International Software Testing Qualifications Board

Certified Tester

Foundation Level Syllabus

Versão 2011br

Comissão Internacional para Qualificação de Teste de Software

---

Tradução realizada pela TAG01 - Documentação do BSTQB baseada na versão 2011 do *Certified Tester Foundation Level Syllabus* do ISTQB.

Brazilian Software Testing Qualifications Board

Histórico de Revisões

Versão	Data	Observação
BSTQB 2011br	Agosto-2011	Ver Apendice E – Notas da Versão
BSTQB 2007br	Agosto-2007	Tradução para português do Brasil
ISTQB 2007	Maio-2007	Certified Tester Foundation Level Syllabus Maintenance Release – see Apendix E
ISTQB 2005	Julho-2005	Certified Tester Foundation Level Syllabus
ASQF V2.2	Julho-2003	ASQF Syllabus Foundation Level Version 2.2 “Lehrplan, Grundlagen des Softwaretestens“
ISEB V2.0	Fevereiro-1999	ISEB Software Testing Foundation Syllabus V2.0 25 February 1999

Índice

Agradecimentos

International Software Testing Qualifications Board Working Party Foundation Level: Thomas Muller (Diretor), Dorothy Graham, Debra Friedenberg e Erik van Veendendal. A comissão principal agradece a equipe de revisão (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Andrers Petterson e Wonil Knon) e todas as comissões nacionais para as sugestões deste syllabus.

International Software Testing Qualifications Board Working Party Foundation Level: Thomas Muller (Diretor), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson e Erik van Veendendal. A comissão principal agradece a equipe de revisão e todas as comissões nacionais para as sugestões deste syllabus.

Agradecimentos especiais para (Áustria) Anastasios Kyriakopoulos, (Dinamarca) Klaus Olsen, Christine Rosenbeck-Larsen, (Alemanha) Matthias Daigl, Uwe Hehn, Tilo Linz, Horst Pohlmann, Ina Schieferdecker, Sabine Uhde, Stephanie Ulrich, (Índia) Vipul Kocher, (Israel) Shmuel Knishinsky, Ester Zabar, (Suécia) Anders Claesson, Mattias Nordin, Ingvar Nordström, Stefan Ohlsson, Kennet Osbjer, Ingela Skytte, Klaus Zeuge, (Suíça) Armin Born, Sandra Harries, Silvio Moser, Reto Muller, Joerg Pietzsch, (Reino Unido) Aran Ebbett, Isabel Evans, Julie Gardiner, Andrew Goslin, Brian Hambling, James Lyndsay, Helen Moore, Peter Morgan, Trevor Newton, Angelina Samaroo, Shane Saunders, Mike Smith, Richard Taylor, Neil Thompson, Pete Williams, (Estados Unidos) Dale Perry.

Introdução do Syllabus

Objetivo deste documento

Este *syllabus* forma a base de conhecimento para a Qualificação Internacional de Teste de Software no nível *Foundation*. O *International Software Testing Qualifications Board* (ISTQB) disponibiliza o *syllabus* às comissões nacionais para que elas autorizem os fornecedores de treinamento e também derivem as questões do exame em suas línguas locais. Os fornecedores de treinamento produzirão o material de curso e determinarão os métodos de ensino apropriados para certificação, e o *syllabus* ajudará os candidatos em sua preparação para o exame. Informações históricas e conceituais do *syllabus* podem ser encontradas no apêndice A.

CTFL (Certified Tester Foundation Level)

A qualificação CTFL (*Certified Tester Foundation Level*) é voltada para qualquer pessoa envolvida em teste de software. Isto inclui pessoas em funções específicas de teste como: testadores, analistas, engenheiros, consultores, gerentes, usuários que realizam teste de aceite e desenvolvedores de software. Este nível de qualificação é também apropriado para qualquer profissional que queira adquirir uma base de compreensão sobre teste de software, como gerentes de projetos, gerentes de qualidade, gerentes de desenvolvimento de software, analistas de negócios, diretores de TI e consultores. Aqueles que alcançarem a Certificação estarão aptos a buscar um nível mais alto de qualificação em teste de software.

Objetivos de aprendizagem / níveis de conhecimento

Os seguintes níveis cognitivos são considerados para cada sessão neste *syllabus*:

- **K1**: relembrar.
- **K2**: entender.
- **K3**: aplicar.
- **K4**: analisar.

Maiores detalhes e exemplos dos objetivos de estudo são dados no Apêndice B.

Todos os termos listados abaixo do tópico "***Termos***" devem ser relembrados (**K1**), mesmo que não forem explicitamente mencionados nos objetivos de estudo.

O Exame

O exame CTFL será baseado neste *syllabus*. Respostas para as questões do exame podem requerer o uso do material baseado em mais de uma sessão do *syllabus*. Todas as sessões do *syllabus* poderão ser contempladas no exame.

O exame é composto por questões de múltipla escolha.

Exames podem ser efetuados como parte de um treinamento certificado ou não. Um treinamento realizado por uma entidade certificada pelo BSTQB não é pré-requisito para a participação de um exame.

Autorização

Provedores de treinamentos que utilizam o *syllabus* como referência em seus cursos podem ser autorizados por uma comissão nacional (*board*) reconhecida pelo ISTQB. Os procedimentos de autorização podem ser obtidos a partir de uma comissão (*board*) ou grupo que realiza a autorização. Um curso autorizado é reconhecido em conformidade com este *syllabus*, sendo permitida a realização de um exame do ISTQB como parte do curso.

Maiores detalhes para os fornecedores de treinamento são dados no Apêndice D.

Nível de Detalhe

O nível de detalhe do *syllabus* permite que o treinamento e o exame sejam feitos internacionalmente e de forma consistente. Com foco em alcançar estes objetivos, o *syllabus* consiste de:

- Objetivos de instrução geral descrevendo as intenções do *syllabus*.
- Uma lista de informações para o treinamento, incluindo uma descrição e referências a fontes adicionais, se necessária.
- Objetivos de estudos para cada área de conhecimento, descrevendo os resultados do conhecimento aprendido e das metas a serem atingidas.
- Uma lista de termos que os estudantes precisam estar aptos a relembrar e compreender.
- Uma explicação dos conceitos principais a serem ensinados, incluindo as fontes, como a literatura aceita ou padrões.

O conteúdo do *syllabus* não é uma descrição completa da área de conhecimento de teste de software; ele reflete o nível de detalhe coberto no treinamento para o nível fundamental.

Como este syllabus está organizado

Há seis capítulos principais. O título no nível mais alto é acompanhado pelos níveis de aprendizagem cobertos pelo capítulo e pelo tempo de estudo para cada capítulo. Por exemplo:

2. Teste durante o ciclo de vida do software (K2)	**115 minutos**

Mostra que o capítulo 2 tem objetivos de estudos de K1 (assumido quando um nível maior é demonstrado) e K2 (mas não K3) e está dimensionado para levar 115 minutos para cobrir o estudo do capítulo.

Cada capítulo é composto de seções. Cada seção também tem os objetivos de aprendizagem e o tempo necessário para estudo. Subseções que não têm um tempo determinado são incluídas no tempo da seção.

1. Fundamentos do Teste (K2)	155 minutos

Objetivos de estudo para os fundamentos do teste

Os objetivos identificam o que você estará apto a fazer após o término de cada módulo.

1.1 Porque é necessário testar? (K2)

LO-1.1.1 Descrever com exemplos, a maneira com que o defeito no software pode causar danos a pessoas, companhias ou ambientes. (K2)

LO-1.1.2 Distinguir entre a causa raiz do defeito e seus efeitos. (K2)

LO-1.1.3 Justificar a necessidade de testar utilizando exemplos. (K2)

LO-1.1.4 Descrever porque teste é parte da garantia da qualidade e dar exemplos de como o teste contribui para atingir um nível de qualidade superior. (K2)

LO-1.1.5 Explicar e comparar termos como erro, defeito, dano, falha e seus termos correspondentes engano *e bug* usando exemplos. (K2)

1.2 O que é teste? (K2)

LO-1.2.1 Recordar os objetivos comuns do teste. (K1)

LO-1.2.2 Exemplificar os objetivos do teste em diferentes fases do ciclo de vida de um software. (K2)

LO-1.2.3 Diferenciar teste de depuração de código (K2)

1.3 Princípios gerais do teste (K2)

LO-1.3.1 Explicar os sete princípios do teste. (K2)

1.4 Fundamentos do processo de teste (K1)

LO-1.4.1 Recordar as cinco atividades fundamentais de teste e suas respectivas no planejamento do teste. (K1)

1.5 A psicologia do teste (K2)

LO-1.5.1 Recordar que o sucesso do teste é influenciado por fatores psicológicos (K1)

LO-1.5.2 Diferenciar a forma de pensar dos testadores e dos desenvolvedores. (K2)

1.1 Porque é necessário testar? (K2)	20 minutos

Termos

Bug, defeito, erro, falha, dano, engano, qualidade, risco.

1.1.1 Contexto dos sistemas de software (K1)

Sistemas de software tornam-se cada vez mais parte do nosso dia-a-dia, desde aplicações comerciais (ex.: bancos) até produtos de consumo (ex.: carros). A maioria das pessoas já teve alguma experiência com um software que não funcionou como esperado. Softwares que não funcionam corretamente podem levar a muitos problemas, incluindo financeiro, tempo e reputação das empresas. Podendo, inclusive, chegar a influenciar na integridade das pessoas.

1.1.2 Causas dos defeitos de software (K2)

O ser humano está sujeito a cometer um **erro (engano)**, que produz um **defeito (falha, *bug*)**, no código, em um software ou sistema ou em um documento. Se um defeito no código for executado, o sistema falhará ao tentar fazer o que deveria (ou, em algumas vezes, o que não deveria), causando uma falha. Defeitos no software, sistemas ou documentos resultam em falhas, mas nem todos os defeitos causam falhas.

Os **defeitos** ocorrem porque os seres humanos são passíveis de falha e porque existe pressão no prazo, códigos complexos, complexidade na infraestrutura, mudanças na tecnologia e/ou muitas interações de sistema.

Falhas também podem ser causadas por condições do ambiente tais como: radiação, magnetismo, campos eletrônicos e poluição, que podem causar falhas em software embarcado (firmware) ou influenciar a execução do software pela mudança das condições de hardware.

1.1.3 Função do teste no desenvolvimento, manutenção e operação de software (K2).

Rigorosos testes em sistemas e documentações podem reduzir os riscos de ocorrência de problemas no ambiente operacional, e contribui para a qualidade dos sistemas de software se os defeitos encontrados forem corrigidos antes de implantados em produção.

O teste de software pode também ser necessário para atender requisitos contratuais ou legais ou determinados padrões de mercado.

1.1.4 Teste e qualidade (K2)

Com a ajuda do teste é possível medir a qualidade do software em termos de defeitos encontrados, por características e requisitos funcionais ou não funcionais do software (confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade). Para mais informações sobre testes não funcionais veja o Capitulo 2. Para mais informações sobre características do software, veja "*Software Engineering – Software Product Quality*" (ISO 9126).

O resultado da execução dos testes pode representar confiança na qualidade do software caso sejam encontrados poucos ou nenhum defeito. Um teste projetado adequadamente e cuja execução não encontra defeitos reduz o nível de riscos em um sistema. Por outro lado, quando os testes encontram defeitos, a qualidade do sistema aumenta quando estes são corrigidos.

Projetos anteriores devem prover lições aprendidas. Através do entendimento da causa raiz dos defeitos encontrados em outros projetos, os processos podem ser aprimorados de modo a prevenir reincidência de erros e, consequentemente, melhorar a qualidade dos sistemas futuros.

Testes devem ser integrados como uma das atividades de garantia da qualidade (ex.: juntamente aos padrões de desenvolvimento, treinamento e análise de defeitos).

1.1.5 Quanto teste é suficiente? (K2)

Para decidir quanto teste é suficiente, deve-se levar em consideração o nível do risco, incluindo risco técnico, do negócio e do projeto, além das restrições do projeto como tempo e orçamento. (Risco será discutido com mais detalhes no Capítulo 5)

O teste deve prover informações suficientes aos interessados (*stakeholders*) para tomada de decisão sobre a distribuição do software ou sistema, para as próximas fases do desenvolvimento ou implantação nos clientes.

1.2 O que é teste? (K2)	30 minutos

Termos

Depuração de código, requisito, revisão, caso de teste, objetivo do teste.

Conceito

Uma visão comum do processo de teste é de que ele consiste apenas da fase de execução, como executar o programa. Esta, na verdade, é uma parte do teste, mas não contempla todas as atividades do teste.

Existem atividades de teste antes e depois da fase de execução. Por exemplo: planejamento e controle, escolha das condições de teste, modelagem dos casos de teste, checagem dos resultados, avaliação do critério de conclusão, geração de relatórios sobre o processo de teste e sobre sistema alvo e encerramento ou conclusão (ex.: após a finalização de uma fase de teste). Teste também inclui revisão de documentos (incluindo o código fonte) e análise estática.

Testes dinâmicos e estáticos podem ser usados para atingir objetivos similares e proveem informações para melhorar o sistema a ser testado e o próprio processo de teste.

Testes podem possuir objetivos diferentes:

- Encontrar defeitos.
- Ganhar confiança sobre o nível de qualidade
- Prover informações para tomada de decisão.
- Prevenir defeitos.

O processo mental de projetar testes de forma antecipada no ciclo de vida (verificando a base de teste através da modelagem de teste) pode ajudar a prevenir defeitos que poderiam ser introduzidos no código. A revisão de documentos (ex.: requisitos) também ajuda a prevenir defeitos que possam aparecem no código.

No processo de teste, diferentes pontos de vista levam a diferentes objetivos. Por exemplo, no teste feito em desenvolvimento (teste de componente, integração e de sistemas), o principal objetivo pode ser causar o maior número de falhas possíveis, de modo que os defeitos no software possam ser identificados e resolvidos. No teste de aceite o objetivo principal pode ser confirmar se o sistema está funcionando conforme o esperado, ou seja, prover a confiabilidade de que esteja de acordo com o requisito. Em alguns casos o principal objetivo do teste pode ser avaliar a qualidade do software (não com a intenção de encontrar defeitos), para prover informações sobre os riscos da implantação do sistema em um determinado momento aos gestores. Os testes de manutenção podem ser usados para verificar se não foram inseridos erros durante o desenvolvimento de mudanças. Durante os testes operacionais, o principal objetivo pode ser avaliar características como confiabilidade e disponibilidade.

Depuração de código e teste são atividades diferentes. Testes podem demonstrar falhas que são causadas por defeitos. Depuração de código é a atividade de desenvolvimento que identifica a causa de um defeito, repara o código e checa se os defeitos foram corrigidos corretamente. Depois é feito um teste de confirmação por um testador para certificar se a falha foi eliminada. As responsabilidades de cada atividade são bem distintas: testadores testam e desenvolvedores depuram.

O processo de teste e suas atividades são explicados na seção 1.4.

1.3 Os sete Princípios do Teste (K2) — 35 minutos

Termos

Teste exaustivo

Princípios

Alguns princípios foram sugeridos ao longo dos últimos 40 anos, oferecendo um guia geral para o processo de teste como um todo.

Princípio 1 – Teste demonstra a presença de defeitos

O teste pode demonstrar a presença de defeitos, mas não pode provar que eles não existem. O Teste reduz a probabilidade que os defeitos permaneçam em um software, mas mesmo se nenhum defeito for encontrado, não prova que ele esteja perfeito.

Princípio 2 – Teste exaustivo é impossível

Testar tudo (todas as combinações de entradas e pré-condições) não é viável, exceto para casos triviais. Em vez do teste exaustivo, riscos e prioridades são levados em consideração para dar foco aos esforços de teste.

Princípio 3 – Teste antecipado

A atividade de teste deve começar o mais breve possível no ciclo de desenvolvimento do software ou sistema e deve ser focado em objetivos definidos.

Princípio 4 – Agrupamento de defeitos

Um número pequeno de módulos contém a maioria dos defeitos descobertos durante o teste antes de sua entrega ou exibe a maioria das falhas operacionais.

Princípio 5 – Paradoxo do Pesticida

Pode ocorrer de um mesmo conjunto de testes que são repetidos várias vezes não encontrarem novos defeitos após um determinado momento. Para superar este "paradoxo do pesticida", os casos de testes necessitam ser frequentemente revisado e atualizado. Um conjunto de testes novo e diferente precisa ser escrito para exercitar diferentes partes do software ou sistema com objetivo de aumentar a possibilidade de encontrar mais erros.

Princípio 6 – Teste depende do contexto

Testes são realizados de forma diferente conforme o contexto. Por exemplo, softwares de segurança crítica são testados diferentemente de um software de comércio eletrônico.

Princípio 7 – A ilusão da ausência de erros

Encontrar e consertar defeitos não ajuda se o sistema construído não atende às expectativas e necessidades dos usuários.

1.4 Fundamentos do Processo de Teste (K1)	35 minutos

Termos

Teste de confirmação, reteste, critério de saída, incidente, teste de regressão, base de teste, condição de teste, cobertura de teste, dados de teste, execução de teste, registro de teste, plano de teste, estratégia de teste, política de teste, suíte de teste, relatório consolidado de teste, *testware*.

Conceito

A parte mais visível do teste é a execução. Mas para se obter eficácia e eficiência, os planos de teste precisam conter o tempo a ser gasto no planejamento dos testes, modelagem dos casos de testes e preparação da execução e avaliação de resultados.

O processo de teste básico consiste das seguintes atividades:

- Planejamento e controle
- Análise e modelagem
- Implementação e execução
- Avaliação dos critérios de saída e relatórios
- Atividades de encerramento de teste

Apesar de serem apresentadas sequencialmente, as atividades durante o processo podem sobrepor-se ou acontecer de forma concorrente. Adaptando essas atividades principais dentro do contexto do sistema e do projeto quando necessário.

1.4.1 Planejamento e controle do teste (K1)

O planejamento de teste é a atividade que consiste em definir os objetivos e especificar as atividades de forma a alcançá-los.

O controle do teste é a constante atividade que consiste em comparar o progresso atual contra o que foi planejado, reportando o status e os desvios do plano. Ele envolve ainda a tomada de ações necessárias para alcançar a missão e objetivos do projeto. Para um controle efetivo, o teste deverá ser monitorado durante todo o projeto. O planejamento do teste leva em consideração o retorno de informações das atividades de monitoração e controle.

As tarefas de Planejamento e Controle estão definidas no Capítulo 5

1.4.2 Análise e modelagem do Teste (K1)

A análise e a modelagem de teste são atividades onde os objetivos gerais do teste são transformados em condições e modelos de teste tangíveis.

A análise e a modelagem de teste são compostas pelas seguintes atividades principais:

- Revisar a base de testes (como requisitos, nível de integridade do software[1] (nível de risco), arquitetura, modelagem, interfaces).
- Avaliar a testabilidade dos requisitos e do sistema.

[1] *Grau em que o software cumpre ou deve cumprir um conjunto e / ou características de sistema baseadas em software (por exemplo: a complexidade do software, avaliação de risco, o nível de segurança, desempenho, confiabilidade ou custo) que são definidos para refletir a importância do software para seus stakeholders.*

- Identificar e priorizar as condições ou requisitos de testes e dados de testes baseados na análise dos itens de teste, na especificação, no comportamento e na estrutura.
- Projetar e priorizar os casos de testes de alto nível.
- Identificar as necessidades de dados para teste suportando as condições e casos de teste
- Planejar a preparação do ambiente de teste e identificar a infraestrutura e ferramentas necessárias.
- Criar uma rastreabilidade bidirecional entre os requisitos e os casos de teste.

1.4.3 Implementação e execução de teste (K1)

A implementação e execução do teste é a atividade onde os procedimentos ou os scripts de teste são especificados pela combinação dos casos de teste em uma ordem particular, incluindo todas as outras informações necessárias para a execução do teste, o ambiente é preparado e os testes são executados.

A implementação e execução de teste são compostas pelas seguintes atividades principais:

- Finalizar, implementar e priorizar os casos de teste (incluindo a identificação dos dados para teste).
- Desenvolver e priorizar os procedimentos de teste, criar dados de teste e, opcionalmente, preparar o ambiente para teste e os scripts de testes automatizados.
- Criar suítes de teste a partir dos casos de teste para uma execução de teste eficiente.
- Verificar se o ambiente está preparado corretamente.
- Verificar e atualizar a rastreabilidade bidirecional entre a base de teste e os casos de teste.
- Executar os casos de teste manualmente ou utilizando ferramentas de acordo com a sequência planejada.
- Registrar os resultados da execução do teste e anotar as características e versões do software em teste, ferramenta de teste e *testware*.
- Comparar resultados obtidos com os resultados esperados.
- Reportar as discrepâncias como incidentes e analisá-los a fim de estabelecer suas causas (por exemplo, defeito no código, em algum dado específico de teste, na documentação de teste ou uma execução de inadequada do teste).
- Repetir os testes como resultado de ações tomadas para cada discrepância. Por exemplo, re-execução de um teste que falhou previamente quando da confirmação de uma correção (teste de confirmação), execução de um teste corrigido e/ou execução de testes a fim de certificar que os defeitos não foram introduzidos em áreas do software que não foram modificadas, ou que a correção do defeito não desvendou outros defeitos (teste de regressão).

1.4.4 Avaliação do critério de saída e relatório (K1)

Avaliação do critério de saída é a atividade onde a execução do teste é avaliada mediante os objetivos definidos. Deve ser feito para cada nível de teste (ver seção 2.2)

A avaliação do critério de saída é composta pelas seguintes atividades principais:

- Checar os registros de teste (*logs*) mediante o critério de encerramento especificado no planejamento de teste.
- Avaliar se são necessários testes adicionais ou se o critério de saída especificado deve ser alterado.
- Elaborar um relatório de teste resumido para os interessados (*stakeholders)*.

1.4.5 Atividades de encerramento de teste (K1)

Na atividade de encerramento de teste são coletados os dados de todas as atividades para consolidar a experiência, *testware*, fatos e números. Por exemplo, quando um software é lançado, um projeto de teste é completado (ou cancelado), um marco do projeto foi alcançado, ou a implantação de um demanda de manutenção foi completada.

As atividades de encerramento de teste são compostas pelas seguintes atividades principais:

- Checar quais entregáveis planejados foram realmente entregues
- Fechar os relatórios de incidentes, ou levantar os registros de mudança que permaneceram abertos.
- Documentar o aceite do sistema.
- Finalizar e arquivar o *testware*, o ambiente de teste e infraestrutura de teste para o reuso.
- Entregar o *testware* para a manutenção da organização.
- Analisar as lições aprendidas para se determinar as mudanças necessárias para futuros *releases* e projetos.
- Utilizar as informações coletadas para melhorar a maturidade de teste

1.5 A Psicologia do Teste (K2)	25 minutos

Termos

Suposição de erro, Teste independente

Conceito

A forma de pensar utilizada enquanto se está testando e revisando é diferente da utilizada enquanto se está analisando e desenvolvendo. Com a sua forma de pensar, os desenvolvedores estão aptos a testarem seus próprios códigos, mas a separação desta responsabilidade para um testador é tipicamente feita para ajudar a focalizar o esforço e prover benefícios adicionais, como uma visão independente, profissional e treinada de recursos de teste. Teste independente pode ser considerado em qualquer nível de teste.

Certo grau de independência (evitando a influência do autor) muitas vezes representa uma forma eficiente de encontrar defeitos e falhas. Independência não significa simplesmente uma substituição, tendo em vista que os desenvolvedores podem encontrar defeitos no código de maneira eficiente. Níveis de independência podem ser definidos como:

- Teste elaborado por quem escreveu o software que será testado (baixo nível de independência).
- Teste elaborado por outra(s) pessoa(s) (por exemplo, da equipe de desenvolvimento).
- Teste elaborado por pessoa(s) de um grupo organizacional diferente (ex.: equipe independente de teste).
- Teste elaborado por pessoa(s) de diferentes organizações ou empresas (terceirizada ou certificada por um órgão externo).

Pessoas e projetos são direcionados por objetivos. Pessoas tendem a alinhar seus planos com os objetivos da gerência e outros envolvidos ("*stakeholders*") para, por exemplo, encontrar defeitos ou confirmar que o software funciona. Desta forma, é importante ter objetivos claros do teste.

Identificar falhas durante o teste pode ser considerado uma crítica contra o produto e o autor (responsável pelo produto). Teste é, nestes casos, visto como uma atividade destrutiva, apesar de ser construtiva para o gerenciamento do risco do produto. Procurar por falhas em um sistema requer curiosidade, pessimismo profissional, um olhar crítico, atenção ao detalhe, comunicação eficiente com os profissionais do desenvolvimento e experiência para encontrar erros.

Se os erros, defeitos ou falhas são comunicados de uma forma construtiva, podem-se evitar constrangimentos entre as equipes de teste, analistas e desenvolvedores, tanto na revisão quanto no teste.

O testador e o líder da equipe de teste precisam ter boa relação com as pessoas para comunicar informações sólidas sobre os defeitos, progresso e riscos de uma forma construtiva. A informação do defeito pode ajudar o autor do software ou documento a ampliar seus conhecimentos. Defeitos encontrados e resolvidos durante o teste trará ganho de tempo e dinheiro, além de reduzir os riscos.

Problemas de comunicação podem ocorrer, especialmente se os testadores forem vistos somente como mensageiros de más notícias ao informar os defeitos. De qualquer forma, existem formas de melhorar a comunicação e o relacionamento entre os testadores e os demais:

- Começar com o espírito de colaboração, ao invés de disputa (conflitos), onde todos têm o mesmo objetivo para alcançar a melhor qualidade do sistema.
- Comunicar os erros encontrados nos produtos de uma forma neutra, dar foco no fato sem criticar a pessoa que o criou, por exemplo, escrevendo objetivamente o relatório de incidentes.
- Tentar compreender como a pessoa se sente ao receber a notícia e interpretar sua reação.
- Confirmar que a outra pessoa compreendeu o que você relatou e vice-versa.

1.6 Código de Ética	10 minutos

O envolvimento em teste de software permite que pessoas conheçam informações confidenciais e privilegiadas. Um código de ética é necessário, entre outros motivos, para garantir que a informação não é usada de forma inapropriada. Reconhecendo o código de ética para engenheiros da ACM e IEEE, o ISTQB® estabelece o seguinte código de ética:

PÚBLICO – Testadores certificados devem atuar consistentemente com o interesse público.

CLIENTE E EMPREGADOR – Testadores certificados devem agir da melhor forma para os interesses de seus clientes e empregadores, consistente com o interesse público.

PRODUTO – Testadores certificados devem garantir que os entregáveis que eles fornecem (produtos e sistemas que eles testam) correspondem aos mais altos padrões profissionais possíveis.

JULGAMENTO – Testadores certificados devem manter integridade e independência em seu julgamento profissional.

GERENCIAMENTO – Gerentes e líderes de teste certificados devem se submeter e promover uma abordagem ética ao gerenciamento do teste de software.

PROFISSÃO – Testadores certificados devem promover a integridade e reputação da profissão, consistentemente com o interesse público.

COLEGAS – Testadores certificados devem ser agradáveis e incentivadores com seus colegas, e promoverem a cooperação com os desenvolvedores de software.

INDIVÍDUO – Testadores certificados devem praticar um aprendizado vitalício em consideração à prática de sua profissão e devem promover uma abordagem ética nessa prática.

Referências

1.1.5 Black, 2001, Kaner, 2002
1.2 Beizer, 1990, Black, 2001, Myers, 1979
1.3 Beizer, 1990, Hetzel, 1998, Myers, 1979
1.4 Hetzel, 1998
1.4.5 Black, 2001, Craig, 2002
1.5 Black, 2001, Hetzel, 1998

2. Teste durante o ciclo de vida do software (K2) — 115 minutos

Objetivos de estudo para o teste durante o ciclo de vida do software

Os objetivos identificam o que você será capaz de fazer após a finalização de cada módulo.

2.1 Modelos de desenvolvimento de software (K2)

LO-2.1.1 Compreender as relações entre o desenvolvimento, atividades de teste e produtos de trabalho durante o ciclo de desenvolvimento, dando exemplos baseados em projetos e características de produtos e contexto (K2).

LO-2.1.2 Reconhecer o fato do modelo de desenvolvimento de software precisar ser adaptado ao contexto do projeto e às características do produto. (K1)

LO-2.1.3 Rever as características de bons testes em qualquer modelo de ciclo de vida (K1).

2.2 Níveis de Teste (K2)

LO-2.2.1 Comparar os diferentes níveis de teste: principais objetivos, objetos típicos de teste, alvos de teste (ex.: funcional ou estrutural), produtos de trabalho, pessoas que testam tipos de defeitos e falhas a serem identificadas. (K2)

2.3 Tipos de teste: os alvos do teste (K2)

LO-2.3.1 Comparar quatro tipos de teste de software (funcional, não funcional, estrutural e relativo à mudança) através de exemplos. (K2)

LO-2.3.2 Reconhecer que os testes funcionais e estruturais ocorrem em qualquer nível de teste. (K1)

LO-2.3.3 Identificar e descrever o tipo de teste não funcional baseado em requisito não funcional. (K2)

LO-2.3.4 Identificar e descrever os tipos de teste baseados na análise da estrutura ou arquitetura do sistema. (K2)

LO-2.3.5 Descrever o propósito do teste de confirmação e regressão. (K2)

2.4 Teste de Manutenção (K2)

LO-2.4.1 Comparar teste de manutenção (testar um sistema existente) com o teste em uma nova aplicação segundo os critérios: tipo de teste, disparadores de testes (*triggers*) e quantidade de teste. (K2).

LO-2.4.2 Identificar as razões teste de manutenção (modificação, migração e retirada). (K1)

LO-2.4.3 Descrever as funções do teste de regressão e da análise de impacto na manutenção. (K2)

2.1 Modelos de Desenvolvimento de Software (K2)	35 minutos

Termos

Software comercial de prateleira (COTS - *"Commercial Off the Shelf"*), modelo de desenvolvimento incremental, validação, verificação, Modelo V.

Conceito

Não existe teste isolado; a atividade de teste está intimamente relacionada com as atividades de desenvolvimento do software. Modelos de ciclo de vida de desenvolvimento diferentes necessitam de abordagens diferentes para testar.

2.1.1 Modelo V (K2)

Apesar das variações do Modelo V, um tipo comum deste modelo usa quatro níveis de teste correspondentes a quatro níveis de desenvolvimento.

Os quatro níveis usados no *syllabus* são:

- Teste de Componente (unidade);
- Teste de Integração;
- Teste de Sistema;
- Teste de Aceite;

Na prática, um Modelo V, pode ter mais, menos ou diferentes níveis de desenvolvimento e teste, dependendo do projeto e do produto. Por exemplo, pode haver teste de integração de componentes após o teste de um componente, e teste de integração de sistemas após o teste de sistemas.

Produtos de trabalho de software (como cenário de negócios ou casos de uso, especificação de requisitos, documentos de modelagem e código) produzidos durante o desenvolvimento muitas vezes são a base do teste em um ou mais nível de teste. Alguns exemplos de referências para produtos de trabalho genéricos: CMMI (*Capability Maturity Model Integration*) ou os "*Software life cycle processes*" IEEE/IEC 12207. Verificação e validação (e modelagem antecipada de teste) podem ser executadas durante a elaboração destes produtos de trabalho.

2.1.2 Modelos iterativos de desenvolvimento (K2)

Desenvolvimento iterativo é o processo que estabelece os requisitos, modelagem, construção e teste de um sistema, realizada como uma série de desenvolvimentos menores. Exemplos desenvolvimento iterativo: prototipagem, desenvolvimento rápido de aplicação (RAD), *Rational Unified Process* (RUP) e modelos ágeis de desenvolvimento. O produto resultante de uma iteração pode ser testado em vários níveis como parte do seu desenvolvimento. Um incremento, adicionado a outros desenvolvidos previamente, forma um sistema parcial em crescimento, que também deve se testado. Teste de regressão tem sua importância aumentada a cada iteração. Verificação e validação podem ser efetuadas a cada incremento.

2.1.3 Teste dentro de um modelo de ciclo de vida (K2)

Os itens abaixo indicam as características para um bom teste para qualquer modelo de ciclo de vida:

- Para todas as atividades do desenvolvimento há uma atividade de teste correspondente.
- Cada nível de teste tem um objetivo específico daquele nível.

- A análise e modelagem do teste para um dado nível de teste devem começar durante a atividade de desenvolvimento correspondente.
- Testadores devem se envolver na revisão de documentos o mais cedo possível utilizando as primeiras versões disponíveis ao longo ciclo de desenvolvimento.

Níveis de teste podem ser combinados ou reorganizados dependendo da natureza do projeto ou arquitetura do sistema. Por exemplo, para a integração de um pacote *(COTS)*, em um sistema, o cliente pode fazer o teste de integração em nível de sistema (ex.: integração da infraestrutura ou outros sistemas, implantação do sistema) e teste de aceite (funcional e/ou não funcional, teste de usuário e/ou operacional).

2.2 Níveis de Teste (K2)	60 minutos

Termos

Alfa Teste, Beta Teste, teste de componente (também conhecido como teste de unidade, módulo ou teste de programa), controlador ("*driver"*), teste no campo, requisitos funcionais, integração, teste de integração, requisitos não funcionais, testes de robustez, simulador ("*stub"*), teste de sistema, nível de teste, desenvolvimento dirigido à teste, ambientes de teste, teste de aceite do usuário.

Conceito

Para cada nível de teste, os seguintes aspectos podem ser identificados: seus objetivos genéricos, os produtos de trabalho utilizados como referência para derivar os casos de testes (ex.: base do teste), o objeto do teste (o que está sendo testado), defeitos e falhas típicas a se encontrar, testes ("*harness*") e ferramentas de suporte e abordagens e responsabilidades específicas.

2.2.1 Teste de Componente (K2)

Teste de componentes procura defeitos e verifica o funcionamento do software (ex.: módulos, programas, objetos, classes, etc.) que são testáveis separadamente. Pode ser feito isolado do resto do sistema, dependendo do contexto do ciclo de desenvolvimento e do sistema. Controladores ("*drivers*") e simuladores ("*stubs*") podem ser usados.

Teste de componente pode incluir teste de funcionalidade e características específicas não funcionais tais como comportamento dos recursos (ex.: falta de memória) e testes de robustez, além de teste estrutural (cobertura de código). Casos de teste são derivados dos produtos de trabalho como, por exemplo, especificação de componente, modelagem do software ou modelo de dados.

Tipicamente, teste de componente ocorre com acesso ao código que está sendo testado e no ambiente de desenvolvimento, assim como um *framework* de teste de unidade ou ferramenta de depuração "*debugging"*. Na prática, envolve o programador do código. Defeitos são normalmente corrigidos assim que são encontrados sem registrar formalmente os incidentes.

Uma abordagem no teste de componente consiste em preparar e automatizar os casos de testes antes de codificar. Isto é chamado de abordagem de teste antecipado ou desenvolvimento dirigido a teste. Esta abordagem é essencialmente iterativa e é baseada em ciclos de elaboração de casos de testes. À medida que são construídas e integradas pequenas partes do código, são executados testes de componente até que eles passem.

2.2.2 Teste de Integração (K2)

Teste de integração é caracterizado por testar as interfaces entre os componentes, interações de diferentes partes de um sistema, como o sistema operacional, arquivos, hardware ou interfaces entre os sistemas.

Pode haver mais que um nível de teste de integração, que pode ser utilizado em objetos de teste de tamanho variado. Por exemplo:

- Teste de integração de componente testa interações entre componentes de software e é realizado após o teste de componente.
- Teste de integração de sistemas testa interação entre diferentes sistemas e pode ser realizado após o teste de sistema. Neste caso a área de desenvolvimento pode controlar apenas um lado da interface, de forma que mudanças podem causar instabilidades. Processos de negócios implementados como fluxogramas

podem envolver uma série de sistemas. Problemas relacionados a múltiplas plataformas podem ser significativos.

Quanto maior o escopo da integração, maior a dificuldade de isolar as falhas para componentes ou sistemas específicos, fato que pode representar um aumento no risco.

Estratégias sistemáticas de integração podem ser baseadas na arquitetura do sistema (*top-down* e *bottom-up*), funções, sequências de processamento de transações, entre outros aspectos do sistema ou componente. Visando reduzir o risco de encontrar defeitos tardiamente, a integração deve, preferencialmente, ser incremental e não "*big bang*".

Teste de características não funcionais específicas (por exemplo, performance) pode ser incluído nos testes de integração.

A cada estágio da integração, os testadores concentram somente na integração propriamente. Por exemplo, o módulo A está sendo integrado com o módulo B o foco é a comunicação entre os módulos, não suas funcionalidades. Tanto testes funcionais quanto estruturais podem ser utilizados.

Idealmente, os testadores devem compreender a arquitetura e influenciar no planejamento da integração. Se o teste de integração for planejado antes que os componentes ou sistemas estejam prontos, eles podem ser preparados visando um teste mais eficiente.

2.2.3 Teste de Sistema (K2)

Teste de sistema se refere ao comportamento de todo do sistema / produto definido pelo escopo de um projeto ou programa de desenvolvimento.

No teste de sistema, o ambiente de teste deve corresponder o máximo possível ao objetivo final, ou o ambiente de produção, para minimizar que os riscos de falhas específicas de ambiente não serem encontradas durante o teste.

Testes de sistemas podem ser baseados em especificação de riscos e/ou de requisitos, processos de negócios, casos de uso, dentre outras descrições de alto nível do comportamento, interações e recursos do sistema.

Teste de sistema deve tratar requisitos funcionais e não funcionais do sistema. Os requisitos podem estar como um texto ou diagramas. Testadores devem também lidar com requisitos incompletos ou não documentados. Teste de sistema em requisitos funcionais deve inicialmente utilizar a técnica baseada em especificação mais apropriada (caixa-preta) de acordo com a característica do sistema a ser testado. Por exemplo, uma tabela de decisão pode ser criada por combinações de efeitos descritos em regras de negócio. A seguir, técnica baseada na estrutura (caixa-branca) pode ser utilizada para avaliar a eficácia do teste com respeito ao elemento estrutural, assim como estrutura do menu ou página web. (Ver Capítulo 4.)

Uma equipe de teste independente é frequentemente responsável pelo teste de sistema.

2.2.4 Teste de Aceite (K2)

Teste de aceite frequentemente é de responsabilidade do cliente ou do usuário do sistema; os interessados (*stakeholders*) também podem ser envolvidos.

O objetivo do teste de aceite é estabelecer a confiança no sistema, parte do sistema ou uma característica não específica do sistema. Procurar defeitos não é o principal foco do teste de aceite. Teste de aceite pode avaliar a disponibilidade do sistema para entrar em produção, apesar de não ser necessariamente o último nível de teste. Por exemplo, teste de integração em larga escala pode vir após o teste de aceite de um sistema.

O teste de aceite pode ser realizado em mais de um único nível de teste, por exemplo:

- Um pacote (COTS) de software ter um teste de aceite quando é instalado ou integrado.
- Teste de aceite de usabilidade de um componente pode ser feito durante o teste de componente.
- Teste de aceite de uma nova funcionalidade pode vir antes do teste de sistema.

As formas de teste de aceite incluem tipicamente os seguintes:

Teste de Aceite de Usuário

Normalmente verifica se o sistema está apropriado para o uso por um usuário com perfil de negócio.

Teste Operacional de Aceite

O aceite do sistema pelo administrador dos sistemas inclui:

- Teste de Backup/Restore.
- Recuperação de Desastre.
- Gerenciamento de Usuário.
- Tarefas de manutenção.
- Checagens periódicas de vulnerabilidades de segurança.

Teste de aceite de contrato e regulamento

Teste de aceite de contrato é realizado verificando-se algum critério de aceite incluso em contrato na produção de software encomendado. O critério de aceite deve ser definido quando o contrato é assinado. Teste de aceite de regulamento é quando se verifica a necessidade de adesão a algum regulamento de acordo com outras normas (ex.: segurança, governamental, legislação).

Alfa e Beta Teste (ou teste no campo)

Desenvolvedores de softwares comerciais ou pacotes, muitas vezes precisam obter um *feedback* de clientes em potencial existente no mercado antes que o software seja colocado à venda comercialmente. Alfa Teste é feito no "site" da organização em que o produto foi desenvolvido. Beta Teste, ou teste no campo, é feito pelas pessoas em suas próprias localidades. Ambos os testes são feitos pelos clientes em potencial e não pelos desenvolvedores do produto.

Organizações podem utilizar outros termos como Teste de Aceite de Fábrica e Teste de Aceite no "site", para sistemas que são testados antes e após terem sido movidos ao "site" do cliente.

2.3 Tipos de Teste: o alvo do teste	40 minutos

Termos

Teste caixa-preta, cobertura de código, teste funcional, teste de interoperabilidade, teste de carga, teste de manutenibilidade, teste de performance, teste de portabilidade, teste de regressão, teste de confiabilidade, teste de segurança, teste baseado em especificação, teste de estresse, teste estrutural, teste de usabilidade, teste caixa-branca.

Conceito

Um grupo de atividades de teste pode ser direcionado para verificar o sistema (ou uma parte do sistema) com base em um motivo ou alvo específico.

Cada tipo de teste tem foco em um objetivo particular, que pode ser o teste de uma funcionalidade, a ser realizada pelo software; uma característica da qualidade não funcional, tal como a confiabilidade ou usabilidade, a estrutura ou arquitetura do software ou sistema; ou mudanças relacionadas, ex.: confirmar que os defeitos foram solucionados (teste de confirmação) e procurar por mudanças inesperadas (teste de regressão).

Modelos do software podem ser elaborados e/ou usados no teste estrutural ou funcional. Por exemplo, para o teste funcional, um diagrama de fluxo de processo, um diagrama de transição de estados ou uma especificação do programa, e para teste estrutural um diagrama de controle de fluxo ou modelo de estrutura do menu.

2.3.1 Teste de Função (Teste funcional) (K2)

As funções que um sistema, subsistema ou componente devem realizar podem ser descritas nos seguintes produtos de trabalho: especificação de requisitos; casos de uso, especificação funcional, ou podem não estar documentados. As funções representam "o que" o sistema faz.

Testes funcionais são baseados em funções (descritas nos documentos ou compreendidas pelos testadores), e devem ser realizados em todos os níveis de teste (ex.: teste de componente deve ser baseado na especificação do componente).

Técnicas baseadas em especificação podem ser utilizadas para derivar as condições de teste e casos de testes a partir da funcionalidade do software ou sistema (Ver Capítulo 4). Teste funcional considera o comportamento externo do software (teste caixa-preta).

Um tipo de teste funcional, o teste de segurança, investiga as funções (ex.: um *"firewall"*) relacionados à detecção de ameaça de vírus ou de ações mal intencionadas.

2.3.2 Teste de características do produto de software (testes não funcionais) (K2)

Testes não funcionais incluem, mas não se limita a: teste de performance; teste de carga; teste de estresse; teste de usabilidade; teste de interoperabilidade; teste de manutenibilidade; teste de confiabilidade e teste de portabilidade. É o teste de "como" o sistema trabalha.

Testes não funcionais podem ser realizados em todos os níveis de teste. O termo teste não funcional descreve que o teste é executado para medir as características que podem ser quantificadas em uma escala variável, como o tempo de resposta em um teste de performance. Estes testes podem ser referenciados a um modelo de qualidade como definido na norma "*Engenharia de Software – Qualidade de Produto de Software*" (ISO 9126).

2.3.3 Teste de estrutura/arquitetura do software (teste estrutural) (K2)

Teste estrutural (caixa-branca) pode ser feito em todos os níveis de testes. Recomenda-se utilizar as técnicas estruturais após as técnicas baseadas em especificação, já que ela auxilia a medição da eficiência do teste através da avaliação da cobertura de um tipo de estrutura.

Cobertura é a extensão que uma estrutura foi exercitada por um conjunto de testes, expresso como uma porcentagem de itens cobertos. Se a cobertura não atinge 100%, então mais testes devem ser construídos a fim de testar aqueles itens que não foram contemplados para, desta forma, aumentar a cobertura. Técnicas de cobertura são discutidas no Capítulo 4.

Em todos os níveis de teste, mas especialmente no teste de componente e teste de integração de componentes, ferramentas podem ser usadas para medir a cobertura do código dos elementos assim como as declarações ou decisões. Teste estrutural deve ser baseado na arquitetura do sistema, como uma hierarquia de chamadas.

Teste de estrutura também pode ser aplicado no sistema, integração de sistema ou nível de teste de aceite (por exemplo, para modelos de negócios ou estrutura de menu).

2.3.4 Teste relacionado a mudanças (teste de confirmação e regressão) (K2)

Quando um defeito é detectado e resolvido, o software pode ser retestado para confirmar que o defeito original foi realmente removido. Isto é chamado de teste de confirmação. Depurar (resolver defeitos) é uma atividade do desenvolvimento, e não uma atividade do teste.

Teste de regressão é o teste repetido de um programa que já foi testado, após sua modificação, para descobrir a existência de algum defeito introduzido ou não coberto originalmente como resultado da mudança. Estes defeitos podem estar no software ou em um componente, relacionado ou não ao software. É realizado quando o software, ou seu ambiente é modificado. A quantidade de teste de regressão é baseada no risco de não se encontrar defeitos no software que estava funcionando previamente.

Os testes devem ser repetíveis se forem utilizados nos teste de confirmação e para suportar o teste de regressão.

Teste de regressão pode ser realizado em todos os níveis de teste, e se aplicam aos testes funcionais, não funcionais e estruturais. Testes de regressão são executados muitas vezes e geralmente desenvolve-se vagarosamente, o que faz com que seja um forte candidato à automação.

2.4 Teste de Manutenção — 15 minutos

Termos

Análise de impacto, teste de manutenção

Conceito

Uma vez desenvolvido, um sistema pode ficar ativo por anos ou até mesmo décadas. Durante este tempo o sistema e seu ambiente podem ser corrigidos, modificados ou completados. Teste de manutenção é realizado no mesmo sistema operacional e é iniciado por modificações, migrações ou retirada de software ou sistema.

Alguns exemplos de modificações incluem melhorias planejadas (ex.: baseadas em *"releases"*), mudanças corretivas e emergenciais, além de mudanças de ambiente, como atualização em sistema operacional ou banco de dados, e correções ("*patches*") para expor e encontrar vulnerabilidades do sistema operacional.

Teste de manutenção por migração (ex.: de uma plataforma a outra) pode incluir testes operacionais do novo ambiente tanto quanto a mudança de software.

Teste de manutenção para retirada de um sistema pode incluir o teste de migração de dados, ou arquivamento se longos períodos de retenção de dados forem necessários.

Além de testar o que foi alterado, o teste de manutenção inclui teste de regressão massivo para as partes do sistema que não foram testadas. O escopo do teste de manutenção está relacionado ao risco da mudança, o tamanho do sistema existente e o tamanho da mudança. Dependendo da mudança, o teste de manutenção pode ser feito em todos ou alguns níveis, e em todos ou alguns tipos de testes.

A determinação de como um sistema pode ser afetado por mudanças é chamado de análise de impacto, e pode ser usado para ajudar a decidir quantos testes de regressão serão realizados.

Teste de manutenção pode se tornar uma tarefa complicada se as especificações estiverem desatualizadas ou incompletas.

Referências

2.1.3 CMMI, Craig, 2002, Hetzel, 1998, IEEE 12207
2.2 Hetzel, 1998
2.2.4 Copeland, 2004, Myers, 1979
2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004
2.3.2 Black, 2001, ISO 9126
2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1998
2.3.4 Hetzel, 1998, IEEE 829
2.4 Black, 2001, Craig, 2002, Hetzel, 1998, IEEE 829

3. Técnicas Estáticas (K2) — 60 minutos

Objetivos de estudo para técnicas estáticas

Os objetivos identificam o que você deverá aprender para completar cada módulo.

3.1 Revisão e o processo de teste (K2)

LO-3.1.1 Reconhecer os produtos de trabalho que podem ser examinados pelas diferentes técnicas estáticas (K1).

LO-3.1.2 Descrever a importância e o valor das técnicas estáticas para a avaliação dos produtos de trabalhos. (K2)

LO-3.1.3 Explicar a diferença entre as técnicas dinâmicas e estáticas, considerando objetivos, tipos de defeitos a serem identificados, e o papel dessas técnicas dentro do ciclo de vida do software. (K2)

3.2 Processo de Revisão (K2)

LO-3.2.1 Relembrar as fases, funções e responsabilidades de um processo típico de revisão (K1).

LO-3.2.2 Explicar as diferenças entre os tipos de revisão: revisão informal, revisão técnica, acompanhamento ("*walkthrough*") e inspeções. (K2)

LO-3.2.3 Explicar os fatores de sucesso das revisões (K1).

3.3 Ferramentas de análise estáticas (K2)

LO-3.3.1 Recordar os defeitos mais comuns e erros identificados pela análise estática e compará-los com a revisão e teste dinâmico. (K1)

LO-3.3.2 Listar os benefícios mais comuns da análise estática. (K1)

LO-3.3.3 Listar defeitos de código e modelagem mais comuns que podem ser identificados por ferramentas de análise estática. (K1)

3.1 Revisão e o Processo de Teste (K2)	15 minutos

Termos

Teste dinâmico, Teste estático.

Conteúdo

Ao contrário dos testes dinâmicos, as técnicas de teste estático não pressupõem a execução do software que está sendo testado. Elas são manuais (revisão) ou automatizadas (análise estática).

Revisão é uma maneira de testar o produto de software (incluindo o código) e pode ser realizada bem antes da execução do teste dinâmico. Defeitos detectados durante as revisões o mais cedo possível no ciclo de vida do software são muitas vezes mais barato do que aqueles detectados e removidos durante os testes (ex.: defeitos encontrados nos requisitos).

Uma revisão pode ser feita inteiramente como uma atividade manual, mas há também ferramentas de suporte. A principal atividade manual é examinar o produto de trabalho e fazer os comentários sobre ele. Qualquer software pode ser revisado, incluindo a especificação de requisitos, diagramas, código, plano de teste, especificação de teste, casos de teste, script de teste, manual do usuário ou páginas web.

Os benefícios das revisões incluem a detecção e correção antecipada de defeitos, ganho no desenvolvimento em termos de produtividade, redução do tempo no desenvolvimento, redução do custo e tempo de teste, menos defeitos e melhoria na comunicação. A revisão pode encontrar omissões, por exemplo, nos requisitos, que não são normalmente encontrados no teste dinâmico.

Revisões, análises estáticas e testes dinâmicos têm os mesmos objetivos – identificar defeitos. Eles são complementares: as diferentes técnicas podem encontrar diferentes tipos de defeitos eficazmente e eficientemente. Em contraste com o teste dinâmico, revisões encontram defeitos ao invés de falhas.

Os defeitos mais facilmente encontrados durante revisões do que em testes dinâmicos são: desvios de padrões, defeitos de requisitos, defeitos de modelagem, manutenibilidade insuficientemente e especificação incorreta de interfaces.

3.2 Processo de Revisão (K2)	25 minutos

Termos

Critério de entrada, revisão formal, revisão informal, inspeção, métricas, moderador, revisão em par, revisor, redator, revisão técnica, acompanhamento (*walkthrough*).

Conteúdo

As revisões variam de muito informais para muito formais (ex.: bem estruturadas e reguladas). A formalidade do processo de revisão é relacionada a fatores como a maturidade do processo de desenvolvimento, requisitos legais e reguladores ou a necessidade de acompanhamento de auditoria.

O modo como uma revisão é conduzida depende do seu objetivo (ex.: encontrar defeitos, obter compreensão, discussão ou decisões por um consenso).

3.2.1 Fases de uma revisão formal (K1)

Uma revisão formal normalmente possui as seguintes fases principais:

- **Planejamento**:
  - Definir os critérios de revisão.
  - Selecionar a equipe.
  - Alocar as funções.
  - Definir os critérios de entrada e de saída para os diversos tipos de revisão formal (ex.: inspeção).
  - Selecionar quais as partes dos documentos será visto.
  - Checar os critérios de entrada (para diversos tipos de revisão formal).
- ***Kick-off***:
  - Distribuir os documentos.
  - Explicar os objetivos, processos e documentos para os participantes.
- **Preparação individual**:
  - Análise da documentação para a reunião de revisão.
  - Anotar os defeitos em potenciais, questões e comentários.
- **Reunião de revisão**:
  - Discussão ou registro, com resultados documentados ou anotações (para os tipos de revisões mais formais).
  - Anotar os defeitos, fazer recomendações para o tratamento de defeitos ou tomar decisões sobre os defeitos.
  - Examinar, avaliar e registrar questões durante as reuniões de acompanhamento.
- **Retrabalho**:
  - Resolver defeitos encontrados, tipicamente feitos pelo autor.
  - Registrar os status atuais dos defeitos (para revisões formais).
- **Acompanhamento**:
  - Checar se os defeitos foram encaminhados.
  - Obter métricas.
  - Checar os critérios de saída (para tipos de revisões formais).

3.2.2 Papéis e responsabilidades (K1)

Uma típica revisão formal inclui as funções abaixo:

- **Gerente**: toma decisão durante a realização da revisão, aloca tempo nos cronogramas de projeto e determina se o objetivo da revisão foi atingido.
- **Moderador**: a pessoa que lidera a revisão do documento ou conjunto de documentos, incluindo o planejamento da revisão, e o acompanhamento após a reunião. Se necessário, o moderador mediará entre os vários pontos de vista e é muitas vezes quem responderá pelo sucesso da revisão.
- **Autor**: é a pessoa que escreveu ou que possui a responsabilidade pelos documentos que serão revisados.
- **Revisores**: indivíduos com conhecimento técnico ou de negócio (também chamados inspetores), que, após a preparação necessária, identificam e descrevem os defeitos encontrados no produto em revisão. Revisores podem ser escolhidos para representar diferentes funções e perspectivas no processo de revisão, e é parte integrante de qualquer reunião de revisão.
- **Redator**: documenta todo o conteúdo da reunião, problemas e itens em aberto que foram identificados durante a reunião.

Olhando os documentos de diferentes perspectivas e usando "*check-lists*", tornamos a revisão mais eficaz e eficiente. Por exemplo, um "*check-list*" baseado em perspectivas tais como a do usuário, desenvolvedor, testador, operador, ou um "*check-list*" típico de problemas de requisitos pode ajudar a descobrir problemas não detectados anteriormente.

3.2.3 Tipos de revisão (K2)

Um único documento pode ser objeto para mais de uma revisão. Se mais de um tipo de revisão for usado, a ordem pode variar. Por exemplo, uma revisão informal pode ser conduzida antes de uma revisão técnica, ou uma inspeção pode ser executada em uma especificação de requisitos antes de um acompanhamento com clientes. As principais características, opções e propósitos dos tipos de revisão comumente são:

Revisão informal:

- Não existe processo formal.
- Pode haver programação em pares ou um líder técnico revisando a modelagem e o código.
- A documentação é opcionalmente.
- A importância pode variar dependendo do revisor.
- Principal propósito: uma forma de obter algum benefício a um baixo custo.

Acompanhamento:

- Reunião conduzida pelo autor.
- Cenários, grupos de discussão, exercícios práticos.
- Sessões sem restrição de tempo.
  - Opcionalmente há uma reunião preparatória dos revisores.
  - Opcionalmente, relatórios de revisão e lista de defeitos encontrados são preparados.
- Opcionalmente há um redator.
- Na prática pode variar de informal para muito formal.
- Principal propósito: aprendizagem, obtenção de entendimento e encontrar defeitos.

Revisões técnicas:

- Documentado, processo de detecção de defeito definido que inclui colegas especialistas ou técnicos com a participação opcional da gerência.
- Pode ser feito por um colega sem a participação da gerência.

- Idealmente são conduzidas por um moderador treinado (que não seja o autor).
- Reunião preparatória dos revisores.
- Opcionalmente usa check-lists.
- Elaboração de um relatório de revisão, que inclui a lista de defeitos encontrados, se o produto de software corresponde às suas exigências e, quando apropriado, recomendações relacionadas com as descobertas.
- Na prática, pode variar de informal para muito formal.
- Principais propósitos: discussão, tomada de decisões, avaliar alternativas, encontrar defeitos, resolver problemas técnicos e checar a conformidade da padronização das especificações.

Inspeção :

- Conduzida pelo moderador (que não seja o autor).
- Geralmente é uma análise por pares.
- Papéis definidos.
- Utilização de métricas.
- Processo formal baseado em regras e utilização de check-list.
- Entrada especificada e critérios de saída para a aceitação do produto de software.
- Reunião de preparação.
- Relatório de inspeção, lista de defeitos encontrados;
- Processo de acompanhamento formal.
- Opcionalmente, ter aperfeiçoamento do processo e um leitor.
- Principal propósito: encontrar defeitos.

Acompanhamento, revisões técnicas e inspeções podem ser executados dentro de um grupo de pessoas no mesmo nível organizacional. Este tipo de revisão é chamado de “revisão por pares”.

3.2.4 Fatores de sucesso para as revisões (K2)

Os fatores de sucesso para as revisões incluem:

- Cada revisão tem um objetivo claramente definido.
- A pessoa adequada para os objetivos da revisão deve ser envolvida.
- Testadores são valorizados como revisores que contribuem para a revisão e aprendizado sobre o produto o que lhes permite preparar os testes facilmente.
- Defeitos encontrados são encorajados e expressados objetivamente.
- Deve-se lidar com os problemas pessoais e aspectos psicológicos (ex.: fazer com que a reunião seja uma experiência positiva para o autor).
- A análise é conduzida em uma atmosfera de confiança, o resultado não será utilizado para a avaliação dos participantes.
- Técnicas de revisão são aplicadas de forma a combinar com o tipo e nível do software e revisores.
- Caso necessário, check-lists ou papéis são utilizados para aumentar a eficiência na identificação de defeitos.
- Treinamento é um item importante para as técnicas de revisão, especialmente para as técnicas formais, assim como as inspeções.
- Gerenciamento é importante para um bom processo de revisão (ex.: incorporando o tempo adequado para as atividades de revisão nos cronogramas de projetos).
- Há uma ênfase em aprender e aprimorar o processo.

3.3 Análise Estática por Ferramentas (K2)	20 minutos

Termos

Compilador, complexidade, controle de fluxo, fluxo de dados, análise estática.

Conceito

O objetivo da análise estática é encontrar defeitos no código fonte do software e na modelagem. Análise estática é feita sem a execução do software examinado pela ferramenta; já o teste dinâmico executa o software. Análise estática pode localizar defeitos que são dificilmente encontrados em testes. Como as revisões, a análise estática encontra defeitos ao invés de falhas. Ferramentas de análise estática analisam o código do programa (ex.: fluxo de controle e fluxo de dados), gerando, como saída, arquivos do tipo HTML e XML, por exemplo.

Os benefícios da análise estática são:

- Detecção de defeitos antes da execução do teste.
- Conhecimento antecipado sobre aspectos suspeitos no código ou programa através de métricas, por exemplo, na obtenção de uma medida da alta complexidade.
- Identificação de defeitos dificilmente encontrados por testes dinâmicos.
- Detecção de dependências e inconsistências em modelos de software, como links perdidos.
- Aprimoramento da manutenibilidade do código e construção.
- Prevenção de defeitos, se as lições forem aprendidas pelo desenvolvimento.

Defeitos mais comuns descobertos por ferramentas de análise estática incluem:

- Referência a uma variável com valor indefinido.
- Inconsistências entre as interfaces dos módulos e componentes.
- Variáveis que nunca são usadas ou impropriamente declaradas.
- Código morto.
- Falta de lógica ou lógica errada (loops infinitos).
- Construções excessivamente complicadas.
- Violação de padrões de programação.
- Vulnerabilidade na segurança.
- Violação de sintaxe e de modelos.

Ferramentas de análises estáticas são tipicamente usadas por desenvolvedores (checando regras pré-definidas ou padrões de programação) antes e durante o teste de componente e de integração e por projetistas durante a modelagem do software. Ferramenta de análise estática pode produzir um grande número de mensagens de advertências que precisam ser gerenciadas para permitir o uso mais efetivo da ferramenta.

Compiladores podem oferecer algum suporte para a análise estática, incluindo o cálculo de métricas.

Referências

3.2 IEEE 1028
3.2.2 Gilb, 1993, van Veenendaal, 2004
3.2.4 Gilb, 1993, IEEE 1028
3.3 Van Veenendaal, 2004

4. Técnica de Modelagem de Teste (K3) — 255 minutos

Objetivos de estudo para Técnicas de Modelagem de Teste

Os objetivos identificam o que você deverá aprender para completar cada módulo.

4.1 Identificar as condições de testes e os casos de testes (K3)

LO-4.1.1 Diferenças entre especificação de Planos de Teste, Casos de Testes e procedimentos de Teste. (K2)

LO-4.1.2 Comparar os termos: Condições do Teste, Caso de Teste e Procedimento de Teste. (K2)

LO-4.1.3 Avaliar a qualidade dos Casos de Teste em termos de clara rastreabilidade aos requisitos e resultados esperados. (K2)

LO-4.1.4 Traduzir Casos de Testes em uma especificação de procedimento de teste bem estruturada em um nível de detalhe relevante para conhecimento dos testadores. (K3)

4.2 Categorias de técnicas de modelagem de teste (K2)

LO-4.2.1 Rever as razões de por que tanto a abordagem de casos testes baseada em especificação (caixa preta) quanto baseada em estrutura interna do software (caixa branca) são importantes, e listar as técnicas mais comuns de cada uma. (K1)

LO-4.2.2 Explicar as características e diferenças entre as técnicas de teste baseada em especificação, baseada em estrutura e baseada em experiência. (K2)

4.3 Técnicas baseada em especificação ou caixa-preta (K3)

LO-4.3.1 Montar os casos de teste de um dado software utilizando as seguintes técnicas: Partição de Equivalência, Análise de Valores Limite, Tabela de Decisão e Diagrama de Transição (K3)

LO-4.3.2 Compreender os principais objetivos de cada uma das quatro técnicas, qual nível e tipo de teste mais adequado para se utilizá-las e como a cobertura poderá ser medida. (K2)

LO-4.3.3 Compreender o conceito de Teste de Caso de Uso e seus benefícios. (K2)

4.4 Técnicas baseadas em estrutura interna do software ou caixa-branca (K3)

LO-4.4.1 Descrever o conceito e a importância da cobertura de código. (K2)

LO-4.4.2 Explicar o conceito da cobertura de decisão ou de comandos, demonstrando que estes conceitos podem ser utilizados em outros níveis de testes além do teste de componente (ex.: Teste de regras de negócios em nível de sistema). (K2)

LO-4.4.3 Criar casos de testes de um dado fluxo de controle utilizando as seguintes técnicas: Teste de Sentença e Teste de Decisão.

LO-4.4.4 Avaliar a abrangência da cobertura de sentença e decisão para a completude com respeito a critérios de saída definidos. (K3)

4.5 Técnicas baseadas em experiência (K2)

LO-4.5.1 Rever as razões para se construir os casos de testes com base na intuição, experiência e conhecimento dos defeitos. (K1)

LO-4.5.2 Comparar a técnica baseada em experiência com as baseadas em especificação. (K2)

4.6 Escolhendo técnicas de teste (K2)

LO-4.6.1 Classificar as técnicas de modelagem de teste de acordo com o contexto, base de teste, modelos e características de software (K2)

4.1 Identificando as condições de testes e projetando os casos de testes (K3)	*15 minutos*

Termos

Especificação de caso de teste, modelagem de teste, cronograma de execução do teste, especificação de procedimento de teste, script de teste e rastreabilidade.

Conceito

O processo pode ser realizado de diferentes maneiras, desde informalmente sem muitos dados ou documentação, até um processo muito formal (como o que será descrito ainda nesta seção). O nível de formalidade depende do contexto do teste, o que inclui a organização, maturidade do processo de teste e desenvolvimento, restrições de tempo e as pessoas envolvidas.

Durante a análise de teste, a documentação base de teste é analisada de maneira a determinar o que testar (ex.: identificar as condições de teste). A condição do teste é definida como um item ou evento que pode ser verificado por um ou mais casos de testes (ex.: uma função, transação, característica de qualidade ou elemento estrutural).

Estabelecer a rastreabilidade das condições de testes de volta até as especificações e requisitos permitem analisar o impacto quando os requisitos mudam e, a cobertura de requisitos a ser determinada por um conjunto de testes. Durante a modelagem do teste, o detalhamento da abordagem de teste será implementado com base – entre outras considerações – nos riscos identificados. (Ver mais informações de análise de riscos no Capítulo 5)

Durante a modelagem de teste, os casos de teste e os dados de teste são especificados e criados. Um caso de teste consiste de um conjunto de valores de entrada, pré-condições de execução, resultados esperados e pós-condições de execução, desenvolvidos para cobrir certas condições de teste. O "*Standard for Software Test Documentation*" (IEEE 829-1998) descreve o conteúdo da especificação da modelagem de teste (contendo as condições de teste) e a especificação de caso de teste.

Resultados esperados devem ser produzidos como parte da especificação de um caso de teste e inclui as saídas, mudança de dados e status, e qualquer outra consequência do teste. Se o resultado esperado não for definido, um resultado plausível, porém errado, pode ser interpretado como correto. O resultado esperado deve ser definido antes da execução do teste.

Durante a implementação do teste os casos de teste são desenvolvidos, implementados, priorizadas e organizadas na especificação de procedimento de teste (IEEE STD 829-1998). O procedimento de teste especifica a sequência de ações para a uma execução de teste. Se os testes são executados por uma ferramenta, a sequência de ações é especificada por um script automatizado (que é um procedimento de teste automatizado).

Os vários e scripts de testes automatizados formam uma sequência de execução de teste que define a ordem em que os procedimentos, e/ou os scripts automatizados serão executados e quem os executará. A sequência de execução de teste considera alguns fatores como testes de regressão, priorização, dependências técnicas e lógicas.

4.2 Categorias das técnicas de modelagem de teste (K2)	15 minutos

Termos

Técnicas caixa-preta, técnicas baseadas em experiência, técnicas de modelagem de teste, técnicas caixa-branca.

Conceito

O propósito da técnica de modelagem de teste é identificar as condições e os casos de testes.

Classificar testes como caixa-preta ou caixa-branca é uma diferenciação clássica. Técnicas caixa- preta, (também chamadas de técnicas baseadas em especificação) são uma forma de derivar e selecionar as condições e casos de testes baseados na análise da documentação. Isto inclui testes funcionais e não funcionais, para um componente ou sistema sem levar em consideração a sua estrutura interna. Técnicas de caixa branca (também chamadas de técnicas estruturais ou baseadas em estrutura) são baseadas na estrutura interna de um componente ou sistema.

Algumas técnicas se encaixam claramente em uma única categoria; outras têm elementos de mais de uma categoria.

Este *syllabus* considera técnicas baseadas em especificação ou técnicas baseadas em experiência como técnicas caixa-preta e técnicas baseadas em estrutura como técnicas caixa-branca.

Características comuns das técnicas baseadas em especificação:

- Modelos, formais ou informais, são utilizados para especificação de um problema a ser resolvido, o software ou seu componente.
- Os casos de testes podem ser derivados sistematicamente destes modelos.

Características comuns das técnicas baseadas em estrutura:

- Informações sobre como o software é construído é utilizada para derivar os casos de testes. Por exemplo, código e informações detalhadas de modelagem.
- A extensão da cobertura do software pode ser medida pelos casos de testes. Além disto, os casos de testes podem ser derivados sistematicamente para aumentar a cobertura.

Características comuns de técnicas baseada em experiência:

- Conhecimento e experiência de pessoas são utilizados para derivar os casos de testes.
- Conhecimento de testadores, desenvolvedores, usuários, outros interessados (*stakeholders*) responsáveis pelo software, seu uso e ambiente.
- Conhecimento sobre defeitos prováveis e sua distribuição.

4.3 Técnicas baseadas em especificação ou Caixa-Preta (K3)	120 minutos

Termos

Análise de valores limites, teste de tabela de decisão, partição de equivalência, teste de transição de estados, teste de caso de uso.

4.3.1 Partição de Equivalência (K3)

Na Partição de Equivalência, as entradas do software ou sistema são divididas em grupos que tenham um comportamento similar, podendo ser tratados da mesma forma. Partições (ou classes) de equivalência podem ser encontradas em dados válidos e inválidos (por exemplo, valores que deveriam ser rejeitados). Partições podem também ser identificadas para valores de saídas, valores internos e valores relacionados a tempo, (antes e após um evento) e para parâmetros de interface (durante teste de integração). Testes podem ser elaborados para cobrir as partições. Partição de Equivalência é aplicável a todos os níveis de testes.

A Partição de Equivalência pode ser usada para atingir a cobertura dos valores de entrada e saída. Pode ser aplicada em uma entrada manual, interface entrada de sistema ou como parâmetro de interface num teste de integração.

4.3.2 Análise do Valor Limite (K3)

O comportamento nos limites de uma partição de equivalência é onde existe maior probabilidade de estar incorreto. Portanto, limites são áreas onde testes estão mais propensos a indicar defeitos. Os valores limites de uma partição são seu máximo e seu mínimo. Um valor limite para uma partição válida é um valor limite válido. O limite de partição inválida é um valor limite inválido. Testes podem ser projetados para cobrir tanto valores inválidos como válidos. Quando os casos de testes são projetados, um valor em cada limite é escolhido.

Análise do valor limite pode ser aplicada em todos os níveis de teste. É relativamente fácil aplicá-la, sua capacidade de encontrar defeitos é alta e especificações detalhadas podem ser úteis em sua elaboração.

Esta técnica é muitas vezes considerada uma extensão da partição de equivalência e pode ser aplicada para entradas manuais como, por exemplo, em escalas de tempo ou tabela de limites. Valores limites podem também ser usados para selecionar dados de testes.

4.3.3 Tabela de Decisão (K3)

A tabela de decisão é considerada uma boa alternativa para capturar requisitos de sistemas que contém condições lógicas e para documentar o comportamento interno do sistema. Elas podem ser utilizadas para registrar regras de negócio complexas a serem implementadas. A especificação é analisada e as condições e ações do sistema são identificadas. As condições de entrada e ações são declaradas de uma forma que possam ser entendidas, como verdadeiras ou falsas (*Booleano*). A tabela de decisão contém as condições que disparam as ações, muitas vezes combinações verdadeiras e falsas para todas as condições de entrada, e ações resultantes para cada combinação de condições. Cada coluna da tabela corresponde a uma regra de negócio que define uma única combinação de condições que resulta na execução de ações associadas com aquela regra. A cobertura padrão comumente usada em uma tabela de decisão é ter no mínimo um teste por coluna cobrindo todas as combinações de condições apresentadas.

O grande ganho na utilização da tabela de decisão é que ela cria combinações de condições que geralmente não foram exercitadas durante os testes. Pode ser aplicada a todas as situações quando a execução do software depende de muitas decisões lógicas.

4.3.4 Teste de transição de estados (K3)

Um sistema pode exibir respostas diferentes dependendo da sua condição atual ou de estado anterior. Neste caso, o comportamento do sistema pode ser representado como um diagrama de transição de estados. Permite ao testador visualizar o software em termos de estados, transições entre estados, as entradas ou eventos que disparam as mudanças de estado (transição) e as ações que podem resultar daquelas transições.

Os estados do sistema, ou objetos em teste, são isolados, identificáveis e finitos. Uma tabela de estado exibe a relação entre estados e entradas, e pode destacar possíveis transições inválidas.

Os testes podem ser construídos para cobrir uma sequência típica de status, cobrir todos os estados, exercitar todas as transições, exercitar uma sequência específica de transições ou testar transições inválidas.

Teste de transição de status é muito utilizada em softwares industriais embarcados e automações técnicas em geral. No entanto, a técnica é também adequada para modelar um objeto de negócio tendo estado específico ou para testar fluxos de telas de diálogos (exemplo: aplicação de internet e cenários de negócios).

4.3.5 Teste de Caso de Uso (K2)

Testes podem ser especificados a partir de casos de uso ou cenários de negócios. Um caso de uso descreve interações entre os atores (usuários e o sistema) que produz um resultado relevante para um usuário do sistema. Cada caso de uso tem pré-condições, que precisam ser garantidas para que o caso de uso funcione com sucesso. Cada caso de uso é finalizado com uma pós-condição que representa os resultados observados e o estado final do sistema após o término do caso de uso. Um caso de uso normalmente tem um cenário mais comum (mais provável), e algumas vezes ramificações.

Caso de uso descreve o fluxo de processo de um sistema baseado nas suas possibilidades de utilização. Os casos de testes derivados de casos de uso são muito úteis na descoberta de defeitos no fluxo do processo durante a utilização do sistema no mundo real. Casos de uso muitas vezes são tratados como cenários, e úteis para construir testes de aceite com a participação do usuário final. Eles podem ajudar a descobrir defeitos de integração causados pela interação e interferência de diferentes componentes, que testes individuais de componentes podem não ter detectado.

4.4 Técnicas baseadas em estrutura ou Caixa-Branca (K3)	60 minutos

Termos

Cobertura de código, cobertura de decisão, cobertura de sentença, teste baseado em estrutura.

Conceito

Teste de estrutura ou caixa-branca é baseado na estrutura do software ou sistema, como veremos nos exemplos que seguem abaixo:

- **Nível de Componente**: a estrutura é o próprio código, ex.: comandos, decisões e desvios.
- **Nível de Integração**: a estrutura pode ser uma árvore de chamadas (um diagrama em que um módulo chama outros módulos).
- **Nível de Sistema**: A estrutura pode ser uma estrutura de menu, processos de negócios ou estruturas das páginas Web.

Nesta seção há basicamente duas técnicas de cobertura de código baseados em comandos e decisões serão discutidas. Para teste de decisão, um diagrama de controle de fluxo pode ser utilizado para visualizar as alternativas de cada decisão.

4.4.1 Teste e Cobertura de Sentença (K3)

No teste de componente, cobertura de sentença é avaliada pela porcentagem de sentenças executáveis que foram exercitadas por um conjunto de casos de testes. No teste de sentenças derivam-se casos de teste para executar sentenças específicas, normalmente para se aumentar a cobertura.

4.4.2 Teste e Cobertura de Decisão (K3)

Cobertura de decisão, também chamada de teste de ramificação, é avaliada pela porcentagem dos resultados da decisão (por exemplo, as opções de "*Verdadeiro*" ou "*Falso*" de uma expressão condicional - *IF*) que foram exercitados em um conjunto de casos de teste. No teste de decisão derivam-se os casos de testes para executar decisões específicas, normalmente para se aumentar a cobertura.

Teste de decisão é uma forma de teste de controle de fluxo, já que ele gera um fluxo específico através dos pontos de decisões. A cobertura de decisão é mais eficiente que a cobertura de sentenças: 100% da cobertura de decisão garante 100% da cobertura de sentença, mas não vice-versa.

4.4.3 Outras técnicas baseadas na estrutura (K1)

Existem formas mais detalhadas de cobertura estrutural além da cobertura de decisão, por exemplo, cobertura de condições e cobertura de múltiplas condições.

O conceito de cobertura também pode ser aplicado a outros níveis de teste (teste de integração) no qual, as porcentagens de módulos, componentes ou classes são exercitadas por um conjunto de casos de teste. Por exemplo, poderia expressá-las como cobertura de módulos, componentes ou classes.

Normalmente é utilizada uma ferramenta para dar o suporte de teste de estrutura do código.

4.5 Técnicas baseadas na experiência (K2)	30 minutos

Termos

Teste exploratório, ataque (de falha).

Conceito

Os testes baseados na experiência são testes derivados da intuição e conhecimento dos testadores através de sua experiência em aplicações e tecnologia similares. Quando usado para aumentar a técnica sistemática, testes intuitivos podem ser úteis para identificar testes específicos que não são facilmente identificados pelas técnicas formais, especialmente quando aplicado após ter estabelecido o processo mais formal. No entanto esta técnica pode produzir amplas variedades e graus de eficiência, dependendo da experiência do testador.

Possivelmente a técnica mais amplamente aplicada é a de supor (adivinhar) onde estão os erros. Uma abordagem estruturada da técnica de dedução de erros é enumerar uma lista de possíveis erros e construir testes com objetivo de atacar/cobrir estes erros. Esta abordagem sistemática é chamada de ataque de falha. Estas listas de defeitos e falhas podem ser construídas com base na experiência, dados de defeitos/falhas disponíveis e do conhecimento comum de como o software falha.

Teste exploratório ocorre simultaneamente à modelagem, execução e registro de teste, e baseia-se nos objetivos de teste, onde é realizado em um tempo predefinido. É uma abordagem muito usual, em locais onde a especificação é rara ou inadequada e existe grande pressão por conta de prazo, ou para aprimorar/complementar um teste mais formal. Pode servir como uma checagem do processo de teste, assegurando que os defeitos mais importantes sejam encontrados.

4.6 Escolhendo as técnicas de teste (K2)	15 minutos

Termos

Não há.

Conceito

A escolha de qual técnica utilizar dependerá de uma série de fatores, incluindo o tipo de sistema, padrões, clientes, requisitos contratuais, nível do risco, tipos de riscos, objetivos do teste, documentação disponível, conhecimento dos testadores, tempo, dinheiro, ciclo de desenvolvimento, modelo de caso de uso e uma experiência prévia do tipo de defeitos encontrados.

Algumas técnicas são mais facilmente aplicadas em certas situações e níveis de teste, já outras são aplicáveis a todos os níveis.

Ao criar casos de teste, os testadores geralmente usam uma combinação de técnicas de teste, incluindo regras de processo e técnicas de *data-driven* para garantir uma cobertura adequada do objeto em teste.

Referências

4.1 Craig, 2002, Hetzel, 1998, IEEE 829
4.2 Beizer, 1990, Copeland, 2004
4.3.1 Copeland, 2004, Myers, 1979
4.3.2 Copeland, 2004, Myers, 1979
4.3.3 Beizer, 1990, Copeland, 2004
4.3.4 Beizer, 1990, Copeland, 2004
4.3.5 Copeland, 2004
4.4.3 Beizer, 1990, Copeland, 2004
4.5 Kaner, 2002
4.6 Beizer, 1990, Copeland, 2004

5. Gerenciamento de Teste (K3) — 180 minutos

Objetivos de aprendizado para o gerenciamento de teste

Os objetivos identificam o que você será capaz de fazer após a finalização de cada módulo.

5.1 Organização do Teste (K2)

LO-5.1.1 Reconhecer a importância e independência do teste. (K1)
LO-5.1.2 Listar as vantagens e desvantagem de uma equipe de teste independente em uma organização. (K2)
LO-5.1.3 Reconhecer os diferentes membros a serem consideradas para criar uma equipe de teste. (K1)
LO-5.1.4 Rever as tarefas típicas de um testador e líder de teste. (K1)

5.2 Planejamento e estimativa do Teste (K2)

LO-5.2.1 Reconhecer os diferentes níveis e objetivos do planejamento do teste. (K1)
LO-5.2.2 Sumarizar o propósito e o conteúdo de um plano de teste, especificação da modelagem e os procedimentos do teste de acordo com "*Standard for Software Test Documentation*" (IEEE 829). (K2)
LO-5.2.3 Diferenciar entre aproximações conceituais do teste, tais como analítico, baseado em modelo, metódico, dinâmico/heurístico e regressão. (K2)
LO-5.2.4 Diferenciar entre o assunto do planejamento do teste para um sistema e para a execução programada do teste (K2)
LO-5.2.5 Programar a execução do teste para um conjunto de casos do teste, considerando a priorização, e dependências técnicas e lógicas. (K3)
LO-5.2.6 Listar as tarefas de preparação e execução de teste que precisam ser planejadas. (K1)
LO-5.2.7 Rever os fatores típicos que influenciam o esforço de teste. (K1)
LO-5.2.8 Diferenciar duas estimativas com diferentes abordagens: baseada em métricas e baseada em especialistas. (K2)
LO-5.2.9 Reconhecer / justificar um critério de saída para um nível de teste específico e grupos de casos de testes (ex.: para um teste de integração, teste de aceite ou testes de usabilidade). (K2)

5.3 Controle e Monitoração do progresso do Teste (K2)

LO-5.3.1 Rever as métricas comuns usadas para monitorar a preparação e execução do teste. (K1)
LO-5.3.2 Compreender e interpretar as métricas de teste de um relatório ou controle de teste (ex.: número de defeitos encontrados, corrigidos e testes que passaram e falharam). (K2)
LO-5.3.3 Sumarizar o objetivo e o conteúdo do relatório de teste sumarizado de acordo com o "*Standard for Software Test Documentation*" (IEEE 829). (K2)

5.4 Gerenciamento de Configuração (K2)

LO-5.4.1 Sumarizar como a gerência de configuração pode servir de suporte ao teste. (K2)

5.5 Riscos e Teste (K2)

LO-5.5.1 Descrever o risco como um possível problema que pode ameaçar a realização do projeto. (K2)
LO-5.5.2 Relembrar que os riscos são determinados pela possibilidade (de acontecer) e o impacto (dano resultante se ele acontecer). (K1)
LO-5.5.3 Distinguir entre o risco do projeto e risco do produto (K2)

LO-5.5.4 Reconhecer um risco de produto e de projeto típico. (K1)

LO-5.5.5 Descrever, utilizando exemplos, como a análise e gerenciamento de risco podem ser utilizados para planejar os testes. (K2)

5.6 Gerenciamento de Incidente (K3)

LO-5.6.1 Reconhecer o conteúdo do relatório de incidente do "*Standard for Software Test Documentation*" (IEEE 829). (K1)

LO-5.6.2 Preparar um relatório de incidente cobrindo a observação de uma falha durante os testes. (K3)

5.1 Organização do Teste (K2)	30 minutos

Termos

Testador, líder de teste, gerente de teste

5.1.1 A organização e o teste independente (K2)

A eficácia na procura por defeitos por testes e revisão pode ser aperfeiçoada utilizando-se testadores independentes. As opções para independências são:

- Nenhum testador independente. Os desenvolvedores testam seu próprio código.
- Testadores independentes provenientes da equipe de desenvolvimento.
- Equipe de teste independente ou grupo dentro da organização, reportando a um gerente de projeto ou diretor executivo.
- Testadores independentes da organização do negócio, dos usuários e da área de TI.
- Especialistas em teste independente para um objetivo específico de teste, como testes de usabilidade, segurança ou certificação (quem certifica se o software está em conformidade com as normas e padrões).
- Equipe de teste terceirizada ou independente da organização.

Para projetos longos, complexos e críticos, normalmente é melhor ter vários níveis de teste, com algum ou todos os níveis efetuados por equipes independentes de teste. A equipe de desenvolvimento pode participar do teste, especialmente para os testes de baixo nível, mas sua falta de objetividade muitas vezes limita a efetividade do teste. A equipe independente de teste deve ter a autoridade para requisitar e definir os processos de teste e suas regras, mas os testadores teriam que exercer estas funções somente sob um claro gerenciamento.

Os benefícios da independência da equipe de testes são:

- Testadores independentes conseguem enxergar outros defeitos e são imparciais.
- Um testador independente é capaz de verificar concepções pessoais criadas durante a especificação e implementação de um sistema.

As desvantagens incluem:

- Isolamento da equipe de desenvolvimento (se for tratado com independência total).
- Equipe pode se tornar um gargalo, considerando-se o último ponto de controle.
- Os desenvolvedores podem perder o senso da responsabilidade pela qualidade.

A atividade do teste deve ser realizada por pessoas com uma função específica de teste, ou pode ser feita por alguém em outra função, assim como gerente de projeto, gerente de qualidade, desenvolvedores, especialistas no negócio, suporte de infraestrutura ou operações em TI.

5.1.2 Tarefas do líder de teste e dos testadores (K1)

Neste *syllabus*, são abordadas duas funções: líder de teste e testador. As atividades feitas por pessoas nestas funções dependem do contexto do produto e projeto da organização.

Algumas vezes o líder de teste é chamado de gerente ou coordenador de teste. A função do líder pode ser efetuada por um gerente: de projeto, de desenvolvimento, da qualidade ou até mesmo por um gerente de um grupo de teste. Em projetos longos podem existir duas posições: Líder do Teste e Gerente de Teste.

Normalmente, o líder é responsável pelo planejamento, monitoração e controle d as atividades de testes e tarefas como as definidas na seção 1.4.

As tarefas típicas de um líder são:

- Coordenar a estratégia de teste e planejar com o gerente de projetos e outros envolvidos.
- Escrever ou revisar uma estratégia de teste para o projeto e uma política de teste para a empresa.
- Contribuir com perspectiva do teste para outras atividades, como o planejamento de integração.
- Planejar os testes considerando o contexto e compreendendo os riscos, incluindo a escolha da abordagem do teste, estimativa de tempo, esforço, custo, aquisição de recursos, definição de níveis e ciclos de testes, definição de objetivos e planejamento da gestão de incidente.
- Iniciar a especificação, preparação, implementação e execução dos testes e monitorar o controle da execução.
- Adaptar o planejamento baseado nos resultados e progresso do teste (algumas vezes documentado em relatório de andamento) e tomar ações necessárias para resolver problemas.
- Preparar o gerenciamento de configuração do *testware* para facilitar a rastreabilidade.
- Introduzir métricas para medir o progresso do teste e avaliar a qualidade do teste e do produto.
- Decidir o que pode ser automatizado, em que grau e como.
- Escolher ferramenta de apoio e organizar treinamento para seus usuários (testadores).
- Decidir sobre a implementação do ambiente de teste.
- Montar um relatório com base nas informações obtidas durante o teste.

As tarefas típicas de um testador podem incluir:

- Revisar e contribuir no planejamento dos testes.
- Analisar, revisar e avaliar os requisitos de usuários, especificações e modelos para testabilidade.
- Criar especificação de teste.
- Configurar o ambiente de teste (muitas vezes com a coordenação do administrador de sistemas e redes).
- Preparar e adquirir dados para os testes.
- Implementar os testes em todos os níveis, execução, registro, avaliação dos resultados e documentar os desvios dos resultados esperados.
- Utilizar ferramenta de administração, gestão e monitoração de teste se necessário.
- Automatizar testes (esta tarefa pode ser efetuada pelo desenvolvedor ou por um especialista em automação).
- Medir a performance dos componentes e sistemas (se aplicável).
- Rever os testes desenvolvidos por outras pessoas.

As pessoas que trabalham na análise, construção, tipos específicos de teste ou automação podem se especializar nestas funções. Dependendo do nível do teste e do risco relacionado ao produto e o projeto, pessoas diferentes podem ocupar a função do testador, mantendo algum grau de independência.

Tipicamente, testadores em nível de componente e integração são desenvolvedores, testadores em nível de aceite podem ser os especialistas no negócio ou usuário final e testadores para o nível de teste operacional podem ser os próprios operadores.

5.2 Organização do Teste (K2)	50 minutos

Termos

Abordagem de teste, estratégia de teste

5.2.1 Planejamento de Teste (K2)

Esta seção trata dos objetivos do planejamento do teste dentro do desenvolvimento e implementação dos projetos. O planejamento pode ser documentado em um plano de teste mestre ou de projeto separado em vários planos de testes para diferentes níveis, assim como teste de aceite ou teste de sistemas. A essência dos documentos de planejamento de teste é coberta pelo "*Standard for Software Test Documentation*" (IEEE 829).

O planejamento é influenciado pela política de teste da organização, o escopo, objetivo, riscos, obstáculos, críticas e disponibilidade de recursos. Quanto maior for o projeto e o progresso do planejamento dos testes, mais informações estarão disponíveis e mais detalhes podem ser incluídos no plano.

Planejamento do teste é uma atividade contínua realizada durante todo o processo do ciclo de vida do software. O retorno (*feedback*) da atividade do teste é utilizado para identificar riscos de mudanças, para que ajustes no planejamento sejam efetuados.

5.2.2 Atividades no Planejamento de testes (K2)

As atividades no planejamento do teste podem incluir:

- Determinação do escopo e risco, identificando os objetivos do teste.
- Definição completa da abordagem do teste (estratégia de teste), incluindo a definição do nível de testes, dos critérios de entrada e saída.
- Integração e coordenação da atividade de teste no ciclo de vida do software (aquisição, fornecimento, desenvolvimento, operação e manutenção).
- Tomar a decisão sobre o que testar, quais as funções executarão as atividades de teste, quando e como as atividades podem ser realizadas, como o resultados dos testes serão avaliados e quando parar o teste (critério de saída).
- Programar as atividades de análise e planejamento dos testes
- Programar a implementação, execução e validade dos testes.
- Designar recursos para as diferentes tarefas definidas.
- Definir o nível de detalhe, estrutura e modelos para a documentação dos testes.
- Escolher métricas para monitorar, controlar a preparação e execução do teste, resolver defeitos e apontar os riscos.
- Configurar o nível de detalhe para os procedimentos de teste de forma a prover informações suficientes para que o suporte possa reproduzir o incidente.

5.2.3 Critério de Entrada (K2)

Os critérios de entrada definem quando começar um teste, no início de um nível de teste ou quando um conjunto de testes está pronto para execução.

Os critérios de entrada podem ser constituídos de:

- Disponibilidade do ambiente de teste.
- Preparação da ferramenta de no ambiente de teste.

- Disponibilidade de código a ser testado.
- Disponibilidade dos dados de teste.

5.2.4 Critério de Saída (K2)

Os critérios de saída definem quando parar de testar, no final de um nível de teste ou quando um conjunto de testes realizados atingiu um objetivo específico.

Os critérios de encerramento podem ser constituídos de:

- Métricas como a cobertura de código, riscos ou funcionalidades.
- Estimativa da densidade de defeitos ou segurança nas medições.
- Custos.
- Riscos residuais, como defeitos não solucionados ou falta de cobertura de teste em algumas áreas.
- Cronograma, baseado na data de entrega do produto.

5.2.5 Estimativa do teste (K2)

Duas abordagens para estimativa do esforço do teste são cobertas no *syllabus*:

- Estimativa do esforço do teste baseado em métricas de projetos anteriores ou similares, ou baseado em valores típicos.
- Estimativas das tarefas pelo próprio executor ou por especialistas.

Uma vez que a estimativa do esforço do teste é efetuada, recursos podem ser alocados e um cronograma pode ser elaborado.

O esforço do teste pode depender de inúmeros fatores que incluem:

- **Características do produto**: a qualidade da especificação ou outra informação usada por projetos de teste, o tamanho do produto, a complexidade do problema, os requisitos para segurança e os requisitos para documentação.
- **Características do processo de desenvolvimento**: A estabilidade da organização, ferramentas usadas, processos de teste, experiência das pessoas envolvidas e pressão no prazo.
- **As saídas do teste**: o número de defeitos e a quantidade de retrabalho necessária.

5.2.6 A Estratégia do Teste, Abordagem de teste (K2)

A abordagem de teste é a implementação da estratégia de teste para um projeto específico. A abordagem de teste é definida e refinada nos planos de teste e na modelagem de teste. Geralmente, ela inclui as decisões tomadas com base na meta do projeto (teste) e avaliação de risco. É o ponto de partida para o planejamento do processo de teste, para selecionar as técnicas de modelagem de teste e tipos de teste a ser aplicado, e para a definição dos critérios de entrada e saída.

A abordagem escolhida depende do contexto e pode considerar os riscos, perigos de segurança, recursos disponíveis e habilidades, tecnologia, natureza do sistema, objetivos do teste, e regulamentações.

Abordagens típicas incluem:

- **Analítica**: nas quais os testes são direcionados nas áreas do software ou sistema onde apresentem maiores riscos.
- **Baseada em modelos**: nas quais os testes são baseados em dados informais estatísticos sobre taxa de erros (tais como erros operacionais e de segurança).
- **Abordagem metódica**: como a baseada em falhas (incluindo dedução de erros e injeção de falhas), baseadas em *check-list*, e baseadas em característica de qualidade.

- **Compatível com processos ou padrões**: como algumas especificadas por padrões da indústria ou as várias metodologias ágeis.
- **Dinâmica e heurística**: tais como os testes exploratórios onde a atividade de testar é mais reativa do que pré-planejada e onde a execução e avaliação são tarefas concorrentes.
- **Baseada em conselhos**: como os testes em que a cobertura é dirigida por conselhos de especialistas em tecnologia ou negócio fora do time de teste.
- **Regressão**: como aqueles em que há o reuso do material de teste, automação extensiva dos testes funcionais de regressão, e um conjunto de testes padrão.

Diferentes abordagens para montar uma estratégia podem ser utilizadas como, por exemplo, abordagem baseada em risco.

5.3 Controle e Monitoração do Progresso do Teste (K2)	20 minutos

Termos

Densidade do defeito, taxa de falha, controle do teste, monitoração do teste e relatório de resumo do teste.

5.3.1 Monitoração do Progresso do Teste (K1)

O propósito da monitoração do progresso do teste é permitir uma visibilidade sobre as atividades do teste. As informações a serem monitoradas podem ser coletadas manualmente ou automaticamente e serem utilizadas para medir os critérios de saída, como cobertura. Métricas podem ser usadas para avaliar o progresso em relação ao orçamento e cronogramas planejados.

As métricas mais comuns incluem:

- Porcentagem de trabalho na preparação do caso de teste (ou porcentagem de casos de testes devidamente planejados).
- Porcentagem de trabalho na preparação do ambiente.
- Execução dos casos de testes (números de casos de teste executados ou não, testes com resultados positivos e negativos).
- Informações dos defeitos (densidade do defeito, defeitos encontrados e resolvidos, taxas de falha e resultado de retestes).
- Cobertura de requisitos, riscos ou código.
- Confiança subjetiva do testador sob o produto
- Datas dos pontos de controle.
- Custo do teste, incluindo o custo comparado ao benefício de encontrar o próximo erro ou de executar o próximo teste.

5.3.2 Relatório do teste (K2)

O relatório do teste é constituído de informações resumidas sobre o esforço do teste, incluindo:

- O que aconteceu durante a o período do teste, e qual o melhor momento de parar.
- Informações e métricas para dar suporte na tomada de decisão e recomendações sobre futuras ações, tais como avaliação dos defeitos persistentes, vantagem econômica da continuação dos testes e dos riscos consideráveis apontados além do nível de confiança no software testado.

A essência de um relatório de teste é tratada no "*Standard for Software Test Documentation*" (IEEE 829). As métricas podem ser coletadas durante ou ao final do teste:

- A adequação dos objetivos do teste com o nível do teste.
- A adequação da abordagem/estratégia do teste.
- A eficácia dos testes em respeito a seus objetivos.

5.3.3 Controle do Teste (K2)

O controle do teste descreve qualquer orientação ou ação corretiva tomada como resultado de informações e métricas coletadas e relatadas. As ações podem abranger qualquer atividade de teste e pode afetar qualquer outro software de atividade do ciclo de vida ou tarefa.

Exemplos de controle de teste:

- Tomar decisões baseadas em informações adquiridas na monitoração dos testes.
- Priorizar novamente os testes quando riscos são identificados.
- Mudar o cronograma de acordo com disponibilidade do ambiente de teste.
- Definir um critério de entrada para se iniciar o reteste de bugs resolvidos pelo desenvolvedor antes de aceitá-lo em uma *build*.

5.4 Gerenciamento de Configuração (K2)	10 minutos

Termos

Gerenciamento de configuração, controle de versão.

Conceito

O propósito do gerenciamento de configuração é estabelecer e manter a integridade dos produtos (componentes, dados e documentação) do software ou sistema durante todo o projeto ou ciclo de vida do produto.

Para o teste, o gerenciamento de configuração pode garantir que:

- Todos os itens do software são identificados, controladas as mudanças, seus relacionamentos, facilitando manutenção e a rastreabilidade durante todo o processo de teste.
- Todos os documentos e itens do software são referenciados sem ambiguidade na documentação do teste.

Para o testador, o gerenciamento de configuração ajuda na identificação única (e a reprodução) do item testado, documentos de testes, aos testes e aos scripts de execução de testes.

Durante o planejamento do teste a ferramenta e processos de gerenciamento de configuração devem ser escolhidos, documentados e implementados.

5.5 Riscos e Teste (K2)	30 minutos

Termos

Risco do produto, risco do projeto, risco, teste baseado em risco.

Conceito

Risco pode ser definido como uma chance de um evento indesejável acontecer, causando um problema em potencial. O nível do risco pode ser determinado pela possibilidade do evento acontecer e os danos que podem causar caso aconteça.

5.5.1 Riscos no Projeto (K2)

Riscos no projeto são os aqueles que comprometem a capacidade do projeto não atender seus objetivos.

Fatores organizacionais:

- Falta de conhecimento da equipe.
- Reciclagem e treinamento pessoal.
- Fatores políticos como:
  - Problemas com testadores comunicando suas necessidades com os resultados dos testes.
  - Dificuldade para passar informações (defeitos) encontradas nos testes e revisões, não permitindo o aprimoramento do desenvolvimento e, da prática do teste.
- Atitudes impróprias em relação às expectativas do teste (ex.: a não valorização dos defeitos encontrados durante os testes).

Fatores Técnicos:

- Problema na definição correta do requisito.
- Até que ponto os requisitos podem ser satisfeitos dadas restrições diversas.
- A qualidade da modelagem, do código e do teste.

Problemas do fornecedor:

- Falhas de terceiros
- Problemas contratuais

Quando analisando, gerenciando e mitigando os riscos, o gerente de teste é conduzido por um princípio de gerenciamento de projeto bem estabelecido. A definição de planos de teste contida no "*Standard for Software Test Documentation*" (IEEE 829-1998) requisita que os riscos e a contingências devam ser declaradas.

5.5.2 Riscos do Produto (K2)

Áreas de potenciais de falhas, (futuros eventos ou riscos indesejáveis) no software ou sistema são conhecidas como riscos para a qualidade do produto final. Incluindo:

- Software instável (com alta probabilidade de erros).
- O dano potencial que um software ou hardware pode causar a uma pessoa ou companhia.
- Software com características pobres (funcionalidade, segurança, confiança, usabilidade e performance).
- Software com integridade de dados e qualidade pobres (pendências na migração de dados, problemas de conversão de dados, problemas de transporte de dados, violação de padrões de dados).
- Software que não cumpre com seus objetivos.

Risco é usado para se decidir quando começar e onde deverá se testar com mais frequência; o teste é utilizado para reduzir o risco da ocorrência de um efeito indesejado, ou para reduzir seu impacto.

Risco do produto é um tipo especial de risco porque compromete o sucesso de um projeto. A atividade de controle de riscos fornece informações sobre riscos residuais, por medir a eficiência da retirada de defeitos críticos e dos planos de contingência.

Um teste baseado nos riscos pode fornecer oportunidades *proativas* para reduzir os níveis do risco do produto quando é abordado no estágio inicial do projeto. Envolve a identificação do risco e seu uso para orientar o planejamento, especificação, preparação e execução do teste. Os riscos identificados podem ser utilizados para:

- Determinar a técnica de teste a ser empregada.
- Determinar o nível de detalhamento do teste.
- Priorizar o teste na tentativa de buscar erros críticos o mais cedo possível.
- Determinar se alguma atividade (que não seja o teste) poderia ser efetuada para reduzir o risco (ex.: prover treinamento).

Teste baseado no risco auxilia os *stakeholders* a determinar os riscos e níveis de testes, com base no conhecimento coletivo e na visão do projeto.

Para assegurar que a chance de um produto falhar seja minimizada, o gerenciamento de riscos deverá prover disciplina para:

- Avaliar (e reavaliar periodicamente) o que poderá dar errado (riscos).
- Determinar quais riscos são importantes para negociá-los.
- Implementar ações para negociar aqueles riscos.

Além disto, o teste pode demonstrar a identificação de novos riscos, que riscos deveriam ser reduzidos e diminuir as incertezas sobre os riscos.

5.6 Gerenciamento de Incidente (K3) — 40 minutos

Termos

Registro de Incidente, Gerência de Incidente, Reporte de Incidente.

Conceito

Levando em consideração que um dos objetivos do teste é encontrar defeitos, as discrepâncias entre o resultado atual e o esperado precisam ser registradas como incidentes. Um incidente precisa ser investigado e pode se tornar um defeito. Ações apropriadas para dispor incidentes e defeitos devem ser disponíveis. Incidente e defeito deve ser rastreável desde a descoberta, classificação até à correção e confirmação da resolução. Para gerenciar os incidentes, a empresa deve estabelecer processos e regras para classificá-los.

Incidentes podem ser descobertos durante o desenvolvimento, o teste e a utilização do software. Eles podem se revelar por problemas no código, por funções do sistema, documentação de desenvolvimento, documentação de teste, manual de instalação ou manual do usuário.

O Relatório de Incidentes tem os seguintes objetivos:

- Prover aos desenvolvedores e outros envolvidos um retorno sobre o problema para permitir a identificação, isolamento e correção se necessário.
- Prover aos líderes de teste um meio para se rastrear a qualidade do sistema em teste e o progresso do teste.
- Prover ideias para aprimorar o processo de testes.

Os detalhes de um relatório de incidente podem incluir:

- Data da emissão, autor, status e organização.
- Resultados esperados e resultados atuais.
- Identificação do item de teste (item de configuração) e ambiente.
- Processo do ciclo de vida do sistema ou software em que o incidente foi descoberto.
- Descrição do incidente para permitir a reprodução e resolução, incluindo *logs*, *database dumbs*, ou *screenshots*.
- Escopo e grau de impacto para os *stakeholder*.
- Severidade do impacto no sistema.
- Urgência / Prioridade na correção.
- Estado (*status*) do incidente (aberto, rejeitado, duplicado, aguardando resolução, aguardando reteste ou fechado).
- Conclusão, recomendações e aprovações.
- Comentários gerais, tais como outras áreas que podem ser afetadas por uma mudança resultante de um incidente.
- Histórico de mudanças, como a sequência de ações tomadas pela equipe envolvida no projeto com respeito ao isolamento do incidente, reparo e confirmação da resolução.
- Referências, incluindo a identificação da especificação do caso de teste que revelou o problema.

A estrutura de um relatório de incidente pode ser encontrada no *"Standard for Software Test Documentation" (IEEE Std. 829-1998)*.

Referências:

5.1.1 Black, 2001, Hetzel, 1998
5.1.2 Black, 2001, Hetzel, 1998
5.2.5 Black, 2001, Craig, 2002, IEEE 829, Kaner 2002
5.3.3 Black, 2001, Craig, 2002, Hetzel, 1998, IEEE 829

5.4 Craig, 2002
5.5.2 Black, 2001 , IEEE 829
5.6 Black, 2001, IEEE 829

6. Ferramentas de Suporte a Teste (K2)	80 minutos

Objetivos de aprendizado para ferramentas de suporte a teste

Os objetivos identificam o que você deverá fazer para completar cada módulo.

6.1 Tipos de ferramentas de teste (K2)

LO-6.1.1 Classificar diferentes tipos de ferramentas de acordo com a atividade de do processo de teste. (K2)

LO-6.1.3 Explicar o termo ferramenta de teste e o propósito de ferramentas de suporte a teste. (K2)

6.2 Uso efetivo de ferramentas: benefícios potenciais e riscos (K2)

LO-6.2.1 Resumir os benefícios e riscos potenciais da automação e das ferramentas de suporte a teste. (K2)

LO-6.2.2 Relembrar considerações especiais para ferramentas de execução de teste, análise estática e gerenciamento de testes. (K1)

6.3 Introduzindo uma ferramenta em uma organização (K1)

LO-6.3.1 Apontar os princípios para a implementação de uma ferramenta em uma organização. (K1)

LO-6.3.2 Discutir os objetivos de uma prova de conceito/piloto para avaliação da ferramenta. (K1)

LO-6.3.3 Reconhecer que outros fatores além da simples aquisição são fundamentais para o bom aproveitamento da ferramenta de suporte. (K1)

6.1 Tipos de Ferramentas de Teste (K2)	45 minutos

Termos

Ferramentas de gerenciamento de configuração, ferramenta de cobertura de código, ferramenta de depuração, ferramenta de análise dinâmica, ferramenta de gestão de incidentes, ferramenta de teste de carga, ferramenta de modelagem, ferramenta de monitoração, ferramenta de teste de performance, efeito da monitoração, ferramentas de gerenciamento de requisitos, ferramenta de revisão, ferramentas de segurança, ferramentas de análise estática, ferramenta de teste de estresse, ferramenta de comparação de teste, ferramenta de preparação de dados, ferramenta de modelagem de teste, ambiente preparado de testes, ferramentas para execução do teste, ferramentas para gerenciamento do teste, *framework* de teste de unidade.

6.1.1 Ferramentas de suporte a teste (K2)

Ferramentas de teste podem ser usadas para uma ou mais atividades de suporte a teste. Estas incluem:

1) Ferramentas que são diretamente utilizadas no teste, como ferramentas de execução de teste, ferramentas de geração de dados de teste e ferramentas de comparação de resultados.
2) Ferramentas que auxiliam na gestão do processo de teste, como aquelas utilizadas para gerenciar testes, resultados de teste, dados, requisitos, incidentes, defeitos, etc., e para relatórios e monitoração da execução de teste.
3) Ferramentas que são usadas em reconhecimento, ou, em termos mais simples, exploração (exemplo: ferramentas que monitoram atividades de arquivos para uma aplicação).
4) Quaisquer ferramentas que auxiliam no teste (uma planilha Excel também é uma ferramenta de teste, neste contexto.).

Ferramentas de suporte ao teste podem ter um ou mais dos seguintes objetivos, dependendo do contexto:

- Aumentar a eficiência das atividades de teste, através da automação de tarefas repetitivas ou dar suporte a atividades de teste manuais, como planejamento, modelagem, relatório e monitoração.
- Automatizar atividades que requerem recursos significativos quando executadas manualmente (exemplo: teste estático).
- Automatizar atividades que não podem ser executadas manualmente (ex.: teste de performance em larga escala de aplicações cliente-servidor).
- Aumentar a confiabilidade do teste (exemplo: pela automação de comparações em larga escala ou simulação de comportamento).

O termo “frameworks de teste” também é frequentemente utilizado na indústria, em pelo menos três sentidos:

- Bibliotecas de teste reutilizáveis e extensivas que podem ser utilizadas para construir ferramentas de teste (chamadas de preparação de teste também)
- Um tipo de design de automação de teste (ex.: *data-driven*, *context-driven*)
- O processo de execução de teste como um todo

Para o objetivo deste *syllabus*, o termo “frameworks de teste” é utilizado em seus dois primeiros significados, como descrito na Seção 6.1.6.

6.1.2 Classificação das Ferramentas de Teste (K2)

Há várias ferramentas que suportam diferentes aspectos do teste. Ferramentas podem ser classificadas baseadas em vários critérios como objetivo, comercial/livre/código aberto/shareware, tecnologia utilizada e daí por diante. Ferramentas são classificadas neste *syllabus* de acordo com as atividades de teste que elas suportam.

Algumas ferramentas suportam claramente apenas uma atividade; outras podem suportar mais que uma, mas estão classificadas sob a atividade de que ela está mais associada. Ferramentas de um único fornecedor, especialmente aquelas que foram planejadas para atuarem em conjunto, podem ser agrupadas em um único pacote.

Alguns tipos de ferramentas de teste são intrusivos, ou seja, eles próprios podem afetar o resultado do teste. Por exemplo, o resultado de uma aferição de tempo pode ser diferente dependendo de como é a medição em diferentes ferramentas de performance, ou então pode-se ter uma medida diferente na cobertura do código dependendo da ferramenta que se utiliza. A consequência de uma ferramenta intrusiva é conhecida como efeito de monitoração.

Algumas ferramentas oferecem suporte mais apropriado para desenvolvedores (durante teste de componente e integração dos componentes). Estas são identificadas com um "(D)" na classificação que segue abaixo.

6.1.3 Ferramentas para gerenciamento do teste (K1)

As ferramentas de gerência aplicam-se a todas as atividades do teste sobre o ciclo de vida do software

Ferramenta de Gerenciamento de Teste

Estas ferramentas fornecem interfaces para a execução de teste, rastreamento de defeitos e gestão de requisitos juntamente com suporte para análise quantitativa e relatório dos objetos de teste. Elas também suportam o rastreamento dos objetos de teste às especificações de requisitos e podem ter uma capacidade de controle de versão independente ou uma interface a um controle de versão externo.

Ferramentas de Gerenciamento de Requisitos

Estas ferramentas armazenam termos de requisitos, armazenam os atributos para os requisitos (incluindo prioridade), fornecem identificadores únicos e dão suporte ao rastreamento dos requisitos aos testes individuais. Estas ferramentas podem também ajudar a identificar requisitos inconsistentes ou ausentes.

Ferramentas de Gerenciamento de Incidentes (Ferramenta de Rastreamento de Defeitos)

Estas ferramentas armazenam e gerenciam relatórios de incidentes (ex.: defeitos, falhas, problemas e anomalias) e ajudam no gerenciamento do ciclo de vida de incidentes, opcionalmente com suporte para análise estatística.

Ferramentas de Gerenciamento de Configuração

Embora não são estritamente ferramentas de teste, mas são necessárias para manter o controle da versão de diferentes builds de softwares e testes, especialmente quando é configurada mais que um ambiente de hardware/software em termos de versões de sistemas operacionais, compiladores, browsers, etc.

6.1.4 Ferramentas de suporte para teste estático (K1)

Ferramentas para testes estático fornecem uma alternativa de baixo custo para encontrar mais defeitos em um estágio inicial dentro do processo de desenvolvimento.

Ferramentas para suporte ao processo de revisão

Essas ferramentas auxiliam com o processo de revisão, check-lists, diretrizes de revisão e são utilizadas para armazenar e comunicar comentários de revisão e fornecer relatórios de defeitos e esforço associado. Podem ser úteis também no fornecimento de ajuda para revisões online para times grandes ou que estão distantes geograficamente.

Ferramentas de Análise Estática (D)

Ferramentas de análise estática ajudam os desenvolvedores, testadores e os envolvidos com a qualidade a encontrar defeitos antes dos testes dinâmicos, através da aplicação de padrões de codificação (incluindo

código seguro), além de análise de estrutura e dependências. Podem ser úteis também no planejamento ou análise de risco devido ao fornecimento de métricas para o código (ex.: complexidade).

Ferramentas de Modelagem (D)

Ferramentas de modelagem validam o modelo do software (ex.: modelo físico de dados, para um banco de dados relacional), enumerando inconsistência e encontrando defeitos. Essas ferramentas podem geralmente ajudar na geração de alguns casos de testes baseados no modelo.

6.1.5 Ferramenta de suporte para especificação de teste(K1)

Ferramenta de Modelagem de Teste

Ferramentas de modelagem de teste geram entradas de teste ou testes a partir dos requisitos, de uma interface gráfica do usuário, dos diagramas de modelagem (estado, dados ou objetos) ou do código.

Ferramentas para preparação de dados de teste (D)

Estas ferramentas manipulam a base de dados, arquivos ou transmissão de dados para ajudar na preparação dos dados de teste a serem utilizados durante a execução de testes, além de garantir a segurança através do anonimato dos dados.

6.1.6 Ferramenta de suporte para execução e registro (K1)

Ferramentas de execução do teste

Ferramentas de execução do teste permitem que o teste seja executado automaticamente, ou semi-automaticamente, usando dados de entradas e resultados esperados através do uso de uma linguagem de script, e geralmente fornecem um registro de teste para cada teste executado. Também podem ser usadas para registrar testes, e geralmente suportam linguagens de script ou parametrização de dados baseados em telas e outras customizações do teste.

Ambiente preparado/Ferramentas framework de teste de unidade (D)

O Ambiente preparado e frameworks facilitam o teste de componente ou parte de um sistema por simular o ambiente em que o objeto de teste será executado, através do fornecimento de simuladores, como stubs ou drivers.

Comparadores de teste

Comparadores de teste determinam as diferenças entre os arquivos, banco de dados ou resultados dos testes. As ferramentas de execução normalmente incluem comparações dinâmicas, mas comparação após a execução pode ser feita por uma ferramenta especificamente de comparação. Um comparador de teste pode ser usado como oráculo de teste, especialmente se for automatizado.

Ferramentas de medição de cobertura (D)

Ferramenta de medição de cobertura pode ser intrusiva ou não intrusiva, e medem a porcentagem de estruturas de código que são exercitados (comandos, decisões, ramos, módulos e chamada de funções) por um dado conjunto de testes.

Ferramentas de Segurança

Ferramentas de segurança são utilizadas para avaliar as características de segurança do software. Isso inclui a avaliação da habilidade do software em proteger confidencialidade, integridade, autenticação, autorização, disponibilidade e não repúdio de dados. Ferramentas de segurança são principalmente focadas em uma plataforma específica de tecnologia e seu objetivo.

6.1.7 Ferramenta de performance e monitoração (K1)

Ferramenta de Análise dinâmica (D)

Estas ferramentas encontram defeitos que só são evidenciados quando o software está em execução, assim como dependência de tempo ou vazamento de memória. São normalmente utilizados em testes de componentes, testes de integração dos componentes e teste de middleware.

Ferramentas de Teste de Performance/Teste de Carga/Teste de Estresse

Ferramenta de teste de performance monitoram e relatam como um sistema se comporta sob uma variedade de condições simuladas, em termos de número de usuários concorrentes, seu padrão ramp-up, frequência e porcentagem relativa de transações. A simulação de carga é conseguida através da criação de usuários virtuais que executam um conjunto selecionado de transações, espalhados através de várias máquinas de testes normalmente chamadas de geradores de carga.

Ferramentas de monitoração

Ferramentas de monitoração continuamente analisam, verificam e reportam a respeito do uso de recursos específicos do sistema, e fornecem avisos de possíveis problemas do serviço.

6.1.8 Ferramenta de suporte para áreas de aplicações específicas (K1)

Avaliação de qualidade de dados

Dados estão no centro de alguns projetos, como aqueles de conversão/migração de dados e aplicações como *data warehouse* e seus atributos podem variar em termos de criticidade e volume. Em tais contextos, ferramentas precisam ser empregadas para avaliação da qualidade dos dados para revisar e verificar a conversão de dados e regras de migração, para garantir que os dados processos estão corretos, completos e aderentes com padrões pré-definidos específicos ao contexto.

Outras ferramentas de teste existem para teste de usabilidade.

6.2 Uso Efetivo das Ferramentas: Riscos e Benefícios em potenciais (K2)	20 minutos

Termos

Teste orientado a dados, Teste orientado a palavras-chaves, linguagem de scripts.

6.2.1 Potenciais benefícios e riscos de ferramentas de suporte ao teste (K1)

A simples compra da ferramenta não significa o sucesso da mesma. Cada tipo de ferramenta pode requerer um esforço para atingir os benefícios reais e contínuos. Há oportunidades d benefícios potenciais com o uso de ferramentas de teste, mas também existem riscos:

Benefícios potenciais do uso das ferramentas incluem:

- Reduzir trabalhos repetitivos (executar os testes de regressão, entrar os mesmos dados do teste e checar a padronização do código).
- Maior consistência e possibilidade de repetições (ex.: testes executados por uma ferramenta e testes derivados de requisitos).
- Avaliação dos objetivos (ex.: medidas estáticas, cobertura e comportamento do sistema)
- Facilidade do acesso a informações sobre o teste (ex.: estatísticas e gráficos referente ao progresso de teste, taxas de incidente e performance).

Riscos no uso das ferramentas são:

- Expectativas falsas referentes à ferramenta (incluindo funcionalidades e facilidade de uso).
- Subestimação do tempo, custo e esforço necessário para a introdução inicial da ferramenta (incluindo treinamento e expertise externo).
- Subestimação do esforço e tempo necessários para obter benefícios significantes e contínuos do uso da ferramenta (incluindo a necessidade para mudanças no processo de teste e melhoria contínua na forma como a ferramenta é utilizada).
- Subestimação do esforço para manter os artefatos de teste gerados pela ferramenta.
- Confiança excessiva na ferramenta (quando o emprego de teste manual e modelagem de teste são mais recomendados).
- Negligência no controle de versão de ativos de teste dentro da ferramenta.
- Negligência na manutenção dos relacionamentos e questões de interoperabilidade entre ferramentas críticas, como ferramentas de gestão de requisitos, controle de versão, gerenciamento de incidente, rastreamento de defeitos e ferramentas de múltiplos fornecedores.
- Risco do fornecedor da ferramenta abandonar o negócio, aposentar a ferramenta, ou vender a ferramenta para outro fornecedor.
- Baixa capacidade do fornecedor para suporte, upgrades e correções de defeitos.
- Risco de suspensão de projetos de código aberto e ferramentas livres.
- Imprevistos, como a falta de habilidade para dar suporte a uma nova plataforma.

6.2.2 Considerações especiais para alguns tipos de ferramentas (K1)

Ferramentas de execução de testes

Ferramentas de execução de testes executam os scripts desenvolvidos para implementar testes que foram armazenados eletronicamente. Este tipo de ferramenta muitas vezes requer um esforço significativo para obter os benefícios.

Capturar testes pela gravação de ações manuais do usuário pode parecer atrativo, mas esta abordagem não tem uma escalabilidade. Um script de captura é uma representação linear com dados e ações específicas como parte de cada script. Este tipo de script pode se tornar instável a eventos inesperados.

Uma abordagem orientada a dados separa os dados de entrada dos testes, normalmente em diversas planilhas contendo variações destes dados, utilizando scripts mais genéricos que possam ler estas planilhas na execução de um mesmo teste fazendo com que os dados utilizados sejam diferentes. Os testadores que não tem familiaridade com a linguagem dos scripts podem entrar com os dados para estes scripts pré-definidos.

Em uma abordagem orientada a palavras-chave, as planilhas contêm palavras-chaves que descrevem ações a serem tomadas e os dados do teste. Testadores (mesmo que não tenham familiaridade com a linguagem de scripts) podem então definir os testes utilizando estas palavras-chaves, que poderá ser montada para o aplicativo que será testado.

Um técnico especialista na linguagem do script será necessário para todos os casos (ainda que como testador ou como especialista em automação).

Para qualquer técnica de script utilizada, os resultados de cada teste precisam ser armazenados para futuras comparações.

Ferramentas de análises estáticas

Ferramentas de análises estáticas aplicadas ao código fonte podem forçar a padronização do código. No entanto se aplicada a um código existente (sistemas legado) pode gerar grande quantidade de mensagens. Mensagens de atenção não param o processo de compilação, mas devem ser tratados para que a manutenção do código seja mais fácil no futuro. Uma implementação gradual com filtros iniciais para excluir algumas mensagens poderá ser uma boa e eficiente alternativa.

Ferramentas de gerenciamento de teste

Ferramentas de gerenciamento de teste precisam fazer a interface com outras ferramentas ou planilhas para produzir as informações no melhor formato possível de modo a atender as necessidades da empresa. O relatório precisa ser bem desenhado e monitorado para que traga o benefício esperado.

6.3 Implementando uma Ferramenta na Organização (K1)	15 minutos

Termos

Não há

Conceito

Os princípios para introduzir uma ferramenta em uma organização são:

- Avaliação da maturidade da organização, pontos fracos e pontos fortes na identificação de oportunidades para um aperfeiçoamento do processo de teste com uso de uma ferramenta.
- Avaliação frente a requisitos claros e os critérios objetivos.
- Uma prova de conceito, utilizando a ferramenta durante a fase de avaliação para estabelecer se ela funciona adequadamente com o software sendo testado e dentro da infraestrutura atual ou para identificar mudanças necessárias à infraestrutura para efetivamente utilizar a ferramenta.
- Avaliação do fornecedor (incluindo treinamento, suporte e aspectos comerciais) ou fornecedores de suporte ao serviço em caso de ferramentas não comerciais.
- Identificação dos requisitos internos para acompanhamento (*mentoring* e *coaching*) no uso da ferramenta.
- Avaliação das necessidades de treinamento, considerando as habilidades de automação de teste da equipe.
- Estimativa de uma razão custo-benefício baseado em um *business case* concreto.

Introduzir a ferramenta seleciona em uma organização se inicia com um projeto piloto, que tem os seguintes objetivos:

- Aprender mais detalhes sobre a ferramenta.
- Ver como a ferramenta poderá se adequar aos processos e práticas e como necessitariam mudar.
- Decidir as formas e padrões de uso, gerenciamento, arquivamento e manutenção da ferramenta e artefatos de testes (ex.: decidir a convenção dos nomes de arquivos, criação de bibliotecas e decidir a modularidade dos grupos de teste).
- Estimar se os benefícios serão atingidos a um custo razoável.

Os fatores de sucesso para a implantação de uma ferramenta em uma organização incluem:

- Implementar a ferramenta ao restante da organização incrementalmente.
- Adaptar e aprimorar o processo para combinar com o uso da ferramenta.
- Providenciar treinamentos para novos usuários.
- Definir diretrizes de utilização.
- Implementar um modo de aprender lições com o uso da ferramenta.
- Fornecer suporte ao time de teste para uma determinada ferramenta.
- Monitorar o uso e os benefícios da ferramenta.

Referências

6.2.2 Buwalda, 2001, Fewster, 1999
6.3 Fewster, 1999

7. Referências

Padrões

- ISTQB Glossário de termos usados no Teste de Software Versão 2.1
- [CMMI] Chrissis, M.B., Konrad, M. and Shrum, S. (2004) *CMMI, Guidelines for Process Integration*
- *and Product Improvement*, Addison Wesley: Reading, MA - Ver seção 2.1
- [IEEE 829-1998] IEEE Std 829™ (1998) IEEE Standard for Software Test Documentation) - Ver seções 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6
- [IEEE 1028] IEEE Std 1028™ (2008) IEEE Standard for Software Reviews and Audits- Ver seção 3.2
- [IEEE 12207] IEEE 12207/ISO/IEC 12207-1996, Software life cycle processes - Ver seção 2.1
- [ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality - Ver seção 2.3

Livros

- [Beizer, 1990] Beizer, B. (1990) Software Testing Techniques (2nd edition), Van Nostrand Reinhold: Boston - Ver seções 1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6
- [Black, 2001] Black, R. (2001) Managing the Testing Process (2nd edition), John Wiley & Sons: NewYork - Ver seções 1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6
- [Buwalda, 2001] Buwalda, H. et al. (2001) Integrated Test Design and Automation, Addison Wesley: Reading, MA - Ver seção 6.2
- [Copeland, 2004] Copeland, L. (2004) A Practitioner's Guide to Software Test Design, Artech House: Norwood, MA - Ver seções 2.2, 2.3, 4.2, 4.3, 4.4, 4.6
- [Craig, 2002] Craig, Rick D. and Jaskiel, Stefan P. (2002) Systematic Software Testing, Artech House: Norwood, MA - Ver seções 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4
- [Fewster, 1999] Fewster, M. and Graham, D. (1999) Software Test Automation, Addison Wesley: Reading, MA - Ver seções 6.2, 6.3
- [Gilb, 1993]: Gilb, Tom and Graham, Dorothy (1993) Software Inspection, Addison Wesley: Reading, MA - Ver seções 3.2.2, 3.2.4
- [Hetzel, 1988] Hetzel, W. (1988) Complete Guide to Software Testing, QED: Wellesley, MA - Ver seções 1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 4.1, 5.1, 5.3
- [Kaner, 2002] Kaner, C., Bach, J. and Pettticord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons - Ver seções 1.1, 4.5, 5.2
- [Myers 1979] Myers, Glenford J. (1979) The Art of Software Testing, John Wiley & Sons - Ver seções 1.2, 1.3, 2.2, 4.3
- [van Veenendaal, 2004] van Veenendaal, E. (ed.) (2004) The Testing Practitioner (Chapters 6, 8, 10), UTN Publishers: The Netherlands - Ver seções 3.2, 3.3

Apêndice A – Syllabus Background

Histórico deste documento

Este documento foi preparado durante os anos de 2004 e 2011 por um trabalho composto por membros apontados pelo ISTQB. Foi inicialmente revisado por uma equipe selecionada e então por representantes da comunidade internacional de teste de software. As regras usadas na produção deste documento são exibidas no Apêndice C.

Este documento é o *syllabus* para a Certificação Internacional de Teste de Software em Nível Fundamental, o primeiro nível de qualificação internacional aprovado pelo ISQTB (www.istqb.org).

Objetivos da Qualificação Certificado em Nível Fundamental

- Obter reconhecimento do teste como uma especialização profissional e essencial da engenharia de software.
- Prover um *framework* padrão para desenvolvimento da carreira dos testadores.
- Capacitar testadores qualificados profissionalmente a serem reconhecidos por empresários, clientes e colegas, e criar um perfil do testador.
- Promover boas e consistentes práticas de teste com todas as disciplinas de engenharia de software
- Identificar os tópicos do teste que são relevantes e o valor para o mercado.
- Capacitar os fornecedores de software a contratar testadores certificados e através disso obter vantagens comerciais sob seus competidores por propagar suas próprias políticas de recrutamento de testadores.
- Promover uma oportunidade para os testadores e aqueles interessados no teste, a adquirir uma qualificação internacionalmente reconhecida no assunto.

Objetivos da Qualificação Internacional (adaptado da reunião do ISTQB em Sollentuna, Novembro de 2001)

- Estar apto a comparar a prática do teste entre os diferentes países.
- Capacitar testadores a trocarem conhecimento entre as comissões mais facilmente.
- Permitir que projetos multinacionais/internacionais tenham uma compreensão comum do empreendimento do teste.
- Aumentar o número de testadores qualificados ao redor do mundo.
- Ter mais impacto como uma iniciativa baseada internacionalmente do que qualquer abordagem de um país específico.
- Desenvolver um corpo comum internacional de compreensão e conhecimento sobre teste e terminologias através do *syllabus* e aumentar o nível de conhecimento sobre teste para todos os participantes.
- Promover o teste como uma profissão em mais países.
- Capacitar testadores a obter uma qualificação reconhecida na sua linguagem nativa.
- Permitir o compartilhamento de conhecimentos e recursos entre os países.
- Prover o reconhecimento internacional dos testadores e desta qualificação devido à participação oriunda de muitos países.

Requisitos básicos para esta qualificação

O critério de entrada para fazer o exame do ISTQB (Certificado dos Fundamentos em Teste de Software) é que o candidato tenha um interesse em teste de software. No entanto, é altamente recomendável que o candidato também:

- Tenha um conhecimento mínimo seja em desenvolvimento de software ou teste de software, como seis meses de experiência como um testador em testes de sistema ou de aceite, ou como desenvolvedor de software.
- Fazer o curso que é autorizado pelos padrões do ISTQB (por uma comissão nacional reconhecida).

Histórico da Certificação em Fundamentos em Teste de Software

A certificação independente de testadores de software começou no Reino Unido com a *British Computer Society´s Information Systems Examination Board* (ISEB), quando uma comissão de teste de software surgiu em 1998 (www.bcs.org.uk/iseb). Em 2002, ASQF na Alemanha começou a dar suporte a um esquema de qualificação ao testador (www.asqf.de). Este *syllabus* é baseado no ISEB e ASQF syllabi; inclui um conteúdo organizado, atualizado e novo, e a ênfase é dada aos tópicos que fornecem ajuda mais prática aos testadores.

Um Certificado em Fundamentos em Teste de Software (ex.: ISEB, ASQF ou outra organização reconhecida por uma comissão nacional do ISQTB) obtido por alguém, antes do Certificado Internacional ter sido criado, será considerado equivalente ao certificado internacional. A Certificação em Fundamentos de Teste de Software não expira e não precisa ser renovado. A data em que foi obtido será demonstrada no Certificado.

Com a participação de cada país, aspectos locais são controlados pela comissão nacional reconhecida pelo ISTQB. As responsabilidades da comissão nacional são especificadas pelo ISTQB, mas são implementadas em cada país. Dentre as obrigações das comissões dos países estão a autorização para prover treinamentos e a ministrar os exames.

Apêndice B – Objetivos de estudo/Níveis de Conhecimento

Os seguintes objetivos de estudos são definidos como aplicáveis para este *syllabus*. Cada tópico neste *syllabus* será examinado de acordo com os seus objetivos de estudo.

Nível 1: Relembrar (K1)

O candidato reconhecerá, relembrará e retomará um termo ou conceito.

Palavras chave: Relembrar, recuperar, reconhecer, saber.

Exemplo:

Ser capaz de reconhecer a definição de "falha" como:

- "Não entrega de serviço a um usuário final ou qualquer outro interessado (*stakeholder)" ou*
- "Desvios atuais do componente ou sistema de suas expectativas de entrega, serviços ou resultados".

Nível 2: Compreender (K2)

O candidato pode selecionar as razões ou explicações para declarações relacionadas ao tópico podendo resumir, comparar, classificar e dar exemplos para os conceitos de teste.

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

Exemplo:

Ser capaz de explicar as razões porque os testes devem ser elaborados o mais cedo possível:

- Encontrar defeitos quando eles podem ser removidos a um preço baixo.
- Encontrar primeiramente os defeitos mais importantes.

Ser capaz de explicar as semelhanças e diferenças entre teste de integração e de sistema:

- Semelhanças: testar mais do que um componente, e poder testar aspectos não funcionais.
- Diferenças: teste de integração concentrado nas interfaces e interações e teste de sistemas se concentram nos aspectos do sistema como um todo, como o processamento do início ao fim.

Nível 3: Aplicar (K3)

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

Palavras chave: Implementar, executar, usar, seguir um procedimento, aplicar um procedimento.

Exemplo:

- Ser capaz de identificar os valores limites para partições válidas e inválidas.
- Ser capaz de selecionar os casos de teste de um dado diagrama de transição de estados de maneira a cobrir todas as transições.

Nível 4: Analisar (K4)

O candidato pode separar a informação relacionada a um procedimento ou técnica em suas partes constituintes para melhor entendimento, e pode distinguir entre fatos e inferências. A aplicação típica é uma análise de documento, software ou situação de projeto e propor ações adequadas para resolver um problema ou tarefa.

Palavras chave: Analisar, organizar, encontrar coerência, integrar, esboçar, estruturar, atribuir, desconstruir, diferenciar, discriminar, distinguir, focar, selecionar.

Exemplo:

- Avaliar riscos de produto e propor atividades de mitigação preventivas e corretivas.
- Descrever quais porções de um relatório de incidente são factuais e quais são inferidas a partir de resultados.

Referências

(Para os níveis cognitivos dos objetivos de estudo)

Anderson, L. W. and Krathwohl, D. R. (eds) (2001) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon:

Apêndice C – Regras aplicadas ao ISTQB Fundation Syllabus

Foundation Syllabus

As regras listadas aqui foram usadas no desenvolvimento e revisão do *syllabys*. (Uma "LEGENDA" é exibida após cada regra como abreviação da mesma).

Regras gerais

SG1 O *syllabus* deve ser compreensível e absorvível por pessoas com 0 a 6 meses (ou mais) de experiência em teste. (6-MESES)

SG2 O *syllabus* é mais prático do que teórico. (PRÁTICO)

SG3 O *syllabus* é claro e objetivo (não tem ambiguidades) aos leitores interessados. (CLARO)

SG4 O *syllabus* é compreensível a pessoas de diferentes países e, é de fácil tradução em diferentes linguagens. (TRADUZÍVEL)

SG5 O *syllabus* original usa o inglês americano. (INGLÊS-AMERICANO)

Conteúdo Atual

SC1 O *syllabus* inclui conceitos recentes de testes e reflete as melhores práticas da atualidade em teste de software, quando estas forem adotadas de forma geral. O *syllabus* está sujeito à revisão de três a cinco anos (ATUALIZADO).

SC2 O *syllabus* deve minimizar aspectos relacionados a tempo, assim como as condições atuais do mercado para permitir possua uma vida útil de três a cinco anos. (VIDA UTIL)

Objetivos de Estudos

LO1 Objetivos de estudos distinguem os itens a ser reconhecidos/relembrados (nível cognitivo K1), itens que o candidato deve compreender conceitualmente (K2) e aqueles que os candidatos estão aptos a praticar/utilizar (K3). (NÍVEL DE CONHECIMENTO)

LO2 A descrição do conteúdo é consistente com os objetivos de estudo. (CONSISTÊNCIA)

LO3 Para ilustrar os objetivos de aprendizado, questões simuladas de exame deverão ser apresentadas para as seções principais do *syllabus*. (EXAME)

Estrutura geral

ST1 A estrutura do *syllabus* é clara e permite o cruzamento de referência e de uma parte a outra das questões dos exames e de outros documentos relevantes. (REFERÊNCIAS CRUZADAS).

ST2 Sobreposição entre seções do *syllabus* é minimizado. (SOBREPOSIÇÃO)

ST3 Cada seção do *syllabus* tem a mesma estrutura. (ESTRUTURA CONSISTENTE)

ST4 O *syllabus* contém versão, data de emissão e número de cada página. (VERSÃO)

ST5 O *syllabus* contempla um guia que aponta a quantidade de tempo a ser gasta em cada seção (para refletir a importância relativa de cada tópico). (TEMPO GASTO)

Referências

SR1 Origens e referências dos conceitos são dadas no *syllabus* para ajudar os instrutores a encontrar mais informações sobre o tópico. (REFS)

SR2 Quando não existir fontes claras ou identificáveis, mas detalhes deveram ser fornecidos no *syllabus*. Por exemplo, definições estão no Glossário, sendo assim apenas os termos serão listados no *syllabus*. (REFS SEM DETL)

Fontes e Informações

Os termos utilizados no *syllabus* são definidos no Glossário do ISTQB que é usado em teste de software. Uma versão do glossário é disponibilizada pelo ISTQB. A tradução poderá ser encontrada no site do BSTQB (www.bstqb.org.br)

Uma lista de livros sobre teste de software é recomendada para estudos em paralelo com este *syllabus*. A principal lista de livros é parte da seção *Referências*.

Apêndice D – Observação aos provedores de treinamentos.

Cada cabeçalho dos temas do *syllabus* é assinalado com o tempo a ser alocado em minutos. O propósito disto é dar uma visão da relativa proporção de tempo a ser alocado em cada seção do curso, e dar um tempo mínimo aproximado para ensinar cada seção. Instrutores podem gastar mais tempo do que é indicado e os candidatos também podem gastar mais tempo em leitura e pesquisa. Um programa de treinamento não precisa seguir a mesma ordem que o *syllabus*.

O *syllabus* contém referências para padrões estabelecidos, que devem ser usados na preparação do material para o treinamento. Cada padrão utilizado deve estar com a versão mencionada na versão atual do *syllabus*. Outras publicações, *templates* ou padrões não referenciados neste *syllabus* podem também ser utilizados e referenciados, mas não fará parte do exame.

Todos os objetivos de estudo K3 e K4 requerem um exercício prático a ser incluído nos materiais de treinamento.

Appendix E – Release Notes

Release 2010

1) Mudança dos objetivos de estudo (LO) incluem alguma clarificação:
   a) Mudança de termos para os seguintes LOs (o conteúdo e nível do LO permanecem inalterados): LO-1.2.2, LO-1.3.1, LO-1.4.1, LO-1.5.1, LO-2.1.1., LO-2.4.2, LO-4.1.3, LO-4.2.1, LO-4.2.2., LO-4.3.1, LO-4.3.2, LO-4.3.3, LO-4.4.1, LO-4.4.2, LO-4.4.3, LO-4.6.1, LO-5.1.2, LO-5.2.2, LO-5.3.2, LO-5.3.3, LO-5.5.2, LO-6.1.1, LO-6.2.2, LO-6.3.2.
   b) LO-1.1.5 foi atualizado e alterado para K2. Pois uma comparação dos termos relacionados a defeito pode ser esperada.
   c) LO-1.2.3 (K2) foi adicionado. O conteúdo já estava coberto no syllabus 2007.
   d) LO-3.1.3 (K2) agora combina o conteúdo do LO-3.1.3 e LO-3.1.4.
   e) LO-3.1.4 foi removido do *syllabus* 2010, já que é parcialmente redundante com o LO-3.1.3.
   f) LO-3.2.1 foi atualizado para consistência com o conteúdo do syllabus 2010.
   g) LO-3.3.2 foi modificado, e seu nível foi alterado de K1 para K2, para consistência com o LO-3.1.2.
   h) LO 4.4.4 foi modificado para maior clareza, e foi mudado de K3 para K4. Razão: o LO-4.4.4 já havia sido escrito em uma forma K4.
   i) LO-6.1.2 (K1) foi retirado do *syllabus* 2010 e substituído pelo LO-6.1.3 (K2). Não há LO-6.1.2 no *syllabus* 2010.
2) Uso consistente da abordagem de teste de acordo com a definição do glossário. O termo estratégia de teste não é mais definido como um termo a ser relembrado.
3) O Capítulo 1.4 agora contém o conceito de rastreabilidade entre base de teste e casos de teste.
4) Capítulo 2.x agora contém objetos de teste e bases de teste.
5) Reteste é agora o principal termo no glossário, ao invés de teste de confirmação.
6) O aspecto qualidade e teste de dados foram adicionados em várias partes no *syllabus*: qualidade de dados e riscos nos Capítulos 2.2, 5.5 e 6.1.8.
7) O Critério de Entrada, Capítulo 5.2.3, foi adicionado como um novo subcapítulo. Razão: consistência com o critério de saída (-> critério de entrada adicionado ao LO-5.2.9).
8) Uso consistente dos termos estratégia de testes e abordagem de testes com sua definição no glossário.
9) O Capítulo 6.1 foi diminuído, pois a descrição das ferramentas estava longa demais para uma aula de 45 minutos.
10) O IEEE Std 829:2008 foi lançado. Esta versão do *syllabus* não considera ainda esta nova edição. A seção 5.2 se refere ao documento Plano de Teste Mestre. O conteúdo do Plano de Teste Mestre é coberto pelo conceito de que o documento "Plano de Teste" cobre diferentes níveis de planejamento: Planos de Teste para os níveis de teste podem ser criados, assim como um plano de teste no nível de projeto cobrindo múltiplos níveis de teste. Este último é chamado de Plano de Teste Mestre neste *syllabus* e no glossário ISTQB.
11) O Código de Ética foi movido do CTAL para o CTFL.

Release 2011

Mudanças feitas no "release de manutenção" 2011:

1) Primeira ocorrência: ISTQB substituído por ISTQB®.
2) Introdução neste *syllabus*: Descrição dos níveis cognitivos de conhecimento removidos, pois isto estava redundante com o Apêndice B.
3) Seção 1.6: Sendo que o intuito não era definir um Objetivo de Estudo para o "Código de Ética", o nível cognitivo para a seção foi removido.
4) Seções 2.2.1, 2.2.2, 2.2.3 e 2.2.4, 3.2.3: Correção de formatação em listas.
5) Seção 2.2.2. A palavra falha não estava correta de acordo com a descrição: "... isolar falhas em um componente específico...". Portanto, foi substituída com a expressão "defeito" para a mesma sentença.
6) Seção 2.3: Formatação corrigida da lista de tópicos de objetivos de teste relacionados aos termos de teste na seção Tipos de Teste (K2).

7) Seção 2.3.4: Descrição atualizada de *debugging* para ser consistente com a Versão 2.1 do Glossário ISTQB.
8) Seção 3.2.1: Devido ao fato das atividades de uma revisão formal terem sido incorretamente formatadas, o processo de revisão tinha 12 atividades principais ao invés de 6, conforme previsto. Portanto, foi alterado de novo para seis, o que torna essa seção aderente ao Syllabus 2007 e o ISTQB Advanced Level Syllabus 2008.
9) Seção 4.2: Mudança de texto para clarificar como o teste caixa-preta e caixa-branca podem ser usados em conjunto com técnicas baseadas na experiência.
10) Seção 4.3.5: caminho alternativo substituído por cenário alternativo.
11) Seção 4.4.2: Para clarear o termo "teste de desvio" no texto da Seção 4.4, uma sentença para clarificar o foco do teste de desvio foi alterado.
12) Seção 6.1: Tópico "6.1.1 Entendendo o Significado e Objetivo do Suporte a Ferramentas para Teste" (K2) substituído por "6.1.1. Suporte a Ferramentas para Teste (K2)".
13) Seção 7 / Livros: A terceira edição de [Black, 2001] listada, substituindo a segunda edição.
14) Apêndice D: Capítulos requerendo exercícios foram substituídos pelos requisitos genéricos informando que todos Objetivos de Estudo K3 e superiores requerem exercícios. Este é um requisito especificado no Processo de Certificação ISTQB (Versão 1.26).
15) Apêndice E: os objetivos de estudo alterados entre as versões 2007 e 2010 estão agora listados corretamente.

Apêncide F – Índice Remissivo