte valiosa de técnicas desenvolvidas especificamente para testes de segurança.

Observe que planos de ataque podem ser desenvolvidos para testes de penetração.

O capítulo 3.2 (análise estática) e o syllabus Security Testing [CT_SEC_SYL] incluem mais detalhes sobre testes de segurança.

4.4 Teste de Confiabilidade

4.4.1 Introdução

A classificação ISO 25010 das características de qualidade do produto define as seguintes subcaracterísticas de confiabilidade: maturidade, disponibilidade, tolerância a falhas e recuperabilidade. O teste de confiabilidade diz respeito à capacidade de um sistema ou software de executar funções especificadas sob condições especificadas durante um período especificado.

4.4.2 Teste de Maturidade

A maturidade é o grau em que o sistema (ou software) atende aos requisitos de confiabilidade em condições normais de operação, que são normalmente especificados usando um perfil operacional (ver capítulo 4.9). As medidas de maturidade, quando usadas, geralmente formam um dos critérios de liberação para um sistema.

Tradicionalmente, a maturidade tem sido especificada e medida para sistemas de alta confiabilidade, tais como aqueles associados a funções críticas de segurança (p. ex., um sistema de controle de voo da aeronave), onde a meta de maturidade é definida como parte de uma norma regulatória. Uma exigência de maturidade para tal sistema de alta confiabilidade pode ser um tempo médio entre falhas (MTBF) de até 109 horas (embora isto seja praticamente impossível de medir).

A abordagem usual para testar a maturidade de sistemas de alta confiabilidade é conhecida como modelo de crescimento de confiabilidade, e normalmente ocorre no final dos testes do sistema, após a conclusão dos testes para outras características de qualidade e a correção de quaisquer defeitos associados a falhas detectadas. É uma abordagem estatística geralmente realizada em um ambiente de teste o mais próximo possível do ambiente operacional. Para medir um MTBF especificado, as entradas de teste são geradas com base no perfil operacional e o sistema é executado e as falhas são registradas (e posteriormente corrigidas). A redução da frequência de falhas permite prever o MTBF usando um modelo de crescimento de confiabilidade.

Quando a maturidade é usada como meta para sistemas de menor confiabilidade (p. ex., não relacionados à segurança), então o número de falhas observadas durante um período definido de uso operacional esperado (p. ex., não mais que 2 falhas de alto impacto por semana) pode ser usado e pode ser registrado como parte do acordo de nível de serviço (SLA) para o sistema.

©International Software Testing Qualifications Board

4.4.3 Teste de Disponibilidade

A disponibilidade é geralmente definida em termos do tempo em que um sistema (ou software) está disponível para os usuários e outros sistemas sob condições normais de operação. Os sistemas podem ter uma maturidade baixa, mas ainda têm uma alta disponibilidade. Por exemplo, uma rede telefônica pode não conseguir conectar várias chamadas (e, portanto, ter baixa maturidade), mas desde que o sistema se recupere rapidamente e permita que as próximas tentativas de conectar a maioria dos usuários fiquem satisfeitas. Entretanto, uma única falha que causasse uma interrupção da rede telefônica por várias horas representaria um nível inaceitável de disponibilidade. A disponibilidade é frequentemente especificada como parte de um SLA e medida para sistemas operacionais, tais como websites e aplicações de software como serviço (SaaS). A disponibilidade de um sistema pode ser descrita como 99,999% ('cinco noves'), caso em que ele deve estar indisponível no máximo 5 minutos por ano, alternativamente, a disponibilidade do sistema pode ser especificada em termos de indisponibilidade (p. ex., o sistema não deve estar indisponível por mais de 60 minutos por mês).

A medição da disponibilidade antes da operação (p. ex., como parte da decisão de liberação) é frequentemente realizada usando os mesmos testes usados para medir a maturidade; os testes são baseados em um perfil operacional de uso esperado durante um período prolongado e realizados em um ambiente de teste o mais próximo possível do ambiente operacional. A disponibilidade pode ser medida como MTTF / (MTTF + MTTR), onde MTTF é o tempo médio até a falha e MTTR é o tempo médio até o reparo (MTTR), que muitas vezes é medido como parte dos testes de manutenção. Quando um sistema é de alta confiabilidade e incorpora a recuperabilidade (ver capítulo 4.4.5), então podemos substituir o tempo médio de recuperação pelo MTTR na equação quando o sistema leva algum tempo para se recuperar de uma falha.

4.4.4 Teste de Tolerância a Falhas

Sistemas (ou software) com requisitos de confiabilidade extremamente altos frequentemente incorporam um projeto tolerante a falhas, permitindo idealmente que o sistema continue operando sem tempo de inatividade perceptível quando ocorrerem falhas. A principal medida de tolerância a falhas para um sistema é a capacidade do sistema de tolerar falhas. O teste de tolerância a falhas, portanto, envolve a simulação de falhas para determinar se o sistema pode continuar operando quando tal falha ocorrer. A identificação de condições de falhas potenciais a serem testadas é uma parte importante dos testes de tolerância a falhas.

Um projeto tolerante a falhas normalmente envolverá um ou mais subsistemas duplicados, proporcionando assim um nível de redundância em caso de falha. No caso de software, tais sistemas duplicados precisam ser desenvolvidos independentemente, para evitar falhas em modo comum; esta abordagem é conhecida como programação *N-version*. Os sistemas de controle de voo de aeronaves podem incluir três ou quatro níveis de redundância, com as funções mais críticas implementadas em diversas variantes. Quando a confiabilidade do hardware é uma preocupação, um sistema incorporado pode funcionar em vários processadores diferentes, enquanto um website crítico pode funcionar com um servidor espelho (falha) executando as mesmas funções que está sempre disponível para assumir no caso de falha do servidor primário. Qualquer que seja a

©International Software Testing Qualifications Board

abordagem de tolerância a falhas implementada, o teste normalmente requer tanto a detecção da falha quanto a resposta subsequente à falha a ser testada.

O teste de injeção de defeito testa a robustez de um sistema na presença de defeitos no ambiente do sistema (p. ex., uma fonte de alimentação defeituosa, mensagens de entrada malformadas, um processo ou serviço não disponível, um arquivo não encontrado ou memória não disponível) e defeitos no próprio sistema (p. ex., um bit invertido causado por radiação cósmica, um projeto ruim ou má codificação). O teste de injeção de falha é uma forma de teste negativo - injetamos deliberadamente defeitos no sistema para assegurar que ele reaja da forma esperada (ou seja, com segurança para um sistema relacionado à segurança). Algumas vezes os cenários de defeito que testamos nunca devem ocorrer (p. ex., uma tarefa de software nunca deve 'morrer' ou ficar presa em um loop infinito) e não podem ser simulados pelos testes tradicionais do sistema, mas com os testes de injeção de defeito, criamos o cenário de defeito e medimos o comportamento subsequente do sistema para garantir que ele detecte e trate a falha.

4.4.5 Teste de Recuperabilidade

A recuperabilidade é uma medida da capacidade de um sistema (ou software) de se recuperar de uma falha, seja em termos do tempo necessário para a recuperação (que pode estar em um estado de operação diminuído) ou da quantidade de dados perdidos. As abordagens aos testes de recuperabilidade incluem testes de falha e backup e testes de restauração; ambos normalmente incluem procedimentos de teste baseados em testes em funcionamento a seco e apenas testes práticos ocasionais e, idealmente, sem aviso prévio em ambientes operacionais.

Os testes de backup e restauração se concentram em testar os procedimentos em vigor para minimizar os efeitos de uma falha nos dados do sistema. Os testes avaliam os procedimentos tanto para o backup quanto para a restauração dos dados. Enquanto os testes para backup de dados são relativamente fáceis, os testes para restaurar um sistema a partir de dados de backup podem ser mais complexos e frequentemente requerem planejamento cuidadoso para garantir que a interrupção do sistema operacional seja minimizada. As medidas incluem o tempo necessário para realizar diferentes tipos de backup (p. ex., completo e incremental), o tempo necessário para restaurar os dados (o objetivo do tempo de recuperação) e o nível de perda de dados que é aceitável (objetivo do ponto de recuperação).

Os testes de falha são realizados quando a arquitetura do sistema compreende tanto um sistema primário quanto um sistema de falha que assumirá o controle se o sistema primário falhar. Quando um sistema deve ser capaz de se recuperar de uma falha catastrófica (p. ex., uma inundação, um ataque terrorista ou um grave ataque de resgate), então os testes de falha são frequentemente conhecidos como testes de recuperação de desastres e o sistema (ou sistemas) de falha podem frequentemente estar em outra localização geográfica. A realização de um teste completo de recuperação de desastres em um sistema operacional precisa de um planejamento extremamente cuidadoso devido aos riscos e perturbações (muitas vezes para o tempo livre dos gerentes seniores, que provavelmente será ocupado no gerenciamento da recuperação). Se um teste completo de recuperação de desastre falhar, então voltaríamos imediatamente ao sistema primário (pois ele não foi realmente destruído!). Os testes de falha incluem testar a mudança para o sistema de falha, uma vez que este tenha assumido o controle, fornece o nível de serviço necessário.

©International Software Testing Qualifications Board

4.4.6 Planejamento do Teste de Confiabilidade

Em geral, os seguintes aspectos são de particular relevância quando se planejam testes de confiabilidade:

- Cronograma - O teste de confiabilidade normalmente exige que o sistema completo seja testado e outros tipos de teste já estejam concluídos - e pode levar muito tempo para ser realizado.
- Custos - Sistemas de alta confiabilidade são notoriamente caros de serem testados devido aos longos períodos para os quais devem ser testados sem deixar de ser capaz de prever um MTBF elevado exigido.
- Duração - O teste de maturidade usando modelos de crescimento de confiabilidade é baseado em falhas detectadas e, para altos níveis de confiabilidade, levará muito tempo para obter resultados estatisticamente significativos.
- Ambiente de teste - O ambiente de teste precisa ser o mais semelhante possível ao operacional, ou o ambiente operacional pode ser usado. Entretanto, se utilizar o ambiente operacional, isto pode ser perturbador para os usuários e pode ser de alto risco se, por exemplo, um teste de recuperação de desastres afetar adversamente o sistema operacional.
- Escopo - Diferentes subsistemas e componentes podem ser testados para diferentes tipos e níveis de confiabilidade.
- Critérios de saída - Os requisitos de confiabilidade devem ser estabelecidos por normas regulamentares para aplicações relacionadas à segurança.
- Falha - As medidas de confiabilidade dependem muito da contagem de falhas e, portanto, deve haver um acordo prévio sobre o que constitui uma falha.
- Desenvolvedores - Para testes de maturidade utilizando modelos de crescimento de confiabilidade, é necessário chegar a um acordo com os desenvolvedores que identificaram defeitos o mais rápido possível.
- Medir a confiabilidade operacional é relativamente simples em comparação com medir a confiabilidade antes da liberação, já que temos que medir apenas as falhas; isto pode precisar de ligação com o pessoal de operações.
- Testes iniciais - Atingir alta confiabilidade (ao contrário de medir a confiabilidade) exige que os testes comecem o mais cedo possível, com revisões rigorosas dos documentos iniciais da linha de base e a análise estática do código.

4.4.7 Especificação do Teste de Confiabilidade

Para testar a maturidade e a disponibilidade, os testes se baseiam em grande parte em testar o sistema em condições normais de operação. Para tais testes, é necessário um perfil operacional que defina como o sistema deve ser utilizado. Ver capítulo 4.9 para mais detalhes sobre os perfis operacionais.

Para testar a tolerância a falhas e a recuperabilidade, muitas vezes é necessário gerar testes que reproduzam falhas no ambiente e no próprio sistema, para determinar como o sistema reage. Os testes de injeção de falhas são frequentemente utilizados para isso. Várias técnicas e listas de

©International Software Testing Qualifications Board

verificação estão disponíveis para a identificação de possíveis defeitos e falhas correspondentes (p. ex., Análise de Árvore de Falhas, Modo de Falha e Análise de Efeito).

4.5 Teste de Performance

4.5.1 Introdução

A classificação das características de qualidade do produto da ISO 25010 define as seguintes subcaracterísticas de eficiência de performance: comportamento do tempo, utilização de recursos e capacidade. Os testes de performance (associados à característica de qualidade da eficiência de performance) se preocupam em medir a performance de um sistema ou software sob condições específicas em relação à quantidade de recursos utilizados. Os recursos típicos incluem o tempo decorrido, tempo de CPU, memória e largura de banda.

4.5.2 Teste de Comportamento no Tempo

Os testes de comportamento temporal medem os seguintes aspectos de um sistema (ou software) sob condições operacionais especificas:

- tempo decorrido desde o recebimento de uma solicitação até a primeira resposta (ou seja, o tempo para começar a responder, não o tempo para completar a atividade solicitada), também chamado de tempo de resposta;
- tempo de retorno desde o início de uma atividade até que a atividade seja concluída, também chamado de tempo de processamento;
- número de atividades completadas por unidade de tempo (p. ex., número de operações de banco de dados por segundo), também chamado de taxa de transferência.

Para muitos sistemas, os tempos máximos de resposta para diferentes funções do sistema são especificados como requisitos. Nesses casos, o tempo de resposta é o tempo decorrido mais o tempo de retorno. Quando um sistema tem que executar uma série de etapas (p. ex., uma tubulação) para completar uma atividade, pode ser útil medir o tempo gasto para cada etapa e analisar os resultados para determinar se uma ou mais etapas estão causando um gargalo.

4.5.3 Teste de Utilização de Recursos

Os testes de utilização de recursos medem os seguintes aspectos de um sistema (ou software) sob condições operacionais específicas:

- utilização da CPU, normalmente como porcentagem do tempo disponível da CPU;
- utilização da memória, normalmente como porcentagem da memória disponível;
- utilização do i/o do dispositivo, normalmente como porcentagem do tempo disponível do i/o do dispositivo;
- utilização da largura de banda, normalmente como porcentagem da largura de banda disponível.

©International Software Testing Qualifications Board

4.5.4 Teste de Capacidade

Os testes de capacidade medem os limites máximos para os seguintes aspectos de um sistema (ou software) sob condições operacionais específicas:

- transações processadas por unidade de tempo (p. ex., máximo de 687 palavras traduzidas por minuto);
- usuários acessando simultaneamente o sistema (p. ex., máximo de 1223 usuários);
- novos usuários adicionados para ter acesso ao sistema por unidade de tempo (p. ex., máximo de 400 usuários adicionados por segundo).

4.5.5 Aspectos comuns dos Testes de Performance

Ao testar o comportamento do tempo, utilização de recursos ou capacidade, é normal que várias medidas sejam tomadas e a média usada como medida relatada; isto porque os valores de tempo medidos podem flutuar dependendo de outras tarefas de fundo que o sistema possa estar realizando. Em algumas situações, as medições serão tratadas de uma maneira mais meticulosa (p. ex., usando variância ou outras medidas estatísticas), ou anômalas podem ser investigadas e descartadas, se apropriado.

A análise dinâmica (ver capítulo 3.3.4) pode ser usada para identificar componentes que causam um gargalo, medir os recursos usados para testes de utilização de recursos e medir os limites máximos para testes de capacidade.

4.5.6 Tipos de Testes de Performance

Os testes de performance diferem da maioria das outras formas de testes na medida em que podem existir dois objetivos distintos. O primeiro é determinar se o software em teste atende aos critérios de aceite específico. Por exemplo, determinar se o sistema exibe uma página web solicitada dentro do máximo especificado de 4 segundos. O segundo objetivo é fornecer informações aos desenvolvedores do sistema para ajudá-los a melhorar a eficiência do sistema. Por exemplo, detectar gargalos e identificar quais partes da arquitetura do sistema são adversamente afetadas quando um número inesperadamente alto de usuários acessa o sistema simultaneamente.

Os testes de performance descritos nos capítulos 4.5.2, 4.5.3 e 4.5.4 podem ser todos usados para determinar se o software em teste atende aos critérios de aceite específico. Eles também são usados para medir valores de base que são usados para comparação posterior quando o sistema é modificado. Os seguintes tipos de teste de performance são mais frequentemente usados para fornecer informações aos desenvolvedores sobre como o sistema responde sob diferentes condições operacionais.

Teste de Carga

Os testes de carga se concentram na capacidade de um sistema de lidar com diferentes cargas. Essas cargas são normalmente definidas em termos do número de usuários acessando o sistema simultaneamente ou do número de processos simultâneos em execução e podem ser definidas como perfis operacionais (ver capítulo 4.9 para mais detalhes sobre perfis operacionais). A

©International Software Testing Qualifications Board

manipulação dessas cargas é tipicamente medida em termos do comportamento do tempo do sistema, e utilização de recursos (p. ex., determinando o efeito sobre o tempo de resposta da duplicação do número de usuários). Ao realizar testes de carga, é prática normal começar com uma carga baixa e aumentar gradualmente, enquanto se mede o comportamento do tempo do sistema e a utilização de recursos. Informações típicas de testes de carga que poderiam ser úteis aos desenvolvedores incluiriam mudanças inesperadas em tempos de resposta ou no uso de recursos do sistema quando o sistema lidou com uma determinada carga.

Teste de Stress

Há dois tipos de testes de stress; o primeiro é semelhante ao teste de carga e o segundo é uma forma de teste de robustez.

No primeiro, os testes de carga são normalmente realizados com a carga inicialmente ajustada ao máximo esperado e depois aumentada até que o sistema falhe (p. ex., os tempos de resposta tornam-se excessivamente longos ou o sistema trava). Algumas vezes, em vez de forçar o sistema a falhar, uma carga alta é usada para tensionar o sistema e então a carga é reduzida a um nível normal e o sistema é verificado para garantir que seus níveis de performance tenham se recuperado para seus níveis de pretensão.

Teste de Escalabilidade

Um sistema escalável pode se adaptar a diferentes cargas. Por exemplo, um website escalável poderia escalar para usar mais servidores back-end à medida que a demanda aumenta e usar menos quando a demanda diminui. O teste de escalabilidade é semelhante ao teste de carga, mas testa a capacidade de um sistema de escalar para cima e para baixo quando confrontado com mudanças de carga (p. ex., mais usuários do que o hardware atual pode suportar).

O syllabus *Performance Testing* [CT_PT_SYL] inclui outros tipos de testes de performance.

4.5.7 Planejamento de Testes de Performance

Em geral, os seguintes aspectos são de particular relevância no planejamento de testes de eficiência de performance:

- Cronograma - Os testes de performance muitas vezes exigem que todo o sistema seja implementado e executado em um ambiente de teste representativo, o que significa que normalmente é executado como parte dos testes do sistema.
- Revisões - As revisões de código, em particular aquelas que se concentram na interação do banco de dados, com os componentes e no tratamento de erros, podem identificar problemas de eficiência de performance (particularmente em relação à lógica de "esperar e tentar novamente" e consultas ineficientes) e devem ser programadas para acontecer assim que o código estiver disponível (ou seja, antes dos testes dinâmicos).
- Testes antecipados - Alguns testes de performance (p. ex., determinação da utilização da CPU para um componente crítico) podem ser programados como parte dos testes de componentes. Os componentes identificados como sendo um gargalo pelo teste de performance podem ser atualizados e testados novamente isoladamente como parte do teste de componentes.

©International Software Testing Qualifications Board

- Mudanças na arquitetura - Resultados adversos dos testes de desempenho podem, às vezes, resultar em uma mudança na arquitetura do sistema. Onde importantes mudanças no sistema poderiam ser sugeridas por resultados de testes de desempenho, que devem começar o mais cedo possível, para maximizar o tempo disponível para resolver tais questões.
- Custos - Ferramentas e ambientes de teste podem ser caros, o que significa que ambientes de teste temporários baseados em nuvens podem ser alugados, e licenças de "recarga" de ferramentas podem ser usadas. Nesses casos, o planejamento de testes normalmente precisa otimizar o tempo gasto com testes para minimizar os custos.
- Ambiente de teste - O ambiente de teste precisa ser o mais representativo possível do ambiente operacional, caso contrário o desafio de dimensionar os resultados do teste de performance do ambiente de teste para o ambiente operacional esperado é maior.
- Critérios de saída - Os requisitos de eficiência de performance às vezes podem ser difíceis de obter do cliente e, portanto, muitas vezes derivam de linhas de base de sistemas anteriores ou similares. No caso de sistemas de segurança incorporados, alguns requisitos, como a quantidade máxima de CPU e memória utilizada, podem ser especificados por normas regulatórias.
- Ferramentas - Ferramentas de geração de carga são frequentemente necessárias para suportar testes de performance. Por exemplo, verificar a escalabilidade de um website popular pode exigir a simulação de centenas de milhares de usuários virtuais. As ferramentas que simulam restrições de recursos também são especialmente úteis para testes de estresse. Deve-se tomar cuidado para garantir que quaisquer ferramentas adquiridas para suportar os testes sejam compatíveis com os protocolos de comunicação usados pelo sistema em teste.

O syllabus *Performance Testing* [CT_PT_SYL] inclui mais detalhes do planejamento do teste de performance.

4.6 Teste de Manutenção

O software muitas vezes passa muito mais de sua vida útil sendo mantido do que sendo desenvolvido. Para garantir que a tarefa de realizar a manutenção seja a mais eficiente possível, são realizados testes de manutenção para medir a facilidade com que o código pode ser analisado, alterado, testado, modularizado e reutilizado. Os testes de manutenção não devem ser confundidos com testes de manutenção, que são realizados para testar as mudanças feitas no software operacional.

Os objetivos típicos de manutenção dos stakeholders afetadas (p. ex., o proprietário ou operador do software) incluem:

- Minimizar o custo de possuir ou operar o software
- Minimização do tempo de inatividade necessário para a manutenção do software

Os testes de manutenção devem ser incluídos em uma abordagem de teste onde um ou mais dos seguintes fatores se aplicam:

©International Software Testing Qualifications Board

- É provável que haja mudanças no software após o início da produção (p. ex., para corrigir defeitos ou introduzir atualizações planejadas)
- Os benefícios de alcançar os objetivos de manutenção durante o ciclo de vida de desenvolvimento de software são considerados pelos stakeholders afetadas para compensar os custos de realizar os testes de manutenção e fazer quaisquer mudanças necessárias

Os riscos de má manutenção do software (p. ex., longos tempos de resposta aos defeitos relatados pelos usuários e/ou clientes) justificam a realização de testes de manutenção

4.6.1 Teste de Manutenção estática e dinâmica

Técnicas apropriadas para testes de manutenção estática incluem análises e revisões estáticas, conforme discutido nos capítulos 3.2 e 5.2. Os testes de manutenção devem ser iniciados assim que a documentação do projeto estiver disponível e devem continuar durante todo o esforço de implementação do código. Como a capacidade de manutenção está incorporada ao código e à documentação para cada componente do código, a capacidade de manutenção pode ser avaliada no início do ciclo de vida do desenvolvimento do software sem ter que esperar por um sistema completo e em funcionamento.

Os testes dinâmicos de manutenção se concentram nos procedimentos documentados desenvolvidos para manter uma determinada aplicação (p. ex., para a realização de atualizações de software). Os cenários de manutenção são utilizados como casos de teste para garantir que os níveis de serviço necessários sejam atingíveis com os procedimentos documentados. Esta forma de teste é particularmente relevante quando a infraestrutura subjacente é complexa, e os procedimentos de suporte podem envolver vários departamentos/organizações. Esta forma de teste pode ocorrer como parte dos testes de aceite operacional.

4.6.2 Subcaracterísticas de manutenção

A capacidade de manutenção de um sistema pode ser medida em termos de:

- Analisabilidade
- Modificabilidade
- Testabilidade

Os fatores que influenciam estas características incluem a aplicação de boas práticas de programação (p. ex., comentários, nomeação de variáveis, indentação), e a disponibilidade de documentação técnica (p. ex., especificações de projeto de sistema, especificações de interface).

Outras subcaracterísticas relevantes da qualidade para manutenção [ISO 25010] são:

- Modularidade
- Reusabilidade

A modularidade pode ser testada por meio de análise estática (ver capítulo 3.2.3). Os testes de reusabilidade podem tomar a forma de revisões arquitetônicas (ver capítulo 5).

©International Software Testing Qualifications Board

4.7 Teste de Portabilidade

4.7.1 Introdução

Em geral, os testes de portabilidade estão relacionados ao grau em que um componente ou sistema de software pode ser transferido para seu ambiente pretendido (inicialmente ou de um ambiente existente), pode ser adaptado a um novo ambiente, ou pode substituir outra entidade.

A ISO 25010 [ISO 25010] inclui as seguintes subcaracterísticas de portabilidade:

- Instabilidade
- Adaptabilidade
- Substituibilidade

Os testes de portabilidade podem começar com os componentes individuais (p. ex., a substituibilidade de um determinado componente, como a mudança de um sistema de gerenciamento de banco de dados para outro) e depois expandir o escopo à medida que mais código se torna disponível. A capacidade de instalação pode não ser testada até que todos os componentes do produto estejam funcionando funcionalmente.

A portabilidade deve ser projetada e incorporada ao produto e, portanto, deve ser considerada no início das fases de projeto e arquitetura. As revisões de arquitetura e projeto podem ser particularmente produtivas para identificar potenciais exigências e problemas de portabilidade (p. ex., dependência de um determinado sistema operacional).

4.7.2 Teste de Instalabilidade

O teste de instalabilidade é realizado no software e nos procedimentos escritos usados para instalar o software em seu ambiente alvo. Isto pode incluir, por exemplo, o software desenvolvido para instalar um sistema operacional, ou um "assistente" de instalação usados para instalar um produto em um PC cliente.

Os objetivos típicos dos testes de instalabilidade incluem o seguinte:

- Validando que o software pode ser instalado seguindo as instruções em um manual de instalação (incluindo a execução de quaisquer scripts de instalação), ou usando um assistente de instalação. Isto inclui o exercício de opções de instalação para diferentes configurações de hardware/software e para vários graus de instalação (p. ex., inicial ou atualização).
- Testar se as falhas que ocorrem durante a instalação (p. ex., falha no carregamento de DLLs específicas) são tratadas corretamente pelo software de instalação sem deixar o sistema em um estado indefinido (p. ex., software parcialmente instalado ou configurações incorretas do sistema)
- Teste se uma instalação/de-instalação parcial pode ser concluída
- Teste se um assistente de instalação pode identificar plataformas de hardware inválidas ou configurações de sistema operacional
- Medir se o processo de instalação pode ser concluído dentro de um número especificado de minutos ou dentro de um número especificado de etapas
- Validando que o software pode ser rebaixado ou desinstalado

©International Software Testing Qualifications Board

O teste de aptidão funcional é normalmente realizado após o teste de instalação para detectar quaisquer defeitos que possam ter sido introduzidos pela instalação (p. ex., configurações incorretas, funções não disponíveis). O teste de usabilidade é normalmente realizado em paralelo com o teste de instabilidade (p. ex., para validar que os usuários recebem instruções compreensíveis e mensagens de feedback/erro durante a instalação).

4.7.3 Teste de Adaptabilidade

Os testes de adaptabilidade verificam se uma determinada aplicação pode funcionar corretamente em todos os ambientes-alvo previstos (hardware, software, middleware, sistema operacional etc.). A especificação de testes de adaptabilidade requer que os ambientes alvo pretendidos sejam identificados, configurados e disponíveis para a equipe de testes. Estes ambientes são então testados usando uma seleção de casos de teste funcionais que exercem os vários componentes presentes no ambiente.

A adaptabilidade pode estar relacionada à capacidade do software de ser portado para vários ambientes especificados, realizando um procedimento pré-definido. Os testes podem avaliar este procedimento.

4.7.4 Teste de Substitutibilidade

O teste de substitutibilidade se concentra na capacidade de um componente de software de substituir um componente de software existente em um sistema. Isto pode ser particularmente relevante para sistemas que utilizam software *comercial-off-the-shelf* (COTS) para componentes específicos do sistema ou para aplicações IoT.

Os testes de substitutibilidade podem ser executados em paralelo com testes de integração funcional onde mais de um componente alternativo está disponível para integração no sistema completo. A substitutibilidade também pode ser avaliada por revisão técnica ou inspeção nos níveis de arquitetura e projeto, onde a ênfase é colocada na definição clara das interfaces do componente que pode ser usado como substituto.

4.8 Teste de Compatibilidade

4.8.1 Introdução

Os testes de compatibilidade consideram os seguintes aspectos [ISO 25010]:

- Coexistência
- Interoperabilidade

4.8.2 Teste de Coexistência

Diz-se que os sistemas de computador que não estão relacionados entre si coexistem quando podem funcionar no mesmo ambiente (p. ex., no mesmo hardware) sem afetar o comportamento um do outro (p. ex., conflitos de recursos). Os testes de coexistência devem ser realizados quando um software novo ou atualizado for lançado em ambientes que já contenham aplicações instaladas.

©International Software Testing Qualifications Board

Podem surgir problemas de coexistência quando a aplicação é testada em um ambiente onde é a única aplicação instalada (onde os problemas de incompatibilidade não são detectáveis) e depois implantada em outro ambiente (p. ex., produção) que também executa outras aplicações.

Os objetivos típicos dos testes de coexistência incluem:

- Avaliação de possível impacto adverso na adequação funcional quando as aplicações são carregadas no mesmo ambiente (p. ex., uso conflitante de recursos quando um servidor executa várias aplicações)
- Avaliação do impacto em qualquer aplicação resultante da implantação de reparos e atualizações do sistema operacional

As questões de coexistência devem ser analisadas ao planejar o ambiente de produção visado, mas os testes reais são normalmente realizados após a conclusão dos testes do sistema.

4.9 Perfis Operacionais

Os perfis operacionais são usados como parte da especificação de testes para vários tipos de testes não-funcionais, incluindo testes de confiabilidade e de performance. Eles são especialmente úteis quando a exigência a ser testada inclui a restrição de "sob condições especificadas", pois podem ser usados para definir essas condições.

O perfil operacional define um padrão de uso para o sistema, normalmente em termos dos usuários do sistema e das operações realizadas pelo sistema. Os usuários são tipicamente especificados em termos de quantos deverão estar usando o sistema (e em que horários), e talvez seu tipo (p. ex., usuário primário, usuário secundário). As diferentes operações que se espera que sejam realizadas pelo sistema são tipicamente especificadas, com sua frequência (e probabilidade de ocorrência). Estas informações podem ser obtidas usando ferramentas de monitoramento (onde a aplicação real ou similar já está disponível) ou prevendo o uso baseado em algoritmos ou estimativas fornecidas pela organização empresarial.

Ferramentas podem ser usadas para gerar entradas de teste com base no perfil operacional, muitas vezes usando uma abordagem que gera as entradas de teste de forma pseudoaleatória. Tais ferramentas podem ser usadas para criar usuários "virtuais" ou simulados em quantidades que correspondam ao perfil operacional (p. ex., para testes de confiabilidade e disponibilidade) ou que o excedam (p. ex., para testes de tensão ou capacidade). Consulte o capítulo 6.2.3 para mais detalhes sobre estas ferramentas.

©International Software Testing Qualifications Board

5 Revisões [165 min]

Palavras-chave

revisão, revisão técnica

Objetivos de aprendizagem

5.1 Tarefas do Analista Técnico de Teste

TTA 5.1.1 (K2) Explicar por que a preparação da revisão é importante para o Analista Técnico de Teste

5.2 Utilização de listas de verificação

TTA 5.2.1 (K4) Analisar um projeto arquitetônico e identificar problemas de acordo com uma lista de verificação fornecida no syllabus.

TTA 5.2.2 (K4) Analisar uma seção de código ou pseudocódigo e identificar problemas de acordo com uma lista de verificação fornecida no syllabus.

©International Software Testing Qualifications Board

5.1 Tarefas do Analista Técnico de Teste em Revisões

Os Analistas Técnicos de Teste devem ser participantes ativos no processo de revisão técnica, fornecendo suas opiniões únicas. Todos os participantes da revisão devem ter treinamento formal de revisão para compreender melhor suas respectivas funções e devem estar comprometidos com os benefícios de uma revisão técnica bem conduzida. Isto inclui manter uma relação de trabalho construtiva com os autores ao descrever e discutir os comentários de revisão. Para uma descrição detalhada das revisões técnicas, incluindo numerosas listas de verificação de revisões, consulte [Wiegers02]. Os Analistas Técnicos de Teste normalmente participam de revisões e inspeções técnicas onde eles trazem um ponto de vista operacional (comportamental) que pode ser perdido pelos desenvolvedores. Além disso, os Analistas Técnicos de Teste desempenham um papel importante na definição, aplicação e manutenção das listas de verificação de revisão e no fornecimento de informações sobre a gravidade dos defeitos.

Independentemente do tipo de revisão que esteja sendo realizada, o Analista Técnico de Teste deve ter tempo suficiente para se preparar. Isto inclui tempo para rever o produto de trabalho, tempo para verificar a documentação cruzada para verificar a consistência, e tempo para determinar o que pode estar faltando no produto de trabalho. Sem tempo adequado de preparação, a revisão pode se tornar um exercício de edição em vez de uma verdadeira revisão. Uma boa revisão inclui compreender o que está escrito, determinar o que está faltando, verificar a exatidão com relação aos aspectos técnicos e verificar se o produto descrito é consistente com outros produtos já desenvolvidos ou que estão em desenvolvimento. Por exemplo, ao revisar um plano de teste de nível de integração, o Analista Técnico de Teste também deve considerar os itens que estão sendo integrados. Eles estão prontos para a integração? Há dependências que devem ser documentadas? Há dados disponíveis para testar os pontos de integração? Uma revisão não é isolada para o produto de trabalho que está sendo revisado. Deve também considerar a interação desse item com outros itens do sistema.

5.2 Utilização de Listas de Verificação em revisões

As listas de verificação (checklists) são usadas durante as revisões para lembrar os participantes de verificar pontos específicos durante a revisão. As listas de verificação também podem ajudar a despersonalizar a revisão, por exemplo, "esta é a mesma lista de verificação que usamos para cada revisão, e não estamos visando apenas seu produto de trabalho". As listas de verificação podem ser genéricas e utilizadas para todas as revisões ou focalizadas em características ou áreas específicas de qualidade. Por exemplo, uma lista de verificação genérica pode verificar o uso adequado dos termos “deve” e “deve” verificar a formatação adequada e itens de conformidade similares. Uma lista de verificação pode se concentrar em questões de segurança ou de eficiência de performance.

As listas de verificação mais úteis são aquelas desenvolvidas gradualmente por uma organização individual, porque refletem:

- A natureza do produto
- O ambiente de desenvolvimento local o Pessoal
  - Ferramentas
  - Prioridades

©International Software Testing Qualifications Board

- Histórico de sucessos e defeitos anteriores
- Questões particulares (p. ex., eficiência de performance, segurança)

As listas de verificação devem ser personalizadas para a organização e talvez para o projeto em particular. As listas de verificação neste capítulo são exemplos.

5.2.1 Revisões Arquitetônicas

A arquitetura de software consiste nos conceitos ou propriedades fundamentais de um sistema, incorporados em seus elementos, relações e nos princípios de seu projeto e evolução [ISO 42010].

As listas de verificação utilizadas para revisões de arquitetura do comportamento temporal de websites poderiam, por exemplo, incluir a verificação da implementação adequada dos seguintes itens, que são citados da [Web-2]:

- "*Pooling*" de conexões - reduzindo o tempo de execução associado ao estabelecimento de conexões de banco de dados através do estabelecimento de um pool compartilhado de conexões;
- Balanceamento de carga - espalhando a carga uniformemente entre um conjunto de recursos
- Processamento distribuído;
- Caching - usando uma cópia local dos dados para reduzir o tempo de acesso;
- Instanciação preguiçosa;
- Concorrência nas transações;
- Isolamento de processos entre Processamento Transacional Online (OLTP) e Processamento Analítico Online (OLAP);
- Replicação de dados".

5.2.2 Revisões de Código

Listas de verificação (checklists) para revisões de código são necessariamente de baixo nível e são mais úteis quando são específicas de cada idioma. A inclusão de antipadrões de nível de código é útil, particularmente para desenvolvedores de software menos experientes.

As listas de verificação utilizadas para revisão de códigos poderiam incluir os seguintes itens:

1. Estrutura
   - O código implementa o projeto corretamente e em sua completude?
   - O código está de acordo com alguma norma de codificação pertinente?
   - O código é bem estruturado, consistente no estilo e formatado de forma consistente?
   - Há algum procedimento desnecessário ou desnecessário ou algum código inalcançável?
   - Há algum resto de simuladores ou rotinas de teste no código?
   - Qualquer código pode ser substituído por chamadas a componentes externos reutilizáveis ou funções de biblioteca?
   - Há algum bloco de código repetido que poderia ser condensado em um único procedimento?
   - O uso do armazenamento é eficiente?
   - São usados símbolos em vez de constantes numéricas ou strings?

©International Software Testing Qualifications Board

- Algum módulo é excessivamente complexo e deve ser reestruturado ou dividido em submódulos?

2. Documentação
   - O código está claro e adequadamente documentado com um estilo de comentário fácil de manter?
   - Todos os comentários são consistentes com o código?
   - A documentação está em conformidade com as normas aplicáveis?
3. Variáveis
   - Todas as variáveis estão devidamente definidas com nomes significativos, consistentes e claros?
   - Existe alguma variável redundante ou não utilizada?
4. Operações aritméticas
   - O código evita comparar números de ponto flutuante para a igualdade?
   - O código evita sistematicamente erros de arredondamento?
   - O código evita adições e subtrações em números com magnitudes muito diferentes?
   - Os divisores são testados para zero?
5. Loops e Ramificações
   - Todos os loops, ramificações e construções lógicas são completos, corretos e corretamente aninhados?
   - Os casos mais comuns são testados primeiro nas cadeias IF-ELSEIF?
   - Todos os casos são cobertos em um bloco IF-ELSEIF ou CASE, incluindo cláusulas ELSE ou DEFAULT?
   - Toda instrução de caso tem um padrão?
   - As condições de terminação de loop são óbvias e invariavelmente realizáveis?
   - Os índices ou subscrições são inicializados corretamente, imediatamente antes do loop?
   - Qualquer instrução que esteja contida dentro de loops pode ser colocada fora dos loops?
     - O código no loop evita manipular a variável de índice ou usá-la ao sair do loop?
6. Programação defensiva
   - Os índices, indicadores e subscritos são testados contra array, registro ou limite de arquivo?
   - Os dados importados e os argumentos de entrada são testados quanto à validade e integralidade?
   - Todas as variáveis de saída são atribuídas?
   - O elemento de dados correto é operado em cada instrução?
   - Toda alocação de memória é liberada?
   - Os timeouts ou armadilhas de erros são usados para acesso a dispositivos externos?
   - Os arquivos são verificados quanto à existência antes de tentar acessá-los?
   - Todos os arquivos e dispositivos são deixados no estado correto após o término do programa?

©International Software Testing Qualifications Board

6 Ferramentas de Teste e Automação [180 min]

Palavras-chave

captura/reprodução, teste de dados, emulador, injeção de falha, semeadura de falha, teste por palavra-chave, teste baseado em modelo (MBT), simulador, execução de teste

Objetivos de aprendizagem

6.1 Definindo o Projeto de Automação de Testes

TTA-6.1.1 (K2) Resumir as atividades que o Analista Técnico de Testes realiza ao estabelecer um projeto de automação de testes

TTA-6.1.2 (K2) Resumir as diferenças entre automação baseada em dados e automação baseada em palavras-chave

TTA-6.1.3 (K2) Resumir questões técnicas comuns que fazem com que os projetos de automação não atinjam o retorno planejado do investimento

TTA-6.1.4 (K3) Construir palavras-chave com base em um determinado processo de negócio.

6.2 Ferramentas de teste específicas

TTA-6.2.1 (K2) Resumir a finalidade das ferramentas para semeadura de falhas e injeção de falhas

TTA-6.2.2 (K2) Resumir as principais características e problemas de implementação de ferramentas de teste de performance

TTA-6.2.3 (K2) Explicar o propósito geral das ferramentas usadas para testes baseados na web

TTA-6.2.4 (K2) Explicar como as ferramentas apoiam a prática de testes baseados em modelos

TTA-6.2.5 (K2) Esboçar a finalidade das ferramentas usadas para apoiar o teste de componentes e o processo de construção TTA-6.2.6 (K2) Esboçar a finalidade das ferramentas usadas para apoiar o teste de aplicações móveis

©International Software Testing Qualifications Board

6.1 Definindo o projeto de Automação de Testes

Para ser econômico, as ferramentas de teste (e particularmente aquelas que suportam a execução de testes), devem ser cuidadosamente arquitetadas e projetadas. Implementar uma estratégia de automação de execução de testes sem uma arquitetura sólida geralmente resulta em um conjunto de ferramentas de manutenção dispendiosa, insuficiente para o propósito e incapaz de atingir o retorno do investimento alvo.

Um projeto de automação de testes deve ser considerado um projeto de desenvolvimento de software. Isto inclui a necessidade de documentação de arquitetura, documentação detalhada de projeto, revisões de projeto e código, testes de integração de componentes e componentes, assim como testes finais do sistema. Os testes podem ser desnecessariamente atrasados ou complicados quando é utilizado um código de automação de teste instável ou impreciso.

Há várias tarefas que o Analista Técnico de Teste pode realizar com relação à automação da execução de testes. Estas incluem:

- Determinação de quem será responsável pela execução do teste (possivelmente em coordenação com um Gerente de Teste)
- Selecionar a ferramenta apropriada para a organização, cronograma, habilidades da equipe e requisitos de manutenção (note que isto pode significar decidir criar uma ferramenta para usar em vez de adquirir uma)
- Definição dos requisitos de interface entre a ferramenta de automação e outras ferramentas, como o gerenciamento de testes, gerenciamento de defeitos e ferramentas utilizadas para integração contínua
- Desenvolver adaptadores que possam ser necessários para criar uma interface entre a ferramenta de execução de testes e o software em teste
- Selecionando a abordagem de automação, ou seja, orientada por palavras-chave ou por dados (ver capítulo 6.1.1)
- Trabalhando com o Gerente de Teste para estimar o custo da implementação, incluindo treinamento. No desenvolvimento ágil de software, este aspecto seria tipicamente discutido e acordado em reuniões de planejamento de projeto/impressão com toda a equipe.
- Agendando o projeto de automação e alocando o tempo para a manutenção
- Treinamento dos Analistas de Teste e Analistas de Negócios para usar e fornecer dados para a automação
- Determinando como e quando os testes automatizados serão executados
- Determinar como os resultados dos testes automatizados serão combinados com os resultados dos testes manuais

Em projetos com forte ênfase na automação de testes, um Engenheiro de Automação de Testes pode ser encarregado de muitas dessas atividades (veja o syllabus *Test Automation Engineer* [CT_TAE_SYL] para detalhes). Certas tarefas organizacionais podem ser assumidas por um Gerente de Teste de acordo com as necessidades e preferências do projeto. No desenvolvimento ágil de software, a atribuição destas tarefas a funções é tipicamente mais flexível e menos formal.

©International Software Testing Qualifications Board

Estas atividades e as decisões resultantes influenciarão a escalabilidade e a capacidade de manutenção da solução de automação. Deve-se gastar tempo suficiente pesquisando as opções, investigando as ferramentas e tecnologias disponíveis e compreendendo os planos futuros para a organização.

6.1.1 Selecionando a abordagem de automação

Este capítulo considera os seguintes fatores que impactam a abordagem de automação de testes:

- Automatizando através da GUI, API e CLI
- Aplicando uma abordagem baseada em dados
- Aplicando uma abordagem orientada por palavras-chave
- Manuseio de falhas de software
- Considerando o estado do sistema

O syllabus *Test Automation Engineer* [CT_TAE_SYL] inclui mais detalhes sobre a seleção de uma abordagem de automação.

Automatizando através da GUI, API e CLI

A automação de testes não se limita aos testes através da GUI. Também existem ferramentas para ajudar a automatizar os testes no nível da API, através de uma interface de linha de comando (CLI) e outros pontos de interface no software em teste. Uma das primeiras decisões que o Analista Técnico de Teste deve tomar é determinar a interface mais eficaz a ser acessada para automatizar os testes. Ferramentas gerais de execução de testes requerem o desenvolvimento de adaptadores para estas interfaces. O planejamento deve considerar o esforço para o desenvolvimento do adaptador.

Uma das dificuldades dos testes através da GUI é a tendência da GUI a mudar à medida que o software evolui. Dependendo da forma como o código de automação de teste é projetado, isto pode resultar em uma significativa carga de manutenção. Por exemplo, o uso da capacidade de captura/reprodução de uma ferramenta de automação de teste pode resultar em casos de teste automatizados (muitas vezes chamados de scripts de teste) que não funcionam mais como desejado se a GUI mudar. Isto porque o script gravado captura as interações com os objetos gráficos quando o testador executa o software manualmente. Se os objetos acessados mudarem, os scripts gravados também podem precisar de atualização para refletir essas mudanças.

Ferramentas de captura/reprodução podem ser usadas como um ponto de partida conveniente para o desenvolvimento de scripts de automação. O testador registra uma sessão de teste e o script gravado é então modificado para melhorar a capacidade de manutenção (p. ex., substituindo seções do script gravado por funções reutilizáveis).

Aplicando uma abordagem baseada em dados

Dependendo do software sendo testado, os dados usados para cada teste podem ser diferentes, embora as etapas de teste executadas sejam virtualmente idênticas (p. ex., o tratamento do erro de teste de um campo de entrada inserindo múltiplos valores inválidos e verificando o erro retornado para cada um). É ineficiente desenvolver e manter um script de teste automatizado para cada um destes valores a serem testados. Uma solução técnica comum para este problema é mover os dados dos scripts para um armazenamento externo, como uma planilha ou um banco de dados. As funções

©International Software Testing Qualifications Board

são escritas para acessar os dados específicos para cada execução do script de teste, o que permite que um único script funcione através de um conjunto de dados de teste que fornece os valores de entrada e os valores de resultado esperados (p. ex., um valor mostrado em um campo de texto ou uma mensagem de erro). Esta abordagem é chamada de ***data-driven***.

Ao utilizar esta abordagem, além dos scripts de teste que processam os dados fornecidos, são necessários um arnês e uma infraestrutura para apoiar a execução do script ou conjunto de scripts. Os dados reais mantidos na planilha ou no banco de dados são criados por Analistas de Teste que estão familiarizados com a função de negócio do software. No desenvolvimento ágil do software, o representante do negócio (p. ex., o Product Owner) também pode estar envolvido na definição dos dados, em particular para testes de aceite. Esta divisão de trabalho permite que os responsáveis pelo desenvolvimento de scripts de teste (p. ex., o Analista Técnico de Teste) se concentrem na implementação de scripts de automação enquanto o Analista de Testes mantém a propriedade do teste real. Na maioria dos casos, o Analista de Testes será responsável pela execução dos scripts de teste uma vez que a automação seja implementada e testada.

Aplicando uma abordagem orientada por palavras-chave

Outra abordagem, chamada de palavra-chave ou ação por palavras (***keyword-driven***), vai um passo além ao separar também a ação a ser executada nos dados de teste fornecidos do roteiro do teste [Buwalda01]. Para realizar esta separação adicional, é criada uma linguagem de alto nível que é descritiva em vez de diretamente executável. As palavras-chave podem ser definidas tanto para ações de alto quanto de baixo nível. Por exemplo, as palavras-chave do processo de negócio poderiam incluir "Login", “CreateUser” e “DeleteUser”. Tais palavras-chave descrevem as ações de alto nível que serão executadas no domínio da aplicação. Ações de nível inferior denotam a interação com a própria interface do software. Palavras-chaves como: "ClickButton", "SelectFromList", ou "TraverseTree" podem ser usadas para testar as capacidades da GUI que não se encaixam perfeitamente nas palavras-chave do processo de negócio. As palavras-chave podem conter parâmetros, por exemplo, a palavra-chave "Login" poderia ter dois parâmetros: “Username” e “Password”.

Uma vez definidas as palavras-chave e os dados a serem utilizados, o automatizador de testes (p. ex., Analista Técnico de Testes ou Engenheiro de Automação de Testes) traduz as palavras-chave do processo de negócio e ações de nível inferior em código de automação de testes. As palavras-chave e as ações, juntamente com os dados a serem usados, podem ser armazenadas em planilhas ou inseridas usando ferramentas específicas que suportam a automação de testes guiados por palavras-chave. A estrutura de automação de teste implementa a palavra-chave como um conjunto de uma ou mais funções ou scripts executáveis. As ferramentas leem os casos de teste escritos com palavras-chave e chamam as funções ou scripts de teste apropriados que os implementam. Os executáveis são implementados de forma altamente modular para permitir um mapeamento fácil para palavras-chave específicas. Habilidades de programação são necessárias para implementar estes scripts modulares.

Esta separação do conhecimento da lógica de negócio da programação real necessária para implementar os scripts de automação de testes proporciona o uso mais eficaz dos recursos de teste. O Analista Técnico de Teste, no papel de automatizador de testes, pode aplicar efetivamente as

©International Software Testing Qualifications Board

habilidades de programação sem ter que se tornar um especialista de domínio em muitas áreas do negócio.

Separar o código dos dados modificáveis ajuda a isolar a automação das mudanças, melhorando a manutenção geral do código e melhorando o retorno sobre o investimento em automação.

Manuseio de falhas de software

No projeto de automação de testes, é importante antecipar e lidar com falhas de software. Se ocorrer uma falha, o automatizador de testes deve determinar o que o software de execução de testes deve fazer. A falha deve ser registrada e os testes devem continuar? Os testes devem ser encerrados? A falha pode ser tratada com uma ação específica (como clicar em um botão em uma caixa de diálogo) ou talvez adicionando um atraso no teste? Falhas desatendidas no software podem corromper os resultados dos testes subsequentes, bem como causar um problema com o teste que estava sendo executado quando a falha ocorreu.

Considerando o estado do sistema

Também é importante considerar o estado do sistema no início e no final de cada teste. Pode ser necessário assegurar que o sistema volte a um estado pré-definido após a conclusão do teste. Isto permitirá que um conjunto de testes automatizados seja executado repetidamente sem intervenção manual para repor o sistema a um estado conhecido. Para fazer isto, a automação de testes pode ter que, por exemplo, apagar os dados que criou ou alterar o status dos registros em um banco de dados. A estrutura de automação deve garantir que uma terminação adequada tenha sido realizada ao final dos testes (ou seja, o logout após a conclusão dos testes).

6.1.2 Modelagem de processos de negócios para automação

Para implementar uma abordagem baseada em palavras-chave para automação de testes, os processos de negócio a serem testados devem ser modelados na linguagem de palavras-chave de alto nível. É importante que a linguagem seja intuitiva para seus usuários que provavelmente serão os Analistas de Teste trabalhando no projeto ou, no caso de desenvolvimento ágil de software, o representante do negócio (p. ex., o Product Owner).

As palavras-chave são geralmente usadas para representar interações de negócio de alto nível com um sistema. Por exemplo, "Cancel_Order" pode exigir a verificação da existência do pedido, a verificação dos direitos de acesso da pessoa que solicita o cancelamento, a exibição do pedido a ser cancelado e o pedido de confirmação do cancelamento. Sequências de palavras-chave (p. ex., "Login", "Select_Order", "Cancel_Order"), e os dados de teste relevantes são usados pelo Analista de Testes para especificar os casos de teste.

As questões a considerar incluem o seguinte:

- Quanto mais detalhadas as palavras-chave, mais específicos são os cenários que podem ser cobertos, mas a linguagem de alto nível pode se tornar mais complexa de manter.
- Permitir que os Analistas de Testes especifiquem ações de baixo nível ("ClickButton", "SelectFromList" etc.) torna os testes de palavras-chave muito mais capazes de lidar com diferentes situações. Entretanto, como essas ações estão diretamente ligadas à GUI, também pode fazer com que os testes exijam mais manutenção quando ocorrerem mudanças.

©International Software Testing Qualifications Board

- O uso de palavras-chave agregadas pode simplificar o desenvolvimento, mas complicar a manutenção. Por exemplo, pode haver seis palavras-chave que, coletivamente, criam um registro. Entretanto, a criação de uma palavra-chave de alto nível que chame todas as seis palavras-chave pode não ser a abordagem mais eficiente em geral.
- Não importa o quanto a análise vá para a linguagem da palavra-chave, muitas vezes haverá momentos em que palavras-chave novas e alteradas serão necessárias. Há dois domínios separados para uma palavra-chave (ou seja, a lógica do negócio por trás dela e a funcionalidade de automação para executá-la). Portanto, deve ser criado um processo para lidar com ambos os domínios.

A automação de testes baseada em palavras-chave pode reduzir significativamente os custos de manutenção da automação de testes. Pode ser mais dispendioso estabelecer-se inicialmente, mas é provável que seja mais barato em geral se o projeto durar o tempo suficiente.

O syllabus *Test Automation Engineer* [CT_TAE_SYL] inclui mais detalhes sobre a modelagem de processos de negócio para automação.

6.2 Ferramentas específicas de teste

Este capítulo contém informações gerais sobre ferramentas que provavelmente serão usadas por um Analista Técnico de Testes (TTA) além do que é discutido no syllabus CTFL [CTFL_SYL].

Observe que as informações detalhadas sobre as ferramentas são fornecidas pelos seguintes ISTQB® syllabi:

- *Mobile Application Testing* [CT_MAT_SYL]
- *Performance Testing* [CT_PT_SYL]
- *Model-based Testing* [CT_MBT_SYL]
- *Test Automation Engineer* [CT_TAE_SYL]

6.2.1 Ferramentas para Plantar Falhas

Ferramentas para plantar falhas modificam o código em teste (possivelmente usando algoritmos predefinidos) para verificar a cobertura alcançada por testes específicos. Quando aplicado de forma sistemática, isto permite avaliar a qualidade dos testes (ou seja, sua capacidade de detectar os defeitos inseridos) e, quando necessário, melhorá-los.

As ferramentas para plantar falhas são geralmente usadas pelo Analista Técnico de Testes, mas também podem ser usadas pelo desenvolvedor ao testar o código recém-desenvolvido.

6.2.2 Ferramentas de Injeção de Falhas

As ferramentas de injeção de falhas fornecem deliberadamente entradas incorretas ao software para garantir que o software possa lidar com o defeito. As entradas injetadas causam condições negativas, o que deve fazer com que o manuseio do erro funcione (e seja testado). Esta interrupção do fluxo normal de execução do código também aumenta a cobertura do código.

©International Software Testing Qualifications Board

As ferramentas de injeção de falhas são geralmente usadas pelo Analista Técnico de Teste, mas também podem ser usadas pelo desenvolvedor ao testar o código recém-desenvolvido.

6.2.3 Ferramentas de Teste de Performance

As ferramentas de teste de performance têm as seguintes funções principais:

- Geração de carga
- Fornece medição, monitoramento, visualização e análise da resposta do sistema a uma determinada carga, dando uma visão do comportamento dos recursos dos componentes do sistema e da rede

A geração de carga é realizada pela implementação de um perfil operacional pré-definido (ver capítulo 4.9) como um roteiro. O script pode ser inicialmente capturado para um único usuário (possivelmente usando uma ferramenta de captura/reprodução) e é então implementado para o perfil operacional especificado usando a ferramenta de teste de performance. Esta implementação deve levar em conta a variação de dados por transação (ou conjuntos de transações).

As ferramentas de performance geram uma carga simulando muitos usuários múltiplos (usuários virtuais) seguindo seus perfis operacionais designados para realizar tarefas, incluindo a geração de volumes específicos de dados de entrada. Em comparação com scripts individuais de automação de execução de testes, muitos scripts de testes de performance reproduzem a interação do usuário com o sistema no nível do protocolo de comunicação e não simulando a interação do usuário através de uma interface gráfica de usuário. Isto geralmente reduz o número de sessões separadas necessárias durante os testes. Algumas ferramentas de geração de carga também podem conduzir a aplicação usando sua interface de usuário para medir mais de perto os tempos de resposta enquanto o sistema está sob carga.

Uma ampla gama de medições é feita por uma ferramenta de teste de performance para permitir a análise durante ou após a execução do teste. As métricas típicas tomadas e os relatórios fornecidos incluem:

- Número de usuários simulados ao longo do teste
- Número e tipo de transações geradas pelos usuários simulados e a taxa de chegada das transações
- Tempos de resposta a solicitações de transações particulares feitas pelos usuários
- Relatórios e gráficos de carga em relação aos tempos de resposta
- Relatórios sobre o uso de recursos (p. ex., uso ao longo do tempo com valores mínimos e máximos)

Entre os fatores significativos a serem considerados na implementação de ferramentas de teste de performance estão:

- O hardware e a largura de banda de rede necessários para gerar a carga útil;
- A compatibilidade da ferramenta com o protocolo de comunicação utilizado pelo sistema em teste;
- A flexibilidade da ferramenta para permitir que diferentes perfis operacionais possam ser facilmente implementados;

©International Software Testing Qualifications Board

- As instalações necessárias de monitoramento, análise e relatórios.

As ferramentas de teste de performance são normalmente adquiridas em vez de desenvolvidas internamente devido ao esforço necessário para desenvolvê-las. Entretanto, pode ser apropriado desenvolver uma ferramenta específica de performance se restrições técnicas impedirem o uso de um produto disponível, ou se o perfil de carga e as instalações a serem fornecidas forem simples em comparação com o perfil de carga e as instalações fornecidas por ferramentas comerciais. Mais detalhes das ferramentas de teste de performance são fornecidos no syllabus *Performance Testing* [CT_PT_SYL].

6.2.4 Ferramentas para testar sites da Web

Uma variedade de ferramentas especializadas de código aberto e comerciais está disponível para testes na web. A lista a seguir mostra o propósito de algumas das ferramentas comuns de teste baseadas na web:

- As ferramentas de teste Hyperlink são usadas para verificar e verificar se não há hyperlinks quebrados ou ausentes em um site
- Os verificadores HTML e XML são ferramentas que verificam a conformidade com os padrões HTML e XML das páginas que são criadas por um website
- Ferramentas de teste de performance para testar como o servidor irá reagir quando muitos usuários conectarem Ferramentas leves de execução de automação que funcionam com diferentes navegadores
- Ferramentas para escanear o código do servidor, verificando arquivos órfãos (sem link) previamente acessados pelo site
- Corretores ortográficos específicos de HTML
- Ferramentas de verificação de folhas de estilo em cascata (CSS)
- Ferramentas para verificar violações de normas, por exemplo, a Seção 508 das normas de acessibilidade nos Estados Unidos ou M/376 na Europa
- Ferramentas que encontram uma variedade de questões de segurança

A seguir estão as boas fontes de ferramentas de teste web de código aberto:

- *The World Wide Web Consortium* (W3C) [Web-3] Esta organização estabelece padrões para a Internet e fornece uma variedade de ferramentas para verificar a existência de erros em relação a esses padrões.
- *The Web Hypertext Application Technology Working Group* (WHATWG) [Web-5]. Esta organização estabelece padrões HTML. Eles têm uma ferramenta que realiza a validação HTML [Web-6].

Algumas ferramentas que incluem um motor de aranha da web também podem fornecer informações sobre o tamanho das páginas e sobre o tempo necessário para baixá-las, e sobre se uma página está presente ou não (p. ex., erro HTTP 404). Isto fornece informações úteis para o desenvolvedor, para o webmaster e para o testador.

Analistas de teste e analistas de teste técnico utilizam estas ferramentas principalmente durante os testes do sistema.

©International Software Testing Qualifications Board

6.2.5 Ferramentas para apoiar Testes Baseados em Modelos

O teste baseado em modelos (MBT) é uma técnica pela qual um modelo como uma máquina de estado finito é usado para descrever o comportamento pretendido no tempo de execução de um sistema controlado por software. Ferramentas comerciais MBT (ver [Utting07]) frequentemente fornecem um motor que permite ao usuário "executar" o modelo. As linhas de execução interessantes podem ser salvas e usadas como casos de teste. Outros modelos executáveis, como Petri Nets e Gráficos de Estado, também suportam MBT.

Os modelos MBT (e ferramentas) podem ser usados para gerar grandes conjuntos de *threads* de execução distintas. As ferramentas MBT também podem ajudar a reduzir o número muito grande de caminhos possíveis que podem ser gerados em um modelo. Os testes usando estas ferramentas podem fornecer uma visão diferente do software a ser testado. Isto pode resultar na descoberta de defeitos que podem ter sido perdidos por testes funcionais.

Mais detalhes das ferramentas de teste baseadas em modelos são fornecidos no syllabus *Model-Based Tester* [CT_MBT_SYL].

6.2.6 Teste de componentes e ferramentas de construção

Enquanto testes de componentes e ferramentas de automação de construção são ferramentas de desenvolvimento, em muitos casos, elas são usadas e mantidas por Analistas Técnicos de Teste, especialmente no contexto do desenvolvimento Ágil.

As ferramentas de teste de componentes são frequentemente específicas para a linguagem que é usada para programar um componente. Por exemplo, se Java fosse usado como a linguagem de programação, JUnit poderia ser usado para automatizar o teste do componente. Muitas outras linguagens têm suas próprias ferramentas de teste especiais; estas são chamadas coletivamente de xUnit frameworks. Tal framework gera objetos de teste para cada classe que é criada, simplificando assim as tarefas que o programador precisa fazer ao automatizar o teste do componente.

Algumas ferramentas de automação de construção permitem que uma nova construção seja acionada automaticamente quando um componente é modificado. Após a conclusão do build, outras ferramentas executam automaticamente os testes de componentes. Este nível de automação em torno do processo de construção é geralmente visto em um ambiente de integração contínua.

Quando configurado corretamente, este conjunto de ferramentas pode ter um efeito positivo sobre a qualidade das construções que estão sendo liberadas para testes. Caso uma mudança feita por um programador introduza defeitos de regressão na construção, isso normalmente causará a falha de alguns dos testes automatizados, desencadeando a investigação imediata da causa das falhas antes que a construção seja liberada para o ambiente de teste.

6.2.7 Ferramentas para apoiar Testes de Aplicações Móveis

Emuladores e simuladores são ferramentas frequentemente usadas para apoiar os testes de aplicações móveis.

©International Software Testing Qualifications Board

Simuladores

Um simulador móvel modela o ambiente de tempo de execução da plataforma móvel. As aplicações testadas em um simulador são compiladas em uma versão dedicada, que funciona no simulador, mas não em um dispositivo real. Os simuladores são usados como substitutos de dispositivos reais nos testes, mas normalmente limitam-se aos testes funcionais iniciais e simulam muitos usuários virtuais nos testes de carga. Os simuladores são relativamente simples (em comparação com os emuladores) e podem executar testes mais rapidamente do que um emulador. Entretanto, a aplicação testada em um simulador difere da aplicação que será distribuída.

Emuladores

Um emulador móvel modela o hardware e utiliza o mesmo ambiente de tempo de execução que o hardware físico. As aplicações compiladas para serem implantadas e testadas em um emulador também poderiam ser utilizadas pelo dispositivo real.

Entretanto, um emulador não pode substituir totalmente um dispositivo porque o emulador pode se comportar de uma maneira diferente do dispositivo móvel que tenta imitar. Além disso, algumas características podem não ser suportadas, tais como (multi)toque e acelerômetro. Isto é parcialmente causado por limitações da plataforma usada para rodar o emulador.

Aspectos comuns

Simuladores e emuladores são frequentemente usados para reduzir o custo de ambientes de teste através da substituição de dispositivos reais. Os simuladores e emuladores são úteis no estágio inicial de desenvolvimento, pois normalmente se integram com ambientes de desenvolvimento e permitem uma rápida implantação, teste e monitoramento de aplicações. O uso de um emulador ou simulador requer seu lançamento, instalação do aplicativo necessário sobre ele, e então testar o aplicativo como se estivesse no dispositivo real. Cada ambiente de desenvolvimento de sistema operacional móvel normalmente vem com seu próprio emulador ou simulador. Emuladores e simuladores de terceiros também estão disponíveis.

Normalmente, os emuladores e simuladores permitem o ajuste de vários parâmetros de uso. Estas configurações podem incluir emulação de rede em diferentes velocidades, força do sinal e perdas de pacotes, mudança de orientação, geração de interrupções e dados de localização GPS. Algumas destas configurações podem ser muito úteis porque podem ser difíceis ou custosas de replicar com dispositivos reais, tais como posições GPS globais ou força do sinal.

O syllabus *Mobile Application Testing* [CT_MAT_SYL] inclui mais detalhes.

©International Software Testing Qualifications Board

Referências

Normas

As seguintes normas são mencionadas nestes respectivos capítulos.

[DO-178C] DO-178C - Considerações sobre Software na Certificação de Sistemas e Equipamentos Aéreos, RTCA, 2011, Capítulo 2.

[ISO 9126] ISO/IEC 9126-1:2001, Software Engineering - Software Product Quality Chapter 4 e Anexo A.

[ISO 25010] ISO/IEC 25010: 2011, Engenharia de sistemas e software - Requisitos e avaliação da qualidade de sistemas e software (SQuaRE) - Modelos de qualidade de sistemas e software, Capítulos 1, 4 e Apêndice A.

[ISO 29119] ISO/IEC/IEEE 29119-4:2015, Engenharia de software e sistemas - Teste de software - Parte 4: Técnicas de teste, Capítulo 2.

[ISO 42010] ISO/IEC/IEEE 42010:2011, Engenharia de sistemas e software - Descrição da arquitetura, Capítulo 5.

[IEC 61508] IEC 61508-5:2010, Segurança Funcional de Elétrica/Eletrônica/Programável Sistemas Eletrônicos de Segurança, Parte 5: Exemplos de métodos para a determinação dos níveis de integridade da segurança

[ISO 26262] ISO 26262-1:2018, Veículos rodoviários - Segurança Funcional, Partes 1 a 12, Capítulo 2.

[IEC 62443-3-2] IEC 62443-3-2:2020, Segurança para automação industrial e sistemas de controle - Parte 3-2: Avaliação de risco de segurança para projeto de sistemas, Capítulo 4.

ISTQB® Documentos

[CTAL_TTA_OVIEW] ISTQB® Technical Test Analyst Advanced Level Overview v4.0

[CT_SEC_SYL] Security Testing Syllabus, Versão 2016

[CT_TAE_SYL] Test Automation Engineer Syllabus, Versão 2017

[CTFL_SYL] Foundation Level Syllabus, Versão 2018

[CT_PT_SYL] Performance Testing Syllabus, Versão 2018

[CT_MBT_SYL] Model-Based Testing Syllabus, Versão 2015

[CTAL_TM_SYL] Test Manager Syllabus, Versão 2012

[CT_MAT_SYL] Test Manager Syllabus, 2019

[ISTQB_GLOSSARY] Glossário de termos usados em testes de software, Versão 3.5, 2020

[CT_AuT_SYL] Automotive Software Tester, Versão 2018

©International Software Testing Qualifications Board

Livros e artigos

[Andrist20] Björn Andrist e Viktor Sehr, C++ Alta performance: Domine a arte de otimizar o funcionamento de seu código C++, 2ª Edição, Packt Publishing, 2020

[Beizer90] Boris Beizer, "Software Testing Techniques Second Edition", International Thomson Computer Press, 1990, ISBN 1-8503-2880-3

[Buwalda01] Hans Buwalda, "Integrated Test Design and Automation", Addison-Wesley Longman, 2001, ISBN 0-201-73725-6

[Kaner02] Cem Kaner, James Bach, Bret Pettichord; "Lessons Learned in Software Testing"; Wiley, 2002, ISBN: 0-471-08112-4

Thomas J. McCabe76] Thomas J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, dezembro de 1976, pp. 308-320

[Utting07] Mark Utting, Bruno Legeard, "Practical Model-Based Testing: A Tools Approach", Morgan-Kaufmann, 2007, ISBN: 978-0-12-372501-1

[Whittaker04] James Whittaker e Herbert Thompson, "How to Break Software Security", Pearson / Addison-Wesley, 2004, ISBN 0-321-19433-0

[Wiegers02] Karl Wiegers, "Peer Reviews in Software": A Practical Guide", Addison-Wesley, 2002, ISBN 0-201-73485-0

Outras Referencias

As seguintes referências apontam para as informações disponíveis na Internet. Mesmo que estas referências tenham sido verificadas no momento da publicação deste syllabus de nível avançado, o ISTQB® não pode ser responsabilizado se as referências não estiverem mais disponíveis.

[Web -1] http://www.nist.gov (NIST National Institute of Standards and Technology)

[Web -2] http://www.codeproject.com/KB/architecture/SWArchitectureReview.aspx

[Web -3] http://www.W3C.org

[Web -4] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[Web -5] https://whatwg.org

[Web -6] https://validator.w3.org/

[Web -7] https://dl.acm.org/doi/abs/10.1145/3340433.3342822

Capítulo 2: [Web-7]

Capítulo 4: [Web-1] [Web-4]

Capítulo 5: [Web-2]

Capítulo 6: [Web-3] [Web-5] [Web-6]

©International Software Testing Qualifications Board

Apêndice A: Visão geral das características de qualidade

A tabela a seguir compara as características de qualidade descritas na agora substituída norma ISO 9126-1 (como usada na versão 2012 do syllabus *Technical Test Analyst*) com as da nova norma ISO 25010 (como usada na última versão do syllabus).

Observe que a adequação funcional e a usabilidade são cobertas como parte do syllabus *Test Analyst*.

ISO/IEC 25010	ISO/IEC 9126-1	Notas
Adequação funcional	Funcionalidade	O novo nome é mais preciso, e evita confusão com outros significados de "funcionalidade
Completude funcional		Cobertura das necessidades declaradas
Correção funcional	Precisão	Mais geral do que a precisão
Adequação funcional	Adequação	Cobertura das necessidades implícitas
	Interoperabilidade	Movido para Compatibilidade
	Segurança	Agora uma característica
Eficiência de performance	Eficiência	Renomeado para evitar conflito com a definição de eficiência na ISO/IEC 25062
Comportamento no tempo	Comportamento no tempo	
Utilização de recursos	Utilização de recursos	
Capacidade		Nova subcaracterística
Compatibilidade		Nova característica
Coexistência	Coexistência	Movido da Portabilidade
Interoperabilidade		Movido da Funcionalidade (Analista de teste)
Usabilidade		Questão de qualidade implícita explicitada
Reconhecimento de adequabilidade	Compreensibilidade	O novo nome é mais preciso
Capacidade de aprendizagem	Capacidade de aprendizagem	
Operabilidade	Operabilidade	
Proteção de erros do usuário		Nova subcaracterística
Estética da interface do usuário	Atratividade	O novo nome é mais preciso
Acessibilidade		Nova subcaracterística
Confiabilidade	Confiabilidade	
Maturidade	Maturidade	
Disponibilidade		Nova subcaracterística
Tolerância a falhas	Tolerância a falhas	
Recuperabilidade	Recuperabilidade	

©International Software Testing Qualifications Board

ISO/IEC 25010	ISO/IEC 9126-1	Notas
Security	Segurança	Nenhuma subcaracterística anterior
Confidencialidade		Nenhuma subcaracterística anterior
Integridade		Nenhuma subcaracterística anterior
Não-repúdio		Nenhuma subcaracterística anterior
Prestação de contas		Nenhuma subcaracterística anterior
Autenticidade		Nenhuma subcaracterística anterior
Manutenção	Manutenção	
Modularidade		Nova subcaracterística
Reusabilidade		Nova subcaracterística
Analisabilidade	Analisabilidade	
Modificabilidade	Estabilidade	Nome mais preciso combinando capacidade de mudança e estabilidade
Testabilidade	Testabilidade	
Portabilidade	Portabilidade	
Adaptabilidade	Adaptabilidade	
Instalabilidade	Instalabilidade	
	Coexistência	Movido para Compatibilidade
Substituibilidade	Substituibilidade	

©International Software Testing Qualifications Board