International Software Testing Qualifications Board

Certified Tester

Advanced Level Syllabus

Test Analyst (TA)

Versão 2012br

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	Observações
ISEB v1.1	04/09/2001	*ISEB Practitioner Syllabus*.
ISTQB 1.2E	Setembro de 2003	*ISTQB Advanced Level Syllabus* do Grupo de *Software* da Organização Europeia para a Qualidade (EOQ-SG, na sigla em inglês).
V2007	12/10/2007	Versão de 2007 do *Certified Tester Advanced Level Syllabus*.
D100626	26/06/2010	Incorporação das alterações aceitas em 2009 e divisão de capítulos por módulos diferentes.
D101227	27/12/2010	Aceitação de alterações no formato e correções que não afetam o significado das frases.
D2011	23/10/2011	Divisão do *syllabus*, revisão dos objetivos de aprendizagem e alterações no texto para manter a coerência com os objetivos de aprendizagem. Acréscimo de resultados comerciais.
Alfa 2012	09/03/2012	Incorporação de todos os comentários das comissões nacionais recebidos em função do lançamento da versão de outubro.
Beta 2012	07/04/2012	Incorporação de todos os comentários das comissões nacionais recebidos em função do lançamento da versão alfa.
Beta 2012	07/04/2012	Envio da versão beta à Assembleia Geral.
Beta 2012	08/06/2012	Envio da versão revisada às comissões nacionais.
Beta 2012	27/06/2012	Incorporação de comentários do EWG e glossário.
RC 2012	15/08/2012	Lançamento de rascunho – inclusão de edições finais das comissões nacionais.
GA 2012	19/10/2012	Edições e correções finais para lançamento por parte da Assembleia Geral.
2012br	05/01/2013	Tradução para a Língua Portuguesa

Índice

Agradecimentos

O presente documento foi elaborado pelo grupo principal do subgrupo de trabalho de nível avançado da International Software Testing Qualifications Board – Advanced Test Analyst (TA) composto por Judy McKay (presidenta), Mike Smith e Erik Van Veenendaal.

O grupo principal gostaria de agradecer à equipe de revisão e às comissões nacionais por suas sugestões e contribuições.

Quando o Advanced Level Syllabus foi concluído, o Grupo de Trabalho de Nível Avançado contava com os seguintes integrantes (em ordem alfabética):

Graham Bath, Rex Black, Maria Clara Choucair, Debra Friedenberg, Bernard Homès (vice-presidente), Paul Jorgensen, Judy McKay, Jamie Mitchell, Thomas Mueller, Klaus Olsen, Kenji Onishi, Meile Posthuma, Eric Riou du Cosquer, Jan Sabak, Hans Schaefer, Mike Smith (presidente), Geoff Thompson, Erik van Veenendaal, Tsuyoshi Yumoto.

As seguintes pessoas revisaram, comentaram e escolheram este syllabus: Graham Bath, Arne Becher, Rex Black, Piet de Roo, Frans Dijkman, Mats Grindal, Kobi Halperin, Bernard Homès, Maria Jönsson, Junfei Ma, Eli Margolin, Rik Marselis, Don Mills, Gary Mogyorodi, Stefan Mohacsi, Reto Mueller, Thomas Mueller, Ingvar Nordstrom, Tal Pe'er, Raluca Madalina Popescu, Stuart Reid, Jan Sabak, Hans Schaefer, Marco Sogliani, Yaron Tsubery, Hans Weiberg, Paul Weymouth, Chris van Bael, Jurian van der Laar, Stephanie van Dijk, Erik van Veenendaal, Wenqiang Zheng e Debi Zylbermann.

A Assembleia Geral do ISTQB® lançou este documento formalmente no dia 19 de outubro de 2012.

0. Introdução ao syllabus

0.1 Objeto do documento

O presente *syllabus* forma a base para a qualificação internacional de testes de *software* no nível avançado para o *Test Analyst (TA)*. O ISTQB® fornece o *syllabus*:

1. Às comissões nacionais a fim de traduzi-lo para o idioma local e com a finalidade de credenciar os provedores de treinamento.
2. As comissões nacionais podem adaptar o *syllabus* às suas necessidades linguísticas específicas e modificar as referências para adaptá-las às publicações locais;
3. Às comissões avaliadoras para a elaboração de perguntas no idioma local adaptadas aos objetivos de aprendizagem de cada *syllabus*;
4. Aos provedores de treinamento para produzirem os materiais dos cursos e definirem os métodos adequados de ensino;
5. Aos candidatos à certificação para prepará-los para a avaliação (em função de um curso de treinamento ou independentemente disso);
6. À comunidade internacional de engenharia de *software* e sistemas para fomentar o desenvolvimento da profissão de teste de *software* e sistemas e servir de base para livros e artigos.

O ISTQB® pode permitir que outras entidades utilizem este *syllabus* para outras finalidades, contanto que procurem e obtenham uma autorização prévia por escrito.

0.2 Panorama

O nível avançado é composto por três *syllabus* diferentes: *Test Manager (TM), Test Analyst (TA)* e *Technical Test Analyst (TTA)*.

O documento que faz um panorama do nível avançado [BSTQB_AL_OVIEW] contém as seguintes informações:

- Os resultados comerciais de cada *syllabus*;
- Um resumo de cada *syllabus*;
- Os relacionamentos entre os *syllabus*;
- Uma descrição dos níveis cognitivos (níveis K);
- Os anexos.

0.3 Objetivos de aprendizagem sujeitos à avaliação

Os objetivos de aprendizagem fundamentam os resultados comerciais e são utilizados na criação de avaliações para a obtenção da certificação *Advanced Test Analyst Certification*. Em geral, todas as partes deste *syllabus* estão sujeitas a uma avaliação no nível K1. Isto é, o candidato reconhecerá e se lembrará de um termo ou de um conceito. Os objetivos de aprendizagem nos níveis K2, K3 e K4 aparecem no começo do capítulo pertinente.

1. Processo de teste (300 min)

Palavras-chave

Caso de teste concreto, critério de saída, caso de teste de alto nível, caso de teste lógico, caso de teste de baixo nível, controle de testes, modelagem de testes, execução de testes, implementação de testes, planejamento de testes

Objetivos de aprendizagem do processo de teste

1.2 Teste no ciclo de vida de desenvolvimento de software

TA-1.2.1 (K2): Explica-se como e por quê a oportunidade e o nível do envolvimento do *Test Analyst (TA)* variam quando ele trabalha com diferentes modelos de ciclo de vida.

1.3 Planejamento, monitoramento e controle de testes

TA-1.3.1 (K2): Resumem-se as atividades realizadas pelo *Test Analyst (TA)* para fundamentar o planejamento e o controle de testes.

1.4 Análise de testes

TA-1.4.1 (K4): Analisa-se determinada situação, inclusive uma descrição de projeto e um modelo de ciclo de vida, para definir as tarefas adequadas para o *Test Analyst (TA)* durante as fases de análise e modelagem.

1.5 Modelagem de testes

TA-1.5.1 (K2): Explica-se por quê as condições de teste devem ser compreendidas pelos *stakeholders*.

TA-1.5.2 (K4): Analisa-se a situação de um projeto para determinar o uso mais adequado de casos de teste de baixo nível (concretos) e de alto nível (lógicos).

1.6 Implementação de testes

TA-1.6.1 (K2): Descrevem-se os critérios de saída normais para a análise e a modelagem de testes e explica-se como o atendimento de tais critérios afeta a implementação de testes.

1.7 Execução de testes

TA-1.7.1 (K3): Em determinada situação, definem-se as etapas e as considerações que devem ser levadas em conta na execução de testes.

1.8 Avaliação de critérios de saída e divulgação

TA-1.8.1 (K2): Explica-se por quê é importante contar com informações exatas sobre a situação da execução de casos de teste.

1.9 Atividades de fechamento de teste

TA-1.9.1 (K2): Dão-se exemplos de produtos de trabalho que devem ser entregues pelo *Test Analyst (TA)* durante as atividades de fechamento de teste.

1.1 Introdução

No *syllabus* do nível fundamental do ISTQB®, o processo de teste fundamental contém as seguintes atividades:

- Planejamento, monitoramento e controle;
- Análise e modelagem;
- Implementação e execução;
- Avaliação de critérios de saída e divulgação;
- Atividades de fechamento de teste.

No nível avançado, algumas destas atividades são consideradas à parte para refinar e otimizar mais os processos, se adequar melhor ao ciclo de vida de desenvolvimento de *software* e facilitar o monitoramento e o controle eficazes de testes. Neste nível, as atividades são as seguintes:

- Planejamento, monitoramento e controle;
- Análise;
- Modelagem;
- Implementação;
- Execução;
- Avaliação de critérios de saída e divulgação;
- Atividades de fechamento de teste.

As atividades podem ser implementadas sequencialmente ou algumas podem ser implementadas paralelamente, por exemplo, a modelagem pode ser realizada junto com a implementação (por exemplo, teste exploratório). A definição, a modelagem e a execução dos testes e dos casos de teste corretos são as principais áreas de concentração do *Test Analyst (TA)*. Embora seja importante compreender as demais etapas do processo de teste, boa parte do trabalho do *Test Analyst (TA)* costuma ser feita durante as atividades de análise, modelagem, implementação e execução do projeto de teste.

Os testadores de nível avançado enfrentam uma série de desafios ao introduzir os diferentes aspectos de teste descritos neste *syllabus* no contexto de suas áreas, suas equipes e suas atribuições. É importante levar em consideração tanto os diferentes ciclos de vida de desenvolvimento de *software* quanto o tipo de sistema testado, já que tais fatores podem influenciar a abordagem ao teste.

1.2 Teste no ciclo de vida de desenvolvimento de software

A abordagem aos testes segundo o ciclo de vida a longo prazo deve ser considerada e definida nos termos da estratégia de teste. O momento do envolvimento do *Test Analyst (TA)* muda de acordo com os vários ciclos de vida e o envolvimento, o tempo necessário, as informações disponíveis e as expectativas podem variar bastante também. Como os processos de teste não são realizados isoladamente, o *Test Analyst (TA)* precisa saber quando as informações podem ser fornecidas a outras áreas organizacionais relacionadas como:

- Engenharia e gestão de requisitos: avaliação de requisitos;
- Gestão de projetos: contribuição para o cronograma;

- Gestão de configurações e alterações: elaboração de testes de verificação e controle de versões;
- Desenvolvimento de *software*: previsões e expectativas;
- Manutenção de *software*: gestão de defeitos e tempo de resposta (por exemplo, o tempo que vai da detecção à solução de um defeito);
- Suporte técnico: documentação exata das soluções de contorno;
- Elaboração de documentos técnicos (por exemplo, especificações de modelagem de bancos de dados): contribuições para os documentos e avaliação técnica dos documentos.

As atividades de teste precisam combinar com o modelo escolhido de ciclo de vida de desenvolvimento de *software*, cuja natureza pode ser sequencial, iterativa ou incremental. Por exemplo, no modelo V sequencial, o processo de teste fundamental do ISTQB® aplicado ao nível de teste de sistema pode ser adequado da seguinte maneira:

- O planejamento de testes de sistema acontece junto com o planejamento de projetos e o controle de testes continua até a conclusão da execução e do fechamento de testes de sistema;
- A análise e a modelagem de testes de sistema acontecem junto com as especificações de requisitos, de modelagem de sistemas e arquiteturas (de alto nível) e de modelagem de componentes (de baixo nível);
- A implementação de ambientes de teste de sistema (por exemplo, bancos e equipamentos de teste) pode ter início durante a modelagem de sistemas, embora boa parte dela ocorra junto com o teste de códigos e componentes, sendo que, frequentemente, os trabalhos de implementação de testes de sistema duram até poucos dias antes do começo da execução de testes de sistema;
- A execução de testes de sistema começa quando os critérios de entrada dos testes de sistema são atendidos (ou desconsiderados), o que, normalmente, significa que pelo menos o teste de componente e, frequentemente, o teste de integração de componentes foram concluídos; A execução de testes de sistema continua até o cumprimento dos critérios de saída dos testes de sistema;
- A avaliação dos critérios de saída dos testes de sistema e a divulgação dos resultados dos testes de sistema ocorrem durante a execução de testes de sistema, geralmente com uma frequência e uma urgência maior quando os prazos dos projetos se aproximam;
- As atividades de fechamento de testes de sistema ocorrem depois que os critérios de saída dos testes de sistema foram atendidos e a execução de testes de sistema foi concluída, embora, às vezes, possam ser atrasadas até a conclusão dos testes de aceitação e a finalização de todas as atividades do projeto.

Pode ser que os modelos iterativos e incrementais não sigam a mesma ordem de tarefas e talvez excluam algumas tarefas. Por exemplo, um modelo iterativo pode utilizar um número menor de processos de teste padrão para cada iteração. A análise, a modelagem, implementação, a execução e a divulgação podem ser realizadas para cada iteração, enquanto o planejamento é feito no início do projeto e a divulgação do fechamento é realizada no final.

Em um projeto ágil, são comuns a utilização de um processo menos formalizado e um relacionamento de trabalho muito mais entrosado para facilitar as mudanças no projeto. Como o processo ágil é “leve”, há menos documentos de teste abrangentes a favor de um método mais rápido de comunicação, como reuniões diárias “em pé” (porque, como são muito rápidas, normalmente de 10 a 15 minutos, ninguém precisa se sentar e todos continuam envolvidos).

De todos os modelos de ciclo de vida, os projetos ágeis são os que mais exigem o envolvimento precoce do *Test Analyst (TA)*. O *Test Analyst (TA)* deve esperar o envolvimento desde o começo do projeto e a colaboração com os desenvolvedores à medida que constroem a arquitetura inicial e fazem a modelagem. Pode ser que as avaliações não sejam formalizadas, mas são contínuas, já que o *software* evolui. Espera-se o envolvimento ao longo do projeto e o *Test Analyst (TA)* deve ficar à disposição da equipe. Em função de tal imersão, os integrantes das equipes ágeis normalmente se dedicam a um só projeto e participam totalmente de todos os aspectos do projeto.

Os modelos iterativos / incrementais vão da abordagem ágil, onde alterações são esperadas à medida que o *software* evolui, a modelos iterativos / incrementais de desenvolvimento que existem no Modelo V (às vezes chamado de iterativo incorporado). Em caso de um modelo iterativo incorporado, espera-se que o *Test Analyst (TA)* participe dos aspectos normais de planejamento e modelagem, mas depois assuma uma função mais interativa à medida que o *software* é desenvolvido, testado, alterado e implantado.

Independentemente do ciclo de vida de desenvolvimento de *software* utilizado, é importante que o *Test Analyst (TA)* compreenda as expectativas de envolvimento e a oportunidade de tal participação. Há muitos modelos híbridos em uso, como o iterativo no modelo V supracitado. Frequentemente, o *Test Analyst (TA)* precisa determinar a função e o trabalho mais eficazes para tal em vez de depender da definição de determinado modelo para indicar o momento adequado do envolvimento.

1.3 Planejamento, monitoramento e controle de testes

Esta parte trata dos processos de planejamento, monitoramento e controle de testes.

1.3.1 Planejamento de testes

Em sua maioria, o planejamento de testes acontece no início dos testes e envolve a identificação e o planejamento de todas as atividades e todos os recursos necessários para cumprir a missão e realizar os objetivos identificados nesta estratégia de teste. Durante o planejamento de testes, é importante que o *Test Analyst (TA)*, em colaboração com o *Test Manager (TM)*, leve em consideração e faça planos para o seguinte:

- Certifique-se de que os planos do teste não se limitam ao teste funcional. Todos os tipos de teste devem ser levados em consideração no plano de testes e assim agendados. Por exemplo, além do teste funcional, o *Test Analyst (TA)* pode ser responsável pelo teste de usabilidade. O documento do plano do teste também deve abranger tal tipo de teste;
- Revise as estimativas de teste junto ao *Test Manager (TM)* e certifique-se de que há tempo suficiente para a obtenção e a validação do ambiente de teste;
- Faça plano para o teste de configurações. Se vários tipos de processadores, sistemas operacionais, máquinas virtuais, navegadores e diversos periféricos puderem ser combinados em muitas possíveis configurações, faça planos para aplicar as técnicas de teste que proporcionarão uma cobertura adequada a tais combinações;
- Faça planos para testar a documentação. Os usuários recebem o *software* com a documentação.
- A documentação precisa ser exata para ser eficaz. O *Test Analyst (TA)* deve reservar tempo para verificar a documentação e pode ter que trabalhar com a equipe de redação técnica a fim de preparar os dados que serão utilizados em capturas de tela e videoclipes;

- Faça planos para testar os procedimentos de instalação. Os procedimentos de instalação, de *back-up* e de restauração devem ser testados o bastante. Tais procedimentos podem ser mais importantes do que o próprio *software*. Se o *software* não puder ser instalado, não será nem sequer utilizado. Pode ser difícil planejar isto, já que, frequentemente, o *Test Analyst (TA)* faz os testes iniciais em um sistema que foi pré-configurado sem a implementação dos processos finais de instalação;
- Faça planos para que o teste se adapte ao ciclo de vida do *software*. A execução sequencial de tarefas não combina com a maioria dos cronogramas. Frequentemente, muitas tarefas precisam ser realizadas ao mesmo tempo (pelo menos parte delas);
- O *Test Analyst (TA)* precisa estar ciente do ciclo de vida escolhido e das expectativas de envolvimento durante a concepção, o desenvolvimento e a implementação do *software*. Isto também inclui reservar tempo para testes de confirmação e regressão;
- Reserve tempo suficiente para a identificação e a análise de riscos com a equipe interfuncional;
- Embora normalmente não seja responsável pela organização das sessões de gestão de risco, espera-se que o *Test Analyst (TA)* participe ativamente de tais atividades.

Podem existir relacionamentos complexos entre a base de teste, as condições de teste e os casos de teste, de modo que muitos relacionamentos podem existir entre tais produtos de trabalho. Eles precisam ser assimilados para possibilitar a implementação eficaz do planejamento e do controle de testes. Normalmente, o *Test Analyst (TA)* é a pessoa mais indicada para determinar tais relacionamentos e separar as dependências o máximo possível.

1.3.2 Monitoramento e controle de testes

Embora normalmente o *Test Manager (TM)* seja responsável pelo monitoramento e pelo controle de testes, o *Test Analyst (TA)* contribui com as medições que possibilitam o controle.

Vários dados quantitativos devem ser coletados ao longo do ciclo de vida de desenvolvimento de *software* (por exemplo, a porcentagem de atividades de planejamento concluídas, a porcentagem de cobertura atingida, o número de casos de teste que foram aprovados e reprovados). Em cada caso, uma *baseline* (isto é, uma referência) precisa ser definida. Então, o progresso deve ser rastreado em relação à *baseline*. Enquanto o *Test Manager (TM)* se responsabiliza pela compilação e pela divulgação do resumo de métricas, o *Test Analyst (TA)* coleta as informações referentes a cada métrica.

Cada caso de teste concluído, cada relatório de defeito escrito, cada marca atingida serão incorporados às métricas gerais do projeto. É importante que as informações introduzidas nas diversas ferramentas de rastreamento sejam as mais exatas possíveis para que as métricas reflitam a realidade.

Métricas exatas permitem que os gestores coordenem projetos (monitoramento) e façam as alterações necessárias (controle). Por exemplo, um número alto de defeitos em uma área do *software* pode indicar que são necessários mais testes naquela área. Os requisitos e as informações da cobertura do risco (rastreabilidade) podem ser utilizados para a priorização do trabalho restante e a alocação de recursos.

As causas-raiz são utilizadas para determinar as áreas para o aprimoramento de processos. Se os dados registrados forem exatos, o projeto pode ser controlado e informações de estado precisas podem ser repassadas aos *stakeholders*. Futuros projetos podem ser planejados com maior eficácia quando o planejamento levar em consideração os dados coletados de projetos passados. Os dados exatos podem ser

utilizados de muitas maneiras. Uma das atribuições do *Test Analyst (TA)* é a de se certificar de que os dados sejam exatos, oportunos e objetivos.

1.4 Análise de testes

Durante o planejamento de testes, a abrangência do projeto de teste é definida. O *Test Analyst (TA)* utiliza a definição da abrangência para:

- Analisar a base de teste;
- Identificar as condições de teste.

Para que o *Test Analyst (TA)* proceda à análise do teste com eficácia, os seguintes critérios de entrada precisam ser atendidos:

- Há um documento que descreve o objeto de teste que pode servir de base de teste;
- Tal documento foi revisado e obteve resultados razoáveis. Após a revisão, recebeu as atualizações necessárias;
- Há um orçamento e um cronograma razoável para a realização do trabalho restante de teste referente a este objeto de teste.

Normalmente, as condições de teste são identificadas pela análise da base de teste e dos objetivos de teste. Dependendo da situação, por exemplo, quando os documentos estão velhos ou não existem, as condições de teste podem ser identificadas através de uma conversa com os *stakeholders* pertinentes (por exemplo, em *workshops* ou durante uma sessão de planejamento-relâmpago). Então, tais condições são usadas para determinar o que testar com técnicas de modelagem de testes identificadas na estratégia e /ou no plano de teste.

Embora as condições de teste costumem ser próprias do item testado, normalmente o *Test Analyst (TA)* precisa levar o seguinte em consideração:

- Normalmente, é aconselhável definir as condições de teste em níveis diferentes de detalhe. A princípio, as condições de alto nível são identificadas para a definição dos objetivos gerais do teste, como a “funcionalidade da tela x”. Em seguida, condições mais detalhadas são identificadas e servem de base para casos de teste específicos, como “a tela x recusa números de conta que ficam um dígito abaixo do comprimento correto”. Com este tipo de abordagem hierárquica para a definição das condições de teste, é possível garantir que haja cobertura suficiente para os itens de alto nível;
- Se os riscos de produto foram definidos, as condições de teste, que existem necessariamente para lidar com cada risco de produto, precisam ser identificadas e devolvidas a tal item de risco.

Quando as atividades de análise de testes forem concluídas, o *Test Analyst (TA)* deve saber quais testes específicos precisam ser modelados para atender às necessidades das áreas atribuídas do projeto de teste.

1.5 Modelagem de testes

Ainda de acordo com a abrangência definida durante o planejamento de testes, o processo de teste continua com o *Test Analyst (TA)*, que modela os testes que serão implementados e executados. O processo de modelagem de testes inclui as seguintes atividades:

- Determinar em quais áreas de teste os casos de teste de baixo nível (concretos) ou de alto nível (lógicos) são mais adequados;
- Determinar a/s técnica/s de modelagem de casos de teste que propiciam a cobertura de teste necessária;
- Criar os casos de teste que exercem as condições de teste identificadas.

Os critérios de priorização identificados durante a análise de riscos e o planejamento de testes devem ser aplicados no processo inteiro, da análise à implementação, passando pela modelagem e pela execução.

Dependendo dos tipos de testes modelados, um dos critérios de entrada para a modelagem de testes pode ser a disponibilidade das ferramentas que serão utilizadas durante os trabalhos de modelagem.

Ao modelar testes, é importante lembrar do seguinte:

- É possível tratar de alguns itens de teste com maior facilidade mediante a definição somente das condições de teste em vez da definição de testes com *script*. Neste caso, as condições de teste devem ser definidos para a utilização como guia para os testes com *script*;
- Os critérios de aprovação / reprovação devem ser claramente identificados;
- Os testes devem ser modelados de modo que possam ser compreendidos por outros testadores e não só o autor. Se o autor não for a pessoa que executa o teste, outros testadores precisarão ler e assimilar os testes especificados anteriormente para compreenderem os objetivos do teste e a importância relativa dele;
- Além disso, outros *stakeholders*, como os desenvolvedores, os quais avaliarão os testes, e os auditores, que talvez tenham que aprová-los, precisam poder compreender os testes;
- Os testes devem ser modelados para que cubram todas as interações do *software* com os atores (por exemplo, usuários finais, outros sistemas) e não só as interações que ocorrem através da interface visível para o usuário. A comunicação entre processos, a execução por lotes e outras interrupções também interagem com o *software* e podem conter defeitos. Assim, o *Test Analyst (TA)* precisa modelar testes para mitigar tais riscos;
- Os testes devem ser modelados para testar as interfaces entre os diversos objetos de teste.

1.5.1 Casos de teste concretos e lógicos

Uma das atribuições do *Test Analyst (TA)* consiste em determinar os melhores tipos de casos de teste em determinada situação. Os casos de teste concretos fornecem todas as informações específicas e os procedimentos necessários para que o testador execute o caso de teste (inclusive quaisquer exigências de dados) e verifique os resultados.

Os casos de teste concretos são úteis quando os requisitos são bem definidos, quando o pessoal da área de teste é menos experiente e quando a verificação externa dos testes, como auditorias, é exigida. Os casos de teste concretos proporcionam uma reprodutibilidade excelente (isto é, outro testador obterá os mesmos resultados), mas também exige muitos trabalhos de manutenção e tende a tolher a engenhosidade do testador durante a execução.

Os casos de teste lógicos fornecem orientações referentes ao que deve ser testado, mas permitem que o *Test Analyst (TA)* varie os dados reais ou até mesmo o procedimento seguido durante a execução do teste. Os casos de teste lógicos podem propiciar uma cobertura melhor do que os casos de teste concretos porque

variam um pouco sempre que são executados. Isto também leva à perda de reprodutibilidade. Os casos de teste lógicos são úteis quando os requisitos não são bem definidos, quando o *Test Analyst (TA)* que executará o teste é experiente tanto com o teste quanto com o produto e quando a documentação formal não é exigida (por exemplo, nenhuma auditoria será realizada).

Os casos de teste lógicos podem ser definidos nos momentos iniciais do processo de requisitos quando eles ainda não estão bem definidos. Tais casos de teste podem ser utilizados para o desenvolvimento de casos de teste concretos quando os requisitos ficarem mais definidos e estáveis. Neste caso, a criação de casos de teste é realizada sequencialmente, fluindo do lógico para o concreto apenas com os casos de teste concretos utilizados na execução.

1.5.2 Criação de casos de teste

Os casos de teste são modelados pela elaboração e pelo refinamento por etapas das condições de teste identificadas através da utilização das técnicas de modelagem de testes (*vide* o capítulo 3) identificadas na estratégia e / ou no plano de teste.

Os casos de teste devem poder ser repetidos e verificados e sua origem deve poder ser determinada de acordo com a base de teste (por exemplo, requisitos), conforme definido pela estratégia de teste utilizada.

A modelagem de casos de teste inclui a identificação do seguinte:

- Objetivo;
- Pré-condições, como requisitos de projeto ou de ambiente de teste localizado e os planos de entrega, estado do sistema *etc.*;
- Requisitos de dados de teste (tanto os dados de entrada do caso de teste quanto os dados que devem existir no sistema para a execução do caso de teste);
- Resultados esperados;
- Pós-condições, como dados afetados, estado do sistema, causas de processamento subsequente *etc.*

O nível do detalhe nos casos de teste, que afeta tanto o custo do desenvolvimento quanto o nível de repetição durante a execução, deve ser definido antes da criação dos casos de teste. Menos detalhes no caso de teste proporcionam ao *Test Analyst (TA)* mais flexibilidade na execução de casos de teste e lhe dá a oportunidade de investigar áreas de possível interesse. Contudo, menos detalhes também tendem a diminuir a reprodutibilidade.

Frequentemente, um desafio em particular é a definição do resultado esperado de um teste. Calcular isto manualmente é muitas vezes tedioso e tende ao erro. Se possível, é preferível encontrar ou criar um oráculo de testes automatizados. Ao identificar os resultados esperados, os testadores se preocupam não só as saídas na tela, mas também com os dados e as pós-condições ambientais. Se a base de teste for claramente definida, a identificação do resultado correto, na teoria, deve ser simples.

No entanto, frequentemente as bases de teste são vagas, contraditórias, carentes de cobertura para áreas importantes ou totalmente inexistentes. Em tais casos, o *Test Analyst (TA)* deve ter *expertise* no assunto ou contar com acesso a ela. Além disso, mesmo quando a base de teste é bem especificada, as interações complexas de estímulos e respostas complexos podem dificultar a definição dos resultados esperados. Portanto, é essencial dispor de um oráculo de teste. A execução do caso de teste, sem nenhuma maneira de

determinar se os resultados estão corretos, agrega muito pouco valor ou benefício, gerando, frequentemente, relatórios falsos de falhas ou desconfiança no sistema.

As atividades supracitadas podem ser aplicadas em todos níveis de teste, embora a base de teste varie. Por exemplo, os testes de aceitação do usuário podem se fundamentar principalmente nas especificações de requisitos, nos casos de uso e nos processos de negócios definidos, enquanto os testes de componente podem se basear principalmente nas especificações de modelagem de baixo nível, nas estórias de usuários e no próprio código.

É importante se lembrar de que tais atividades acontecem em todos os níveis de teste, embora o alvo dos testes possa variar. Por exemplo, o teste funcional no nível da unidade foi modelado para garantir que determinado componente lhe proporcione a funcionalidade especificada na modelagem detalhada do componente. O teste funcional no nível de integração serve para verificar se os componentes interagem em conjunto são funcionais ao longo da interação.

No nível do sistema, a funcionalidade integrada deve ser o alvo do teste. Ao analisar e modelar os testes, é importante se lembrar do nível do alvo do teste e o objetivo do teste. Isto ajuda a determinar o nível dos detalhes necessários e as ferramentas necessárias (por exemplo, controladores e simuladores no nível do teste de componente).

Durante o desenvolvimento das condições de teste e dos casos de teste, normalmente alguns documentos são criados, o que resulta em produtos de trabalho de teste. Na prática, a documentação dos produtos de trabalho de teste varia bastante. Ela pode ser afetada por qualquer uma das seguintes alternativas:

- Riscos do projeto (o que deve / o que não deve ser documentado);
- O “valor agregado” que a documentação proporciona ao projeto;
- As normas que devem ser seguidas e / ou os regulamentos que precisam ser cumpridos;
- O modelo de ciclo de vida utilizado (por exemplo, uma abordagem ágil procura a documentação “suficiente”);
- O requisito de rastreabilidade da base de teste através da análise e da modelagem de testes.

Dependendo da abrangência do teste, a análise e a modelagem de testes lidam com as características de qualidade do/s objeto/s de teste. A norma ISO 25000 [ISO25000] (que suplanta a ISO 9126) contém referências úteis. Ao testar sistemas de *hardware* / *software*, outras características podem ser aplicadas.

Os processos de análise e modelagem de testes podem ser aprimorados ao entrelaçá-los com as avaliações e as análises estáticas. Na verdade, a realização da análise e da modelagem de testes é frequentemente uma forma de teste estático porque é possível encontrar problemas nos documentos da base durante este processo. A análise e a modelagem de testes com base nas especificações de requisitos são maneiras excelentes de fazer preparativos para a reunião de avaliação de requisitos.

A leitura dos requisitos para que sejam utilizados na criação de testes exige a compreensão dos requisitos e a capacidade de determinar uma maneira para a avaliar o cumprimento deles. Frequentemente, esta atividade descobre requisitos que não são claros, não são passíveis de teste ou não contam com critérios de aceitação definidos. Igualmente, os produtos de trabalho de teste, como os casos de teste, as análises de risco e os planos de teste, ficam sujeitos a avaliações.

Alguns projetos, como os que seguem um ciclo de vida ágil, podem ter apenas requisitos minimamente documentados. Às vezes, aparecem na forma de *estórias de usuário*, que descrevem partes pequenas, mas

demonstráveis, da funcionalidade. As estórias de usuário devem incluir a definição dos critérios de aceitação. Se o *software* conseguir demonstrar que os critérios de aceitação foram atendidos, normalmente considera-se que está pronto para a integração com as demais funcionalidades concluídas ou já pode ter sido integrado para demonstrar sua funcionalidade.

Durante a modelagem de testes, é possível definir os requisitos necessários de detalhamento da infraestrutura de teste, embora, na prática, só sejam finalizados na implementação do teste. Vale lembrar que a infraestrutura de teste vai além dos objetos de teste e do *testware*. Por exemplo, os requisitos da infraestrutura podem incluir salas, equipamentos, pessoal, *software*, ferramentas, periféricos, equipamentos de comunicação, autorizações de usuário e todos os demais itens necessários para a execução dos testes.

Os critérios de saída da análise e modelagem de testes variarão dependendo dos parâmetros do projeto, mas todos os itens discutidos nestas duas seções devem ser levados em consideração para que sejam incluídos nos critérios de saída definidos. É importante que os critérios sejam mensuráveis e é preciso garantir que todas as informações e os preparativos necessários para as etapas seguintes tenham sido providenciados.

1.6 Implementação de testes

A implementação do teste é a realização da modelagem do teste. Ela inclui a criação de testes automatizados, a organização de testes (manuais e automatizados) por ordem de execução, a finalização dos dados do teste e dos ambientes de teste e a elaboração de um cronograma de execução de testes, inclusive a alocação de recursos, para possibilitar o início da execução dos casos de teste. Isto também inclui a verificação de critérios de entrada explícitos e implícitos para o nível de teste em questão e garante que os critérios de saída das etapas anteriores no processo foram atendidos.

Se os critérios de saída foram pulados, ou por causa do nível do teste ou por conta de uma etapa no processo de teste, os trabalhos de implementação provavelmente serão afetados com atrasos no cronograma, qualidade insuficiente e esforços extra inesperados. É importante garantir que todos os critérios de saída tenham sido atendidos antes de dar início aos trabalhos de implementação de testes.

Ao determinar a ordem de execução, muitas questões podem vir à tona. Em alguns casos, talvez faça sentido organizar os casos de teste em suítes de teste (isto é, grupos de casos de teste). Isto pode contribuir para a organização do teste. Assim, casos de teste relacionados são executados juntos. Se uma estratégia de teste baseada em risco for utilizada, a ordem de prioridade dos riscos pode determinar a ordem de execução para os casos de teste.

Outros fatores podem determinar a ordem, como a disponibilidade das pessoas certas, os equipamentos, os dados e as funcionalidades que devem ser testadas. Não é incomum que o código seja lançado em seções e os trabalhos de teste precisam ser coordenados a ordem segundo a qual o *software* fica disponível para o teste. Particularmente em modelos de ciclo de vida incrementais, é importante que o *Test Analyst (TA)* se coordene com a equipe de desenvolvimento para garantir o lançamento do *software* para teste por ordem testável.

Durante a implementação do teste, o *Test Analyst (TA)* deve finalizar e confirmar a ordem segundo a qual os testes manuais e automatizados serão executados, verificando cuidadosamente as restrições que possam

exigir a execução dos testes de acordo com uma ordem específica. As dependências precisam ser documentadas e verificadas.

O nível de detalhe e complexidade relacionada referente ao trabalho realizado durante a implementação de testes pode ser influenciado pelo detalhamento dos casos e das condições de teste. Em alguns casos, é preciso cumprir certos regulamentos e os testes devem fornecer provas de conformidade com as normas pertinentes, como a DO-178B/ED 12B [RTCA DO-178B/ED-12B] da Agência Nacional de Aviação Civil dos Estados Unidos.

Conforme especificado acima, os dados do teste precisam ser testados e, em alguns casos, tais conjuntos de dados podem ser bastante grandes. Durante a implementação, o *Test Analyst (TA)* cria dados de entrada e de ambiente para que sejam carregados nos bancos de dados e em outros repositórios. O *Test Analyst (TA)* também cria dados que serão utilizados com testes automatizados e testes manuais orientados para dados.

A implementação de testes também se responsabiliza pelo/s ambiente/s de teste. Durante esta etapa, o/s ambiente/s devem ser totalmente preparados e verificados antes da execução de testes. O ambiente de teste de “adequação” é essencial. Em outras palavras, o ambiente de teste deve possibilitar a descoberta de defeitos presentes durante o teste controlado, funcionar normalmente quando não houver nenhuma falha e replicar adequadamente, se necessário, o ambiente de produção ou do usuário final para níveis superiores de teste.

Talvez sejam necessárias mudanças no ambiente de teste durante a execução dos testes, dependendo de imprevistos nas alterações, nos resultados dos testes ou outras considerações. Se houver alguma mudança no ambiente durante a execução, é importante avaliar o impacto das mudanças nos testes que já foram executados.

Durante a implementação de testes, os testadores precisam garantir que os responsáveis pela criação e pela manutenção do ambiente de testes sejam conhecidos e estejam à disposição e que todo o *testware* e todas as ferramentas de suporte ao teste e processos relacionados estejam prontos para o uso. Isto inclui a gestão de configurações, a gestão de defeitos e o registro e a gestão de testes. Além disso, o *Test Analyst (TA)* deve verificar os procedimentos que coletam dados para a avaliação de critérios de saída e a divulgação dos resultados dos testes.

A realização de uma abordagem equilibrada à implementação dos testes, conforme determinado durante o planejamento de testes, é uma sábia decisão. Por exemplo, as estratégias de testes analíticos baseados em risco frequentemente se misturam com as estratégias de testes dinâmicos. Neste caso, uma porcentagem dos trabalhos de implementação de testes é alocada ao teste, que não segue *scripts* predeterminados (sem *script*).

Os testes sem *script* não devem ser nem *ad hoc* nem despropositados, já que a duração e a cobertura deles podem ser imprevisíveis, a menos que um cronograma seja definido e os testes sejam credenciados. Ao longo dos anos, os testadores desenvolveram uma variedade de técnicas baseadas na experiência, como ataques, suposição de erros [Myers79] e testes exploratórios. A análise, a modelagem e a implementação de testes ainda ocorrem, mas acontecem principalmente durante a execução de testes.

Ao seguir tais estratégias de testes dinâmicos, os resultados de cada teste influenciam a análise, a modelagem e a implementação dos testes seguintes. Embora tais estratégias sejam leves e, frequentemente, eficazes na descoberta de defeitos, alguns contratempos existem. As técnicas exigem *expertise* do *Teste*

Analyst. A duração pode ser difícil de prever, a cobertura pode ser difícil de rastrear e a repetição pode se perder sem uma boa documentação ou um bom suporte das ferramentas.

1.7 Execução de testes

A execução do teste começa assim que o objeto de teste é entregue e os critérios de entrada para a execução do teste são atendidos (ou desconsiderados). Os testes devem ser executados de acordo com o plano determinado durante a implementação de testes, mas o *Test Analyst (TA)* deve ter tempo o suficiente para garantir a cobertura de outros cenários e comportamentos interessantes de teste que são observados durante o teste (qualquer falha detectada durante tais desvios deve ser descrita, inclusive as variações do caso de teste com *script*, as quais são necessárias para reproduzir a falha). Tal integração de técnicas de teste com e sem *script* (por exemplo, testes exploratórios) ajudam a criar uma proteção contra as fugas de teste devido a lacunas na cobertura com *script* e a evitar o paradoxo do pesticida.

A comparação dos resultados reais com os resultados esperados é a alma da atividade de execução de testes. O *Test Analyst (TA)* deve prestar atenção a estas tarefas e se concentrar nelas, senão todos os trabalhos de modelagem e implementação de testes podem ser em vão quando as falhas não são detectadas (resultado falso negativo) ou um comportamento correto é classificado como incorreto (resultado falso positivo). Se os resultados esperados e os reais não baterem, houve um incidente.

Os incidentes devem ser examinados com cuidado para determinar a causa (que talvez seja um defeito no objeto de teste) e coletar dados para facilitar a resolução do incidente (*vide* o capítulo 6 para obter mais detalhes sobre a gestão de defeitos).

Quando uma falha é identificada, os documentos de teste (as especificações de teste, os casos de teste *etc.*) devem ser cuidadosamente avaliados para ter a certeza de que estão corretos. Um documento de teste pode estar incorreto por uma série de motivos. Se estiver incorreto, deve ser corrigido e o teste deve ser executado novamente.

Como as mudanças na base de teste e no objeto de teste podem deixar um caso de teste incorreto esmo depois de o teste ser executado com sucesso várias vezes, os testadores devem estar cientes da possibilidade de que os resultados observados se devam a um teste incorreto.

Durante a execução do teste, os resultados do teste devem ser devidamente registrados. Os testes executados cujos resultados não foram registram talvez tenham que ser repetidos para identificar o resultado correto, o que leva a ineficiência e atrasos. (Repare que o registro adequado pode lidar com as questões de cobertura e repetibilidade relacionadas às técnicas de teste, como os testes exploratórios.) Como o objeto de teste, o *testware* e o ambiente de teste podem estar evoluindo, o registro deve identificar as versões específicas que foram testadas e as configurações específicas do ambiente. O registro de testes faz um cronograma dos detalhes relevantes sobre a execução dos testes.

O registro de resultados vale tanto para testes individuais quanto para atividades e eventos. Cada teste deve ter uma identificação única e o *status* deve ser cadastrado à medida que o teste é executado. Todo e qualquer evento que afetar a execução de testes deve ser registrado.

Um número suficiente de informações deve ser registrado para medir a cobertura do teste e documentar os motivos de atrasos e interrupções no teste. Além disso, as informações devem ser registradas para apoiar o

controle de testes, a divulgação do andamento dos testes, a medição dos critérios de saída e o aprimoramento do processo de teste.

O cadastramento varia dependendo do nível de teste e da estratégia. Por exemplo, se o teste automatizado de componente estiver sendo realizando, os testes automatizados devem produzir boa parte das informações de registro. Se o teste manual estiver sendo realizado, o *Test Analyst (TA)* registrará as informações referentes à execução do teste, frequentemente em uma ferramenta de gestão de testes que rastreará as informações de execução dos testes. Em alguns casos, como na implementação de testes, o número registrado de informações sobre a execução de testes é influenciado por exigências dos regulamentos ou das auditorias.

Em alguns casos, os usuários ou os clientes podem participar da execução de testes. Isto pode servir para criar confiança no sistema, embora isto suponha que os testes detectaram poucos defeitos. Tal suposição é frequentemente inválida nos níveis iniciais de teste, mas pode ser válida durante o teste de aceitação.

Seguem algumas áreas específicas que devem ser levadas em consideração durante a execução de testes:

- Detectar e explorar singularidades “irrelevantes”. As observações ou os resultados que podem parecer irrelevantes costumam indicar defeitos que (como os *icebergs*) ficam abaixo da superfície;
- Verifique se o produto não está fazendo o que não deveria estar fazendo. Verificar se o produto está fazendo o que deve estar fazendo é o foco normal do teste, mas o *Test Analyst (TA)* também deve garantir que o produto não tenha um comportamento inadequado por fazer algo que não deveria estar fazendo (por exemplo, outras funções indesejadas);
- Crie a suíte de testes, que crescerá e mudará com o tempo. O código evoluirá e será necessário implementar outros testes para cobrir estas novas funcionalidades e verificar regressões em outras áreas do *software*. As lacunas nos testes são frequentemente descobertas durante a execução. A elaboração de uma suíte de testes é um processo contínuo;
- Tome notas para o próximo teste. As tarefas do teste não acabam quando o *software* é fornecido ao usuário ou distribuído no mercado. Uma nova versão ou *release* do *software* será provavelmente produzido, então, o conhecimento deve ser armazenado e transferido aos testadores responsáveis pelo próximo teste;
- Não espere uma nova execução de todos os testes manuais. Não é realista esperar que todos os testes manuais sejam executados novamente. Se houver suspeita de algum problema, o *Test Analyst (TA)* deve investigá-lo e observá-lo e não supor que será detectado em uma execução futura dos casos de teste;
- Minere dados na ferramenta de rastreamento de defeitos para obter outros casos de teste. Crie casos de teste para defeitos descobertos durante testes sem *script* ou exploratórios e acrescente-os à suíte de testes de regressão;
- Encontre os defeitos antes do teste de regressão. Frequentemente, o tempo do teste de regressão é limitado e encontrar falhas durante o teste de regressão pode levar a atrasos no cronograma. Geralmente, os testes de regressão não detectam boa parte dos defeitos e isso acontece muito porque são testes que já foram executados (por exemplo, para uma versão anterior do mesmo *software*) e os defeitos já deveriam ter sido detectados nos testes anteriores. Isto não quer dizer que os testes de regressão devam ser totalmente eliminados e sim que a eficácia dos testes de regressão em termos de capacidade de detecção de novos defeitos é inferior à dos outros testes.

1.8 Avaliação de critérios de saída e divulgação

Do ponto de vista do processo de teste, o monitoramento do andamento dos testes garante a coleta das informações adequadas para atender aos requisitos de divulgação. Isto inclui a medição do andamento até a conclusão. Quando os critérios de saída são definidos nas etapas de planejamento, os critérios "imprescindíveis" e "indicados" podem ser detalhados.

Por exemplo, os critérios podem declarar que "não deverá haver nenhum *bug* em aberto de Prioridade 1 ou Prioridade 2" e que "deve existir uma nota de corte de 95% em todos os casos de teste". Neste caso, o descumprimento dos critérios imprescindíveis ("deverá") deve provocar a reprovação dos critérios de saída, enquanto uma nota de 93% ainda permitiria que o projeto passasse ao nível seguinte. Os critérios de saída devem ser claramente definidos para que possam ser avaliados objetivamente.

O *Test Analyst (TA)* é responsável por fornecer as informações utilizadas pelo *Test Manager (TM)* para a avaliação do progresso até o cumprimento dos critérios de saída e por garantir a exatidão dos dados.

Se, por exemplo, o sistema de gestão de testes fornece os seguintes códigos de *status* para a conclusão do caso de teste:

- Aprovado;
- Reprovado;
- Aprovado com ressalva.

Então, o *Test Analyst (TA)* precisa ser muito claro em relação ao que cada um destes termos significa e deverá aplicar o *status* com coerência. *Aprovado com ressalva* significa que um defeito foi detectado, mas ele não afeta a funcionalidade do sistema? E o defeito de usabilidade que deixa o usuário confuso? Se a nota de corte for um critério de saída imprescindível, reprovar um caso de teste em vez de aprová-lo com ressalva vira um fator crucial.

Também devem ser levados em consideração os casos que foram reprovados, mas cuja causa de reprovação não é um defeito (por exemplo, o ambiente de teste foi configurado inadequadamente). Se houver alguma confusão referente às métricas rastreadas ou ao uso dos valores de *status*, o *Test Analyst (TA)* deve esclarecer isto junto ao *Test Manager (TM)* para que as informações possam ser rastreadas com exatidão e coerência ao longo do projeto.

Não é incomum que o *Test Analyst (TA)* receba um pedido de relatório de *status* durante os ciclos de teste e uma solicitação de contribuição para o relatório final quando o teste for concluído. Talvez isto exija a coleta de métricas dos sistemas de gestão de defeitos e testes e uma avaliação da cobertura e do andamento geral. O *Test Analyst (TA)* deve conseguir usar as ferramentas de divulgação e proporcionar as informações solicitadas para que o *Test Manager (TM)* obtenha as informações necessárias.

1.9 Atividades de fechamento de teste

Assim que se determinar a conclusão da execução do teste, as principais saídas dos trabalhos de teste devem ser captadas e repassadas ao responsável ou arquivadas. Coletivamente, são as chamadas atividades de fechamento de teste. Espera-se que o *Test Analyst (TA)* participe da entrega de produtos de trabalho aos

que precisarem deles. Por exemplo, defeitos conhecidos que tenham sido delegados ou aceitos devem ser comunicados às pessoas que utilizarão e apoiarão o uso do sistema.

Os testes e os ambientes de teste devem ser propiciados aos responsáveis pelos testes de manutenção. Entre os outros produtos de trabalho também está o conjunto de testes de regressão (quer automatizados, quer manuais). As informações sobre os produtos de trabalho de teste devem ser claramente documentadas, inclusive os *links* adequados, e os devidos direitos de acesso devem ser concedidos.

Também se espera que o *Test Analyst (TA)* participe de reuniões retrospectivas (aprendizados) em que as lições importantes (do projeto de teste e de todo o ciclo de vida de desenvolvimento de *software*) podem ser documentadas e planos podem ser definidos para ressaltar o *bom* e eliminar, ou pelo menos controlar, o *ruim*.

O *Test Analyst (TA)* é uma profunda fonte de informações para tais reuniões e deverá participar para que as informações válidas de melhoria de processos sejam coletadas. Se o *Test Manager (TM)* participar da reunião, o *Test Analyst (TA)* deverá repassar as informações pertinentes ao *Test Manager (TM)* para a apresentação de um panorama exato do projeto.

O arquivamento de resultados, registros, relatórios e outros documentos e produtos de trabalho no sistema de gestão de configurações também deve ser realizado. Esta tarefa frequentemente é responsabilidade do *Test Analyst (TA)* e é uma atividade importante de fechamento, particularmente se um futuro projeto precisará utilizar estas informações.

Embora o *Test Manager (TM)* normalmente determine quais informações devem ser arquivadas, o *Test Analyst (TA)* também deve pensar em quais informações seriam necessárias se o projeto tivesse que ser reiniciado no futuro. Pensar em tais informações no final de um projeto pode evitar o desperdício de meses de trabalho quando o projeto for recomeçado no futuro ou com outra equipe.

2. Gestão de testes: responsabilidades do Test Analyst (TA) (90 min)

Palavras-chave

Risco de produto, análise de risco, identificação de riscos, nível de risco, gestão de risco, mitigação de risco, teste baseado em risco, monitoramento de testes, estratégia de teste

Objetivos de aprendizagem da gestão de testes: responsabilidades do Test Analyst (TA)

2.2 Monitoramento e controle do andamento dos testes

TA-2.2.1 (K2): Explicam-se os tipos de informações que deverão ser rastreados durante o teste para possibilitar o devido monitoramento e controle do projeto.

2.3 Teste distribuído, terceirizado e internalizado

TA-2.3.1 (K2): São dados exemplos de boas práticas de comunicação quando se trabalha em um ambiente de teste 24 horas por dia.

2.4 As tarefas do Test Analyst (TA) nos testes baseados em riscos

TA-2.4.1 (K3): Em determinada situação de projeto, participa-se da identificação de riscos, a avaliação de risco é realizada e a devida mitigação de risco é proposta.

2.1 Introdução

Embora existam muitas áreas nas quais o *Test Analyst (TA)* interage com o *Test Manager (TM)* e lhe fornece dados, esta parte se concentra nas áreas específicas do processo de teste nas quais o *Test Analyst (TA)* é um grande contribuinte. Espera-se que o *Test Manager (TM)* busque as informações necessárias do *Test Analyst (TA)*.

2.2 Monitoramento e controle do andamento dos testes

Existem cinco dimensões principais nas quais o andamento dos testes é monitorado:

- Riscos de produto (qualidade);
- Defeitos;
- Testes;
- Cobertura;
- Confiança.

Os riscos de produto, os defeitos, os testes e a cobertura podem ser e frequentemente são medidos e divulgados de modos específicos durante o projeto ou a atividade pelo *Test Analyst (TA)*. A confiança, embora mensurável através de pesquisas, costuma ser divulgada subjetivamente. A coleta das informações necessárias para apoiar estas métricas faz parte do trabalho diário do *Test Analyst (TA)*. É importante se lembrar de que a precisão de tais dados é crucial, já que dados inexatos criarão informações equivocadas sobre as tendências e podem levar a conclusões incorretas. Na pior das hipóteses, dados inexatos resultarão em decisões incorretas por parte da administração e prejudicarão a credibilidade da equipe de teste.

Ao utilizar uma abordagem de teste baseado em risco, o *Test Analyst (TA)* deve rastrear:

- Quais riscos foram mitigados pelo teste;
- Quais riscos são considerados não mitigados.

O rastreamento da mitigação de riscos é realizado frequentemente com uma ferramenta que também rastreia a conclusão do teste (por exemplo, ferramentas de gestão de testes). Isto exige que os riscos identificados sejam mapeados nas condições de teste, que são mapeadas nos casos de teste, que mitigarão os riscos se os casos de teste forem executados e aprovados. Desta maneira, as informações sobre a mitigação de riscos são atualizadas automaticamente à medida que os casos de teste são atualizados. Isto pode ser feito para os testes manuais e os automatizados.

O rastreamento de defeitos costuma ser realizado com uma ferramenta de rastreamento de defeitos. Quando os defeitos são registrados, as informações específicas sobre a classificação de cada defeito são registradas também. Tais informações são utilizadas para produzir tendências e gráficos que indicam o andamento do teste e a qualidade do *software*. As informações sobre a classificação são discutidas mais detalhadamente no capítulo sobre *Gestão de defeitos*. O ciclo de vida pode afetar a quantidade registrada de documentos sobre defeitos e os métodos usados para registrar as informações.

À medida que os testes vão sendo realizados, as informações sobre a situação do caso de teste devem ser registradas. Isto costuma ser feito com a ferramenta de gestão de testes, mas pode ser realizado manualmente, se necessário. As informações sobre os casos de teste podem incluir:

- A situação da criação do caso de teste (por exemplo, modelado, avaliado);
- A situação da execução do caso de teste (por exemplo, aprovado, reprovado, bloqueado, pulado);
- Informações sobre a execução do caso de teste (por exemplo, data e hora, nome do testador, dados utilizados);
- Itens de execução do caso de teste (por exemplo, capturas de tela, registros de acompanhamento).

À maneira dos itens de risco identificados, os casos de teste devem ser mapeados nos itens dos requisitos testados. É importante que o *Test Analyst (TA)* lembre-se de que, se o caso de teste A for mapeado no requisito A e este for o único caso de teste mapeado naquele requisito, quando o caso de teste A for executado e aprovado, o requisito A será considerado atendido. Isto pode ser correto ou incorreto.

Em muitos casos, mais casos de teste são necessários pra testar um requisito totalmente, mas, por conta da limitação de tempo, apenas um subgrupo de tais testes é realmente criado. Por exemplo, se 20 casos de teste eram necessários para testar totalmente a implementação de determinado requisito, mas apenas 10 foram criados e executados, as informações sobre a cobertura dos requisitos indicarão 100% de cobertura quando, na verdade, foi obtida uma cobertura de apenas 50%. O rastreamento exato da cobertura e das situações avaliadas dos próprios requisitos pode ser utilizado como medida de confiança.

A quantidade (e o nível de detalhe) das informações registradas depende de diversos fatores, inclusive o modelo de ciclo de vida de desenvolvimento de *software*. Por exemplo, em projetos ágeis, normalmente menos informações sobre situações serão registradas devido à interação próxima da equipe e a uma maior comunicação em pessoa.

2.3 Teste distribuído, terceirizado e internalizado

Em muitos casos, nem todos os trabalhos de teste são realizados por uma única equipe de teste composta por colaboradores do resto da equipe de projetos no mesmo local que o resto da equipe de projetos. Se os trabalhos de teste acontecerem em vários locais, os trabalhos de teste se chamam distribuídos. Se acontecerem em um só local, podem ser chamados de centralizados.

Se os trabalhos de teste forem realizados em um ou mais locais por pessoas que não são colegas do resto da equipe de projetos e não dividem o mesmo local com a equipe de projetos, podem ser chamados de terceirizados. Se os trabalhos de teste forem realizados por pessoas que dividem o mesmo local com a equipe de projetos, mas que não são seus colegas, podem ser chamados de internalizados.

Ao trabalhar em um projeto em que parte da equipe de teste se distribui em vários locais ou até mesmo em várias empresas, o *Test Analyst (TA)* precisa prestar atenção especial à eficácia da comunicação e à transferência de informações. Algumas organizações trabalham de acordo com um modelo de “teste 24 horas” no qual se espera que a equipe em um fuso horário entregue o trabalho à equipe de outro fuso horário para permitir que o teste continue 24 horas por dia. Isto exige planejamento especial por parte do *Test Analyst (TA)*, que entregará e receberá trabalho. O bom planejamento é importante para entender as responsabilidades, mas é vital garantir a disponibilidade das informações adequadas.

Quando a comunicação verbal não for possível, a comunicação por escrito deve bastar. Isto quer dizer que o *e-mail*, relatórios de situação e o uso eficaz das ferramentas de gestão de testes e de rastreamento de defeitos devem ser empregados. Se a ferramenta de gestão de testes permitir a atribuição de testes a indivíduos, também pode funcionar como ferramenta de agendamento e uma maneira fácil de transferir

trabalho entre as pessoas. Os defeitos que forem registrados exatamente podem ser repassados aos colaboradores para o acompanhamento, quando necessário. A utilização eficaz de tais sistemas de comunicação é vital para uma organização que não pode depender da interação pessoal todo dia.

2.4 As tarefas do Test Analyst (TA) nos testes baseados em riscos

2.4.1 Panorama

No geral, o *Test Manager (TM)* frequentemente tem a responsabilidade de definir e gerenciar uma estratégia de testes baseados em riscos. Normalmente, o *Test Manager (TM)* solicita o envolvimento do *Test Analyst (TA)* para garantir que a abordagem baseada em riscos seja implementada corretamente.

O *Test Analyst (TA)* deve estar envolvido ativamente nas seguintes tarefas de testes baseados em riscos:

- Identificação de riscos;
- Avaliação de riscos;
- Mitigação de riscos.

Estas tarefas são realizadas iterativamente ao longo do ciclo de vida do projeto para lidar com os riscos que surgirem e as mudanças de prioridades e para avaliar e comunicar as situações de risco com regularidade.

O *Test Analyst (TA)* deve trabalhar com a estrutura de testes baseados em riscos definida pelo *Test Manager (TM)* para o projeto. Ele deve compartilhar seu conhecimento dos riscos de domínios de negócios que são inerentes ao projeto, como riscos relacionados a questões comerciais, econômicas e de segurança e a fatores políticos.

2.4.2 Identificação de riscos

Ao recorrer ao maior número possível de *stakeholders*, o processo de identificação de riscos provavelmente detectará o maior número possível de riscos significativos. Como, frequentemente, o *Test Analyst (TA)* possui conhecimentos singulares sobre o domínio de negócios específico do sistema testado, é particularmente indicado para a realização de entrevistas com especialistas e usuários do domínio, a realização de avaliações independentes, a utilização e a facilitação do uso de modelos de risco, a realização de *workshops* sobre risco, a realização de sessões de *brainstorming* com possíveis usuários e usuários atuais, a definição de listas de verificação de teste e o recurso a experiências passadas com sistemas ou projetos parecidos. Em particular, o *Test Analyst (TA)* deve colaborar com os usuários e outros especialistas em domínios para determinar as áreas de risco operacional que devem ser tratadas durante o teste. O *Test Analyst (TA)* também pode ser particularmente útil na identificação de possíveis efeitos de risco sobre usuários e *stakeholders*.

Entre os exemplos de riscos que podem ser identificados em um projeto estão:

- Problemas de acurácia na funcionalidade do *software*, por exemplo, cálculos incorretos;
- Problemas de usabilidade, por exemplo, teclas de atalho insuficientes;
- Problemas de assimilação, por exemplo, falta de instruções para o usuário em pontos de decisão fundamentais;
- Considerações a respeito do teste de características de qualidade específicas são discutidas no capítulo 4 deste *syllabus*.

2.4.3 Avaliação de riscos

Embora a identificação de riscos procure identificar o maior número possível de riscos pertinentes, a avaliação de riscos é o estudo dos riscos identificados. Especificamente, a categorização de cada risco e a determinação da probabilidade e do impacto relacionado a cada risco.

A determinação do nível de risco normalmente envolve a avaliação da probabilidade de ocorrência e do impacto após a ocorrência de cada item de risco. A probabilidade de ocorrência costuma ser interpretada como a probabilidade de que o problema possa existir no sistema testado e será observada quando o sistema estiver sendo produzido. Em outras palavras, provém do risco técnico. O *Technical Test Analyst (TTA)* deve contribuir com a descoberta e a compreensão do possível nível de risco técnico para cada item de risco, enquanto o *Test Analyst (TA)* contribui com a compreensão do possível impacto comercial do problema se ele vir a ocorrer.

O impacto após a ocorrência costuma ser interpretado como a gravidade do efeito sobre os usuários, os clientes ou outros *stakeholders*. Em outras palavras, provém do risco operacional. O *Test Analyst (TA)* deve contribuir com a identificação e a avaliação do possível impacto de cada item de risco sobre o domínio de negócios ou o usuário. Entre os fatores que influenciam o risco operacional estão:

- Frequência de uso da característica afetada;
- Perda de negócios;
- Possível perda ou responsabilidade financeira, ecológica ou social;
- Sanções legais civis ou penais;
- Questões de segurança;
- Multas e cassação de licença;
- Falta de soluções de contorno razoáveis;
- Visibilidade da característica;
- Visibilidade da falha que levou à publicidade negativa e a um possível impacto negativo na reputação;
- Perda de clientela.

Após receber as informações de risco disponíveis, o *Test Analyst (TA)* precisa definir os níveis de risco operacional de acordo com as diretrizes estabelecidas pelo *Test Manager (TM)*. Eles podem ser classificados com palavras (por exemplo, baixo, médio, alto) ou números. A menos que haja uma forma de medir objetivamente o risco em uma escala definida, não pode ser uma medida verdadeiramente quantitativa. Medir a probabilidade e o custo / a consequência com precisão costuma ser difícil, então, normalmente, o nível de risco é determinado qualitativamente.

Números podem ser atribuídos ao valor qualitativo, mas isso não o torna uma verdadeira medida quantitativa. Por exemplo, o *Test Manager (TM)* pode determinar que o risco operacional deve ser categorizado com um valor que vá de 1 a 10, sendo que 1 é o impacto mais alto e, portanto, o de maior risco no negócio. Assim que a probabilidade (a avaliação do risco técnico) e o impacto (a avaliação do risco operacional) forem atribuídos, tais valores podem ser multiplicados para definir a classificação geral de risco de cada item de risco.

Depois, a classificação geral é utilizada para priorizar as atividades de mitigação de risco. Alguns modelos de testes baseados em riscos, como o PRISMA® [vanVeenendaal12], não combinam os valores de risco e permitem que a abordagem ao teste trate dos riscos técnico e operacional separadamente.

2.4.4 Mitigação de riscos

Durante o projeto, o *Test Analyst (TA)* deve procurar fazer o seguinte:

- Reduzir o risco de produto com casos de teste bem modelados que demonstrem, sem sombra de dúvida, se os itens de teste foram aprovados ou reprovados e participar em avaliações de itens de *software*, como requisitos, modelagens e documentos de usuário;
- Implementar as atividades adequadas de mitigação de risco identificadas na estratégia de teste e no plano de teste;
- Reavaliar riscos conhecidos com base em outras informações coletadas à medida que o projeto é implantado, ajustando, conforme for o caso, a probabilidade, o impacto ou os dois;
- Reconhecer novos riscos identificados pelas informações obtidas durante o teste.

Quando se fala de risco de produto (qualidade), o teste é uma forma de mitigação de tais riscos. Ao detectar os defeitos, os testadores reduzem o risco ao ficarem cientes dos defeitos e das oportunidades para lidar com os defeitos antes do lançamento. Se os testadores não encontrarem nenhum defeito, o teste passa a reduzir o risco ao garantir que, em certas condições (isto é, as condições testadas), o sistema funcione corretamente. O *Test Analyst (TA)* ajuda a determinar as opções de mitigação de risco ao investigar as oportunidades de coleta de dados de teste exatos, criando e testando cenários de usuário realistas e realizando ou supervisionando os estudos de usabilidade.

2.4.4.1 Priorização dos testes

O nível do risco também é utilizado para priorizar os testes. Um *Test Analyst (TA)* pode determinar que há um alto risco na área de acurácia transacional em um sistema de contabilidade. Consequentemente, para mitigar tal risco, o testador pode trabalhar com outros especialistas em domínios de negócios para coletar uma boa amostra de dados, cuja exatidão pode ser processada e verificada.

Igualmente, o *Test Analyst (TA)* pode determinar que os problemas de usabilidade representam um grande risco para um novo produto. Em vez de aguardar o teste de aceitação do usuário para descobrir qualquer problema, o *Test Analyst (TA)* pode priorizar a realização de um teste antecipado de usabilidade durante o nível de integração para ajudar a identificar e resolver os problemas de usabilidade no início do teste. Tal priorização deve ser considerada o quanto antes nas etapas de planejamento para que o cronograma consiga contemplar os testes necessários no devido tempo.

Em alguns casos, todos os testes de risco mais alto são executados antes de quaisquer testes de risco mais baixo e os testes são executados por uma ordem estrita de risco (chamada frequentemente de “primeiro por conteúdo”). Em outros casos, uma amostragem é utilizada para selecionar uma amostra de testes com todos os riscos identificados, usando o risco para ponderar a seleção e, ao mesmo tempo, garantindo a cobertura de todos os riscos pelo menos uma vez (chamada frequentemente de “primeiro por abrangência”).

Tanto faz se o teste baseado em risco é executado primeiro por conteúdo ou primeiro por abrangência, é possível que o tempo alocado para o teste termine sem que todos os testes sejam executados. O teste baseado em risco permite que os testadores divulguem os riscos à administração em termos do nível de risco

restante em determinado momento e permite que a administração decida se vai prorrogar o teste ou repassar o risco restante aos usuários, aos clientes, ao *helpdesk* / suporte técnico e /ou à equipe operacional.

2.4.4.2 Ajuste de testes para futuros ciclos de teste

A avaliação de risco não é uma atividade única realizada antes do início da implementação de testes. Trata-se de um processo contínuo. Cada futuro ciclo de vida planejado deve passar por uma nova análise de risco para levar em conta fatores como:

- Qualquer risco de produto novo ou consideravelmente alterado;
- Áreas de instabilidade ou propensas a defeitos detectadas durante o teste;
- Riscos advindos de defeitos corrigidos;
- Defeitos típicos encontrados durante o teste;
- Áreas que não foram testadas o suficiente (baixa cobertura do teste).

Se o teste receber mais tempo, é possível expandir a cobertura dos riscos para áreas de menor risco.

3. Técnicas de teste (825 min)

Palavras-chave

análise de valor limite, gráfico de causa e efeito, teste baseado em *checklist*, método de classificação por árvore, teste combinatório, teste de tabela de decisão, taxonomia de defeitos, técnica baseada em defeitos, análise de domínios, suposição de erros, partição de equivalência, técnica baseada em experiência, teste exploratório, arranjo ortogonal, teste de arranjo ortogonal, teste alternado, teste baseado em requisitos, técnica baseada em especificações, teste de transição de estado, carta de teste, teste de caso de uso, teste de estória de usuário

Objetivos de aprendizagem das técnicas de teste

3.2 Técnicas baseadas em especificações

TA-3.2.1 (K2): Explica-se o uso dos gráficos de causa e efeito.

TA-3.2.2 (K3): Redigem-se casos de teste para determinado item de especificação mediante a aplicação da técnica de modelagem de testes de partição de equivalência para definir um nível de cobertura.

TA-3.2.3 (K3): Redigem-se casos de teste para determinado item de especificação mediante a aplicação da técnica de modelagem de testes de análise de valores-limite para definir um nível de cobertura.

TA-3.2.4 (K3): Redigem-se casos de teste para determinado item de especificação mediante a aplicação da técnica de modelagem de testes da tabela de decisão para definir um nível de cobertura.

TA-3.2.5 (K3): Redigem-se casos de teste para determinado item de especificação mediante a aplicação da técnica de modelagem de testes de transição de estado para definir um nível de cobertura.

TA-3.2.6 (K3): Redigem-se casos de teste para determinado item de especificação mediante a aplicação da técnica de modelagem de testes alternados para definir um nível de cobertura.

TA-3.2.7 (K3): Redigem-se casos de teste para determinado item de especificação mediante a aplicação da técnica de modelagem de testes de classificação por árvore para definir um nível de cobertura.

TA-3.2.8 (K3): Redigem-se casos de teste para determinado item de especificação mediante a aplicação da técnica de modelagem de testes de caso de uso para definir um nível de cobertura.

TA-3.2.9 (K2): Explica-se como as estórias de usuário são utilizadas para orientar os testes em um projeto ágil.

TA-3.2.10 (K3): Redigem-se casos de teste para determinado item de especificação mediante a aplicação da técnica de modelagem de testes de análise de domínios para definir um nível de cobertura.

TA-3.2.11 (K4): Analisam-se o sistema ou as especificações de requisitos para determinar os prováveis tipos de defeitos detectados e selecionam-se as técnicas baseadas em especificações adequadas.

3.3 Técnicas baseadas em defeitos

TA-3.3.1 (K2): Descreve-se a aplicação das técnicas de teste baseado em defeitos e diferencia-se o uso das técnicas baseadas em especificações.

TA-3.3.2 (K4): Analisa-se determinada taxonomia de defeitos para a aplicabilidade em determinada situação com critérios para uma boa taxonomia.

3.4 Técnicas baseadas na experiência

TA-3.4.1 (K2): Explicam-se os princípios de técnicas baseadas na experiência e comparam-se os prós e os contras com as técnicas baseadas em especificações e defeitos.

TA-3.4.2 (K3): Em determinada situação, especificam-se testes exploratórios e explica-se como os resultados podem ser divulgados.

TA-3.4.3 (K4): Em determinada situação de projeto, determinam-se as técnicas baseadas em especificações, em defeitos ou na experiência que devem ser aplicadas para atingir metas específicas.

3.1 Introdução

As técnicas de modelagem de testes consideradas neste capítulo são divididas nas seguintes categorias:

- Técnicas baseadas em especificações (ou baseadas em comportamentos ou caixa-preta);
- Técnicas baseadas em defeitos;
- Técnicas baseadas na experiência.

As técnicas são complementares e podem ser utilizadas conforme for o caso em qualquer atividade de teste, independentemente do nível de teste realizado.

As três categorias de técnicas podem ser utilizadas para testar as características de qualidade funcionais ou não funcionais. O teste de características não funcionais é discutido no capítulo seguinte.

As técnicas de modelagem de testes discutidas nestas partes podem se concentrar principalmente na determinação de dados de testes ideais (por exemplo, partições de equivalência) ou na obtenção de sequências de teste (por exemplo, modelos de estados). A combinação de técnicas para a criação de casos de teste completos é comum.

3.2 Técnicas baseadas em especificações

As técnicas baseadas em especificações são aplicadas às condições de teste para obter casos de teste baseados em uma análise da base de teste referente a um componente ou um sistema sem alusão à sua estrutura interna.

Entre as características comuns das técnicas baseadas em especificações estão:

- Modelos, por exemplo, diagramas de transição de estado e tabelas de decisão, são criados durante a modelagem de testes de acordo com a técnica de teste;
- As condições de testes são obtidas sistematicamente destes modelos.

Algumas técnicas também dispõem de critérios de cobertura, que podem ser utilizados para medir as atividades de modelagem e execução de testes. O atendimento total aos critérios de cobertura não significa que todos os testes foram concluídos e, sim, que o modelo deixou de recomendar testes para aumentar a cobertura com base em tal técnica.

Os testes baseados em especificações costumam se fundamentar nos documentos de requisitos do sistema. Como as especificações de requisitos devem especificar como o sistema deve se comportar, particularmente na área da funcionalidade, a obtenção de testes a partir dos requisitos frequentemente faz parte do teste do comportamento do sistema. Em alguns casos, os requisitos podem não estar documentados, mas existem requisitos implícitos, como a substituição da funcionalidade de um sistema legado.

Há uma série de técnicas de teste baseadas em especificações. As técnicas têm como objetivo tipos diferentes de *software* e cenários. As seções abaixo mostram a aplicabilidade de cada técnica, algumas limitações e dificuldades com as quais o *Test Analyst (TA)* pode se deparar, o método de medição da cobertura de teste e os tipos de defeitos almejados.

3.2.1 Partição de equivalência

A partição de equivalência é utilizada para reduzir o número necessário de casos de teste para testar com eficácia o tratamento de entradas, saídas, valores internos e valores relacionados ao tempo. A partição é utilizada para criar classes de equivalência (frequentemente chamadas de partições de equivalência), que são criadas a partir de conjuntos de valores processados da mesma maneira. Ao selecionar um valor representativo de uma partição, supõe-se a cobertura de todos os itens da mesma partição.

Aplicabilidade

Esta técnica pode ser aplicada a qualquer nível de teste e é adequada quando se espera que todas as partes de um conjunto de valores a ser testado serão tratadas da mesma maneira e quando os conjuntos de valores utilizados na aplicação não interagem entre si.

A seleção dos conjuntos de valores vale para partições válidas e inválidas (isto é, partições que contenham valores que devem ser considerados inválidos para o *software* testado). Esta técnica funciona melhor quando for utilizada junto com a análise de valores-limite, que expande os valores de teste e inclui os que ficam nos limites das partições. Normalmente, esta técnica é utilizada em testes básicos de um novo pacote ou um novo *release* já que ela determina se a funcionalidade básica está funcionando.

Limitações / dificuldades

Se a suposição for incorreta e os valores na partição não forem tratados exatamente da mesma maneira, a técnica pode acabar ignorando os defeitos. Também é importante escolher as partições com cuidado. Por exemplo, um campo de entrada que aceitar números positivos e negativos seria mais bem testado como duas partições válidas, uma para os números positivos e outra para os negativos, em função da probabilidade de diferença no tratamento. Pode haver outra partição dependendo da admissão do zero. É importante que o *Test Analyst (TA)* entenda o processamento subjacente para determinar a melhor partição dos valores.

Cobertura

Determina-se a cobertura pegando o número de partições para o qual o valor foi testado e dividindo-o pelo número identificado de partições. O uso de vários valores para uma partição não aumenta a porcentagem da cobertura.

Tipos de defeitos

Esta técnica encontra defeitos funcionais no tratamento de diversos valores de dados.

3.2.2 Análise de valores-limite

A análise de valores-limite é usada para testar os valores que existem nos limites das partições de equivalência ordenadas. Existem duas maneiras de abordar a análise de valores-limite: teste com dois ou três valores. No teste com dois valores, o valor-limite (no limite) e o valor que fica um pouco acima do limite (pelo menor incremento possível) são utilizados. Por exemplo, se a partição incluir valores de 1 a 10 em incrementos de 0,5, os valores do teste com dois valores referentes ao limite superior seriam 10 e 10,5. Os valores de teste do limite inferior seriam 1 e 0,5. Os limites são definidos pelos valores máximos e mínimos na partição de equivalência definida.

Nos testes com três valores, os valores antes do limite, no limite e acima do limite são utilizados. No exemplo anterior, os testes de limite superior incluem 9,5, 10 e 10,5. Os testes de limite inferior incluem 1,5, 1 e 0,5. A decisão de usar dois ou três valores-limite deve se basear no risco relacionado ao item testado, sendo que a abordagem de três limites é usada para os itens de maior risco.

Aplicabilidade

Esta técnica vale em qualquer nível de teste e é adequada na presença de partições de equivalência ordenadas. A ordenação é necessária em função do conceito de estar no limite e acima dele. Por exemplo, uma variedade de número é uma partição ordenada. Uma partição que consiste em todos os objetos retangulares não é uma partição ordenada e não tem valores-limite. Além das variedades de números, a análise de valores-limite pode ser aplicada ao seguinte:

- Atributos numéricos de variáveis não numéricas (por exemplo, comprimento);
- Laços, inclusive os que existem em casos de uso;
- Estruturas de dados armazenados;
- Objetos físicos (inclusive a memória);
- Atividades determinadas pelo tempo.

Limitações / dificuldades

Como a acurácia desta técnica depende da identificação precisa das partições de equivalência, fica sujeito às mesmas limitações e dificuldades.

O *Test Analyst (TA)* também deve estar ciente dos incrementos nos valores válidos e inválidos para conseguir determinar exatamente os valores que serão testados. Apenas as partições ordenadas podem ser utilizadas para a análise de valores-limite, mas não se limitam a uma variedade de entradas válidas. Por exemplo, quando um teste é realizado com o número de células suportado por uma planilha, há uma partição que contém o número de células que chega ao máximo permitido (o limite) e outra partição que começa pela célula acima do máximo (acima do limite).

Cobertura

Determina-se a cobertura pegando o número de condições-limite que foram testadas e dividindo-o pelo número de condições-limite identificadas (quer com o método dos dois valores, quer com o de três valores). Isto estipulará a porcentagem de cobertura do teste de valores-limite.

Tipos de defeitos

A análise de valores-limite detecta, de maneira confiável, o deslocamento ou a omissão de limites e pode encontrar casos de limites extra. Esta técnica detecta defeitos relacionados ao tratamento dos valores-limite, particularmente erros com lógica menor e maior (isto é, o deslocamento). Também pode ser utilizada para detectar defeitos não funcionais, por exemplo, a tolerância de limites de carga (por exemplo, o sistema suporta 10.000 usuários ao mesmo tempo).

3.2.3 Tabelas de decisão

As tabelas de decisão são utilizadas para testar a interação entre as combinações de condições. As tabelas de decisão são um método claro para verificar a realização de testes com todas as combinações de condições pertinentes e para garantir que todas as possíveis combinações sejam tratadas pelo *software* em teste. A

meta do teste de tabela de decisão consiste em garantir que todas as combinações de condições, relacionamentos e restrições sejam testadas. Ao tentar testar todas as combinações possíveis, as tabelas de decisão podem ficar bem grandes. Um método para a redução inteligente do número de combinações de todas as possibilidades para as que são "interessantes" se chama teste de tabela de decisão recolhida. Quando esta técnica é utilizada, as combinações são reduzidas para que produzem saídas diferentes ao remover conjuntos de condições que não são relevantes para o resultado. São removidos os testes redundantes ou os testes nos quais a combinação de condições não é possível. A decisão de utilizar tabelas de decisão cheias ou recolhidas costuma se estar baseada no risco. [Copeland03]

Aplicabilidade

Esta técnica costuma ser aplicada aos níveis de teste de integração, sistema e aceitação. Dependendo do código, também pode valer para teste de componentes quando um componente for responsável por um conjunto de lógicas decisórias. A técnica é particularmente útil quando os requisitos são apresentados na forma de fluxogramas ou tabelas de regras de negócios. As tabelas de decisão também são uma técnica de definição de requisitos e algumas especificações de requisitos podem já estar neste formato. Mesmo quando os requisitos não são apresentados em forma de tabela ou fluxograma, as combinações de condições costumam ser encontradas no memorial descritivo. Ao projetar tabelas de decisão, é importante considerar as combinações de condições definidas e as que não foram expressamente definidas, mas que existem. Para modelar uma tabela de decisão válida, o testador deve obter todas as consequências de todas as combinações de condições das especificações ou do oráculo de teste. Apenas quando todas as condições de interação forem consideradas, a tabela de decisão pode ser utilizada como boa ferramenta de modelagem de testes.

Limitações / dificuldades

Encontrar todas as condições que interagem entre si pode ser difícil, especialmente quando os requisitos não são bem definidos ou não existem. Não é incomum preparar um conjunto de condições e determinar que o resultado esperado é desconhecido.

Cobertura

A cobertura mínima de teste para uma tabela de decisão consiste em ter um caso de teste para cada coluna. Isto supõe a inexistência de condições compostas e que todas as combinações de condições possíveis podem ser registradas em uma coluna.

Ao determinar os testes de uma tabela de decisão, também é importante pensar nas condições-limite que devem ser testadas. As condições-limite podem levar a um aumento do número necessário de casos de teste para testar o *software* adequadamente. A análise de valores-limite e a partição de equivalência complementam a técnica da tabela de decisão.

Tipos de defeitos

Entre os defeitos comuns estão o processamento incorreto com base em combinações específicas de condições que produzem resultados inesperados. Durante a criação das tabelas de decisão, os defeitos podem ser encontrados no documento de especificações. Os tipos mais comuns de defeitos são as omissões (não há nenhuma informação referente ao que realmente deve acontecer em determinada situação) e as contradições. O teste também pode encontrar problemas nas combinações de condições que não são tratadas ou que não são bem tratadas.

3.2.4 Gráficos de causa e efeito

Os gráficos de causa e efeito podem ser gerados a partir de qualquer fonte que descreva a lógica funcional (isto é, as "regras") de um programa, como estórias de usuários ou fluxogramas. Podem ser úteis para obter um panorama gráfico da estrutura lógica de um programa e normalmente servem de base para a criação de tabelas de decisão. A captura das decisões em gráficos de causa e efeito e / ou tabelas de decisão possibilita a cobertura sistemática de teste da lógica do programa.

Aplicabilidade

Os gráficos de causa e efeito são aplicados nas mesmas situações que as tabelas de decisão e também valem para os mesmos níveis de teste. Em particular, um gráfico de causa e efeito mostra combinações de condições que produzem resultados (causalidade), combinações de condições que excluem resultados (não), condições múltiplas que devem ser verdadeiras para produzirem um resultado (e) e condições alternativas que podem ser verdadeiras para produzirem um resultado específico (ou). Pode ser mais fácil visualizar os relacionamentos em um gráfico de causa e efeito do que um memorial descritivo.

Limitações / dificuldades

Os gráficos de causa e efeito exigem mais tempo e esforço para a assimilação em comparação com outras técnicas de modelagem de testes. Além disso, exigem suporte das ferramentas. Os gráficos de causa e efeito possuem uma notação específica que deve ser compreendida pelo autor e pelo leitor do gráfico.

Cobertura

Cada linha de causa e efeito possível deve ser testada, inclusive as condições de combinação, para alcançar a cobertura mínima. Os gráficos de causa e efeito incluem um meio para definir as restrições nos dados e no fluxo da lógica.

Tipos de defeitos

Tais gráficos detectam os mesmos tipos de defeitos combinatórios encontrados com as tabelas de decisão. Além disso, a criação dos gráficos facilita a definição do nível necessário de detalhe na base de teste e, assim, contribui com o aprimoramento do o detalhamento e da qualidade da base de teste e facilita a identificação de requisitos ausentes por parte do testador.

3.2.5 Teste de transição de estados

O teste de transição de estados é utilizado para testar a capacidade que o *software* tem de entrar em e sair de estados definidos através de transições válidas e inválidas. Os eventos provocam a transição do *software* de um estado para outro e a realização de ações.

Os eventos podem ser qualificados por condições (chamadas às vezes de condições de proteção ou proteções de transição), que influenciam o caminho da transição a ser tomado. Por exemplo, um evento de *login* com uma combinação válida de usuário / senha resultará em uma transição diferente de um evento de *login* com uma senha inválida.

As transições de estado são rastreadas em um diagrama de transição de estados que mostra todas as transições válidas entre estados em formato gráfico ou uma tabela de estados que mostra todas as possíveis transições, tanto as válidas quanto as inválidas.

Aplicabilidade

O teste de transição de estado vale para qualquer *software* que tenha estados definidos e eventos que causarão as transições entre estados (por exemplo, a mudança de telas). O teste de transição de estado pode ser utilizado em qualquer nível do teste. *Software* embarcado, *web software* e qualquer tipo de *software* transacional são bons candidatos para este tipo de teste. Os sistemas de controle, isto é, os controladores de semáforos, também são bons candidatos para este tipo de teste.

Limitações / dificuldades

A determinação dos estados é frequentemente a parte mais difícil da definição da tabela ou do diagrama de estados. Quando o *software* possui uma interface de usuário, as diversas telas exibidas ao usuário são frequentemente utilizadas para definir os estados. Em relação ao *software* embarcado, os estados podem depender dos estados com os quais o *hardware* lidará.

Além dos próprios estados, a unidade básica do teste de transição de estados é a transição individual, também conhecida como *0-switch*. O simples teste com todas as transições detectará alguns tipos de defeitos na transição de estados, porém outros podem ser descobertos ao testar as sequências de transações. Uma sequência de duas transições sucessivas é chamada de *1-switch*. Uma sequência de três transições sucessivas se chama *2-switch* e assim por diante. [Às vezes, as *switches* são chamadas de *N-1 switches*, onde o N representa o número de transições que será realizado. Uma única transição, por exemplo (uma *0-switch*) seria uma *1-1 switch*.] [Bath08]

Cobertura

À maneira de outros tipos de técnicas de teste, existe uma hierarquia de níveis de cobertura de teste. O grau mínimo aceitável de cobertura consiste em ter visitado todos os estados e ter realizado todas as transições. Uma cobertura de transição de 100% (também conhecida como cobertura *0-switch* 100% ou cobertura de desvio lógico 100%) garantirá que todos os estados serão visitados e todas as transições serão realizadas, a menos que o modelo de modelagem de sistemas ou de transição de dados (diagrama ou tabela) tenha algum defeito. Dependendo dos relacionamentos entre estados e transições, talvez seja necessário realizar algumas transições mais de uma vez para executar outras transições uma só vez.

O termo cobertura *N-switch* diz respeito ao número de transições cobertas. Por exemplo, uma cobertura *1-switch* 100% exige que todas as sequências válidas de duas transições sucessivas tenham sido testadas pelo menos uma vez. Tal teste pode provocar alguns tipos de falhas que a cobertura *0-switch* 100% teria ignorado.

A cobertura de ida e volta corresponde às situações em que as sequências de transições formam *loops* ou laços. A cobertura de ida e volta 100% é alcançada quando todos os laços de qualquer estado de volta ao mesmo estado foram testados. Todos os estados incluídos nos laços precisam ser testados.

Em relação a qualquer uma de estas abordagens, um grau ainda maior de cobertura tentará incluir todas as transições inválidas. Os requisitos de cobertura e os conjuntos de cobertura para o teste de transição de estado devem verificar a inclusão de transições inválidas.

Tipos de defeitos

Entre os defeitos comuns estão o processamento incorreto no estado atual, que é um resultado do processamento ocorrido no estado anterior, de transições incorretas ou não suportadas, de estados sem saídas e da necessidade de estados ou transições que não existem. Durante a criação do modelo da máquina

de estado, os defeitos podem ser encontrados no documento de especificações. Os tipos mais comuns de defeitos são as omissões (não há nenhuma informação referente ao que realmente deve acontecer em determinada situação) e as contradições.

3.2.6 Técnicas de testes combinatórios

O teste combinatório é utilizado quando o *software* é testado com diversos parâmetros, cada qual com diversos valores, o que gera mais combinações do que é possível testar dentro do prazo.

Os parâmetros devem ser independentes e compatíveis no sentido de que qualquer opção de qualquer fator pode ser combinada com qualquer opção de qualquer outro fator. As árvores de classificação permitem a exclusão de algumas combinações se certas opções forem incompatíveis. Isto não parte da premissa de que os fatores combinados não afetaram um ao outro. É bem possível que isto aconteça, mas eles devem afetar um ao outro de uma maneira aceitável.

O teste combinatório serve para identificar um subconjunto propício de tais combinações para atingir um nível pré-determinado de cobertura. O número de itens que devem ser incluídos nas combinações pode ser selecionado pelo *Test Analyst (TA)*, inclusive itens únicos, pares, trios ou mais [Copeland03].

Há uma série de ferramentas disponíveis para ajudar o Test Analyst (TA) nesta tarefa (vide www.pairwise.org para obter amostras). As ferramentas exigem a listagem dos parâmetros e de seus valores (teste alternado e teste de arranjo ortogonal) ou a sua representação em formato gráfico (árvores de classificação) [Grochtmann94].

O teste alternado é um método aplicado nos testes com pares de variáveis em combinação. Os arranjos ortogonais, tabelas matematicamente exatas que permitem que o *Test Analyst (TA)* substitua os itens que serão testados pelas variáveis no arranjo, produzindo um conjunto de combinações que atingirá um nível de cobertura quando testado [Koomen06], são predefinidos.

As ferramentas da classificação por árvores permitem que o *Test Analyst (TA)* defina o tamanho das combinações que serão testadas (isto é, combinações de dois valores, três valores *etc.*).

Aplicabilidade

O problema de ter combinações de valores de parâmetros demais se manifesta em pelo menos duas situações diferentes que dizem respeito ao teste. Alguns casos de teste contêm diversos parâmetros, cada qual com uma série de valores possíveis, por exemplo, uma tela com diversos campos de entrada. Neste caso, as combinações de valores de parâmetros foram os dados de entrada para os casos de teste.

Além disso, alguns sistemas podem ser configurados em uma série de dimensões, o que resulta em um espaço de configuração possivelmente grande. Em ambas as situações, o teste combinatório pode ser utilizado para identificar um subconjunto de combinações exequíveis em tamanho.

Em relação a parâmetros com um grande número de valores, a partição de equivalência de classes ou algum outro mecanismo de seleção pode ser aplicado em cada parâmetro individualmente para reduzir o número de valores para cada parâmetro antes que o teste combinatório seja aplicado para reduzir o conjunto de combinações resultantes.

Estas técnicas costumam ser aplicadas aos níveis de teste de integração, sistema e integração de sistemas.

Limitações / dificuldades

A grande limitação destas técnicas é a premissa de que os resultados de alguns testes representam todos os testes e de que esses poucos testes representam o uso esperado. Se houver uma interação inesperada entre certas variáveis, pode passar despercebida com este tipo de teste se a combinação específica não for testada. Talvez seja difícil explicar estas técnicas para um público leigo, já que a redução lógica dos testes talvez não seja compreendida.

A identificação dos parâmetros e de seus respectivos valores é difícil às vezes. É difícil encontrar manualmente um conjunto mínimo de combinações para satisfazer certo nível de cobertura. Normalmente, as ferramentas são utilizadas para encontrar um conjunto mínimo de combinações. Algumas ferramentas apoiam a capacidade de forçar a inclusão ou exclusão de algumas (sub)combinações na / da seleção final de combinações. Este recurso pode ser utilizado pelo *Test Analyst (TA)* para enfatizar fatores com base no conhecimento de domínios ou nas informações sobre o uso do produto ou tirar ênfase deles.

Cobertura

Existem diversos níveis de cobertura. O nível mais baixo de cobertura se chama *1-wise* ou cobertura única. Ele exige a presença de cada valor de cada parâmetro em pelo menos uma das combinações escolhidas. O nível seguinte de cobertura se chama *2-wise* ou cobertura alternada. Exige a inclusão de todos os pares de valores de dois parâmetros em pelo menos uma combinação. Esta ideia pode ser generalizada para abranger a cobertura *N-wise*, que exige a inclusão de todas as sub combinações de valores de qualquer conjunto de parâmetros N no conjunto de combinações selecionadas.

Quanto maior o N, mais combinações são necessárias para chegar a 100% de cobertura. A cobertura mínima com estas técnicas consiste em ter um caso de teste para cada combinação produzida pela ferramenta.

Tipos de defeitos

O tipo de defeito mais comum detectado com este tipo de teste é o relacionado aos valores combinados de diversos parâmetros.

3.2.7 Teste de caso de uso

O teste de caso de uso realiza testes transacionais baseados em cenários que devem emular o uso do sistema. Os casos de uso são definidos em termos de interações entre os atores e o sistema que atingem alguma meta. Os atores podem ser os usuários ou sistemas externos.

Aplicabilidade

O teste de caso de uso costuma ser aplicado aos níveis de teste de sistema e aceitação. Pode ser utilizado no teste de integração dependendo do nível de integração e até mesmo no teste de componente dependendo do comportamento do componente. Frequentemente, os casos de uso também servem de base para o teste de desempenho porque retratam o uso realista do sistema. Os cenários descritos nos casos de uso podem ser atribuídos a usuários virtuais para a criação de uma carga realista no sistema.

Limitações / dificuldades

Para serem válidos, os casos de uso devem retratar transações de usuário realistas. Esta informação deve partir de um usuário ou um representante de usuário. O valor de um caso de uso é reduzido se o caso de uso

não refletir com precisão as atividades do usuário real. Uma definição precisa dos diversos caminhos alternativos (fluxos) é importante para que a cobertura do teste seja completa. Os casos de uso devem ser considerados diretrizes, mas não uma definição completa do que deve ser testado, já que talvez não sejam uma definição clara do conjunto total de requisitos. Além disso, talvez seja favorável criar outros modelos, como fluxogramas, a partir do memorial de casos de uso para aprimorar a acurácia do teste e verificar o próprio caso de uso.

Cobertura

A cobertura mínima de um caso de uso consiste em ter um caso de teste para o caminho principal (positivo) e um caso de teste para cada caminho ou fluxo alternativo. Os caminhos alternativos incluem os caminhos de exceção e falha. Às vezes, os caminhos alternativos aparecem como extensões do caminho principal. A porcentagem de cobertura é determinada pegando o número de caminhos testados e dividindo-o pelo número total de caminhos principal e alternativos.

Tipos de defeitos

Entre os defeitos estão o tratamento inadequado de cenários definidos, o tratamento inexistente de caminhos alternativos, o processamento incorreto das condições apresentadas e a notificação estranha ou incorreta de erros.

3.2.8 Teste de estória de usuário

Em algumas metodologias ágeis, como a SCRUM, os requisitos são preparados na forma de estórias de usuário que descrevem pequenas unidades funcionais que podem ser modeladas, desenvolvidas, testadas e demonstradas em uma única iteração [Cohn04].

As estórias de usuário incluem uma descrição da funcionalidade a ser implementada e qualquer critério não funcional e também incluem critérios de aceitação que precisam ser atendidos para que a estória de usuário seja considerada completa.

Aplicabilidade

As estórias de usuário são utilizadas principalmente em ambientes iterativos e incrementais ágeis e afins. São usadas em testes funcionais e não funcionais. As estórias de usuário são usadas em testes de todos os níveis com a expectativa de que o desenvolvedor demonstre a funcionalidade implementada para a estória de usuário antes da entrega do código aos integrantes da equipe com o próximo nível de tarefas de teste (por exemplo, testes de integração e de desempenho).

Limitações / dificuldades

Como as estórias são pequenos incrementos de funcionalidade, pode haver um requisito para a produção de controladores e simuladores a fim de realmente testar a funcionalidade entregue. Isto costuma exigir a capacidade de programar e utilizar ferramentas que facilitarão o teste, como ferramentas de teste de interface de programação de aplicativos (API, na sigla em inglês).

O desenvolver é normalmente responsável pela criação dos controladores e dos simuladores, embora um *Technical Test Analyst (TTA)* também possa estar envolvido na produção do código e na utilização de ferramentas de teste API. Se um modelo de integração contínua for utilizado, como acontece com a maioria dos projetos ágeis, a necessidade de controladores e simuladores é minimizada.

Cobertura

A cobertura mínima de uma estória de usuário serve para verificar se todos os critérios de aceitação especificados foram atendidos.

Tipos de defeitos

Normalmente, os defeitos são funcionais, o que quer dizer que o *software* deixa de executar a funcionalidade especificada. Além disso, existem defeitos com problemas de integração da funcionalidade à nova estória com a funcionalidade que já existe. Como as estórias podem ser desenvolvidas independentemente, podem existir problemas de desempenho, interface e tratamento de erros.

É importante que o *Test Analyst (TA)* teste a funcionalidade individual fornecida e a integração sempre que uma nova estória for liberada para teste.

Análise de domínios

Um domínio é um conjunto definido de valores. O conjunto pode ser definido como uma faixa de valores de uma única variável (um domínio unidimensional, por exemplo, “homens com mais de 24 anos e menos de 66 anos”) ou como faixas de valores de variáveis que interagem entre si (um domínio multidimensional, por exemplo, “homens com mais de 24 anos e menos de 66 anos E que pesem mais de 69 kg e menos de 90 kg”). Cada caso de teste de um domínio multidimensional deve incluir os valores adequados de cada variável envolvida.

A análise de domínios de um domínio unidimensional normalmente usa a partição de equivalência e a análise de valores-limite. Assim que as partições forem definidas, o *Test Analyst (TA)* seleciona os valores de cada partição que representam um valor que está dentro da partição (IN), fora da partição (OUT), no limite da partição (ON) e um pouco fora do limite da partição (OFF). Ao determinar estes valores, cada partição é testada junto com as condições-limite. [Black07]

Com os domínios multidimensionais, o número de casos de teste gerados por estes métodos aumenta exponencialmente com o número de variáveis envolvidas, enquanto uma abordagem baseada na teoria dos domínios leva a um crescimento linear.

Além disso, como a abordagem formal incorpora a teoria dos defeitos (um modelo de falha), o que a partição de equivalência e a análise de valores-limite não têm, seu conjunto menor de testes encontrará defeitos em domínios multidimensionais que o conjunto de testes heurístico maior provavelmente ignoraria.

Ao lidar com domínios multidimensionais, o modelo de teste pode ser construído como tabela de decisão (ou “matriz de domínios”). A identificação dos valores de casos de teste referentes a domínios multidimensionais acima de três dimensões provavelmente exigirá suporte computacional.

Aplicabilidade

A análise de domínios combina as técnicas utilizadas nas tabelas de decisão, a partição de equivalência e análise de valores-limite para criar um conjunto menor de testes que ainda cobre as áreas importantes e as prováveis áreas de falha. É frequentemente aplicada nos casos em que as tabelas de decisão seriam um incômodo por conta do grande número de variáveis que possivelmente interagiriam entre si.

A análise de domínios pode ser realizada em qualquer nível de teste, mas é mais frequentemente aplicada nos níveis de teste de integração e de sistemas.

Limitações / dificuldades

A realização de análises de domínios completas exige uma boa compreensão do *software* para identificar os diversos domínios e a possível interação entre os domínios. Se um domínio não for identificado, o teste pode ficar bastante incompleto, mas é provável que o domínio seja detectado porque as variáveis OFF e OUT podem aparecer do domínio não detectado. A análise de domínios é uma boa técnica na hora de trabalhar com um desenvolvedor para definir as áreas de teste.

Cobertura

A cobertura mínima da análise de domínios consiste em ter um teste para cada valor IN, OUT, ON e OFF em cada domínio. Quando os valores se sobrepuserem (por exemplo, o valor OUT de um domínio é um valor IN em outro domínio), não há nenhuma necessidade de duplicar os testes. Por conta disto, os testes que são realmente necessários frequentemente ficam abaixo de quatro por domínio.

Tipos de defeitos

Entre os defeitos estão problemas funcionais no domínio, o tratamento de valores-limite, problemas de interação entre as variáveis e o tratamento de erros (particularmente os valores que não aparecem em um domínio válido).

3.2.9 Combinação de técnicas

Às vezes as técnicas são combinadas para criar casos de teste. Por exemplo, as condições identificadas em uma tabela de decisão podem ficar sujeitas à partição de equivalência para a detecção de várias maneiras de cumprimento de uma condição. Então, os casos de teste cobririam não só todas as combinações de condições, mas também, para as condições que foram partilhadas, outros casos de teste seriam gerados para cobrir as partições de equivalência.

Ao selecionar a técnica específica a ser aplicada, o *Test Analyst (TA)* deve considerar a aplicabilidade da técnica, as limitações e as dificuldades e as metas do teste em termos de cobertura e defeitos a serem detectados. Talvez a "melhor" técnica para determinada situação não exista. A combinação de técnicas frequentemente proporcionará a cobertura mais completa, contanto que existam tempo e habilidade suficientes para aplicar as técnicas corretamente.

3.3 Técnicas baseadas em defeitos

3.3.1 Uso das técnicas baseadas em defeitos

Uma técnica de modelagem de testes baseada em defeitos é aquela em que o tipo de defeito procurado serve de base para a modelagem de testes, sendo que os testes são obtidos sistematicamente do que se sabe sobre o tipo de defeito.

Diferentemente do teste baseado em especificações, que obtém os testes das especificações, o teste baseado em defeitos obtém os testes das taxonomias de defeitos (isto é, listas divididas em categorias) que podem ser completamente independentes do *software* testado. As taxonomias podem incluir relações de tipos de defeitos, causas-raiz, sintomas de falha e outros dados relacionados a defeitos.

O teste baseado em defeitos também pode utilizar listas de riscos identificados e cenários de risco como base para a objetivação dos testes. Esta técnica de teste permite que o testador objetive um tipo específico de defeito ou trabalhe sistematicamente com uma taxonomia de defeitos conhecidos e comuns de um tipo específico.

O *Test Analyst (TA)* utiliza os dados da taxonomia para determinar a meta do teste, que consiste em detectar um tipo específico de defeito. Com base nestas informações, o *Test Analyst (TA)* criará os casos de teste e as condições de teste que provocarão a manifestação do defeito, se ele existir.

Aplicabilidade

O teste baseado em defeito pode ser aplicado em qualquer nível de teste, mas, normalmente, é aplicado durante o teste de sistema. Existem taxonomias padrão que correspondem a vários tipos de *software*. Este tipo específico de teste com não-produtos contribui para o aproveitamento dos conhecimentos do setor para a obtenção de testes específicos. Ao aderir às taxonomias próprias do setor, as métricas referentes à ocorrência de defeitos podem ser rastreadas em diversos projetos e até mesmo em diversas organizações.

Limitações / dificuldades

Existem várias taxonomias de defeitos e elas podem se concentrar em tipos específicos de teste, como a usabilidade. É importante escolher uma taxonomia que possa ser aplicada ao *software* testado, se houver disponibilidade. Por exemplo, talvez não haja nenhuma taxonomia disponível para um *software* inovador. Algumas organizações compilaram suas próprias taxonomias de defeitos prováveis ou frequentemente vistos. Independentemente da taxonomia utilizada, é importante definir a cobertura esperada antes de dar início ao teste.

Cobertura

A técnica estipula os critérios de cobertura que são utilizados para determinar quando todos os casos de teste úteis foram identificados. Na prática, os critérios de cobertura de técnicas baseadas em defeitos tendem a ser menos sistemáticos do que as técnicas baseadas em especificações, já que apenas as regras gerais de cobertura são determinadas e a decisão específica sobre o que constitui o limite da cobertura útil é discricionária. À maneira de outras técnicas, os critérios de cobertura não significam que todo o conjunto de testes está completo, mas que os defeitos considerados deixaram de indicar testes úteis baseados em tal técnica.

Tipos de defeitos

Os tipos de defeitos descobertos costumam depender da taxonomia utilizada. Se a taxonomia da interface de usuário for utilizada, a maioria dos defeitos descobertos provavelmente manteria relação com a interface de usuário, mas outros defeitos podem ser descobertos como subproduto do teste específico.

3.3.2 Taxonomias de defeitos

As taxonomias de defeitos são relações (listas) de tipos de defeitos divididas em categorias. As listas podem ser muito gerais e servem de diretrizes de alto nível ou podem ser muito específicas. Por exemplo, a taxonomia dos defeitos da interface do usuário pode conter itens gerais, como funcionalidade, tratamento de erros, visualização de gráficos e desempenho. Uma taxonomia detalhada pode incluir uma lista de todas

os possíveis objetos da interface de usuário (particularmente quando se trata de uma interface de usuário gráfica) e designar o tratamento inadequado de tais objetos, como:

Campo de texto

- Os dados válidos não são aceitos;
- Os dados inválidos são aceitos;
- O comprimento da entrada não é verificado;
- Os caracteres especiais não são detectados;
- As mensagens de erro do usuário não são informativas;
- O usuário não consegue corrigir dados errôneos;
- As regras não são aplicadas.

Campo de dados

- Os dados válidos não são aceitos;
- Os dados inválidos não são rejeitados;
- As faixas de dados não são verificadas;
- Os dados de precisão não são tratados corretamente (por exemplo, hh:mm:ss);
- O usuário não consegue corrigir dados errôneos;
- As regras não são aplicadas (por exemplo, a data final deve ser posterior à data de início).

Muitas taxonomias de defeitos estão disponíveis e vão de taxonomias formais que podem ser adquiridas às modeladas para fins específicos por diversas organizações. As taxonomias de defeitos desenvolvidas internamente também podem ser utilizadas para objetivar defeitos específicos normalmente encontrados na organização. Ao criar uma nova taxonomia de defeitos ou customizar uma disponível, é importante definir primeiro as metas ou os objetivos da taxonomia. Por exemplo, a meta pode ser a de identificar problemas na interface de usuário que tenham sido descobertos em sistemas de produção ou identificar problemas relacionados ao tratamento dos campos de entrada.

Para a criação de uma taxonomia:

1. Crie uma meta e defina o nível desejado de detalhamento;
2. Selecione determinada taxonomia para servir de base;
3. Defina valores e defeitos comuns que a organização presencia e /ou que são encontrados na prática externamente.

Quanto mais detalhada for a taxonomia, mais tempo levará para desenvolvê-la e mantê-la, mas resultará em um nível maior de reprodutibilidade nos resultados dos testes. Taxonomias detalhadas podem ser redundantes, mas permitem que a equipe de teste divida o teste sem perda de informações ou cobertura.

Assim que a taxonomia adequada tiver sido selecionada, pode ser utilizada para criar condições de teste e casos de teste. Uma taxonomia baseada em risco pode ajudar o teste a se concentrar em uma área de risco específica. As taxonomias também podem ser utilizadas em áreas não funcionais, como usabilidade, desempenho *etc*. As listas de taxonomia estão disponíveis em várias publicações do Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) e na Internet.

3.4 Técnicas baseadas na experiência

Os testes baseados na experiência utilizam a habilidade e a intuição dos testadores, juntamente com sua experiência com aplicações ou tecnologias parecidas. Os testes são eficazes na hora de detectar defeitos, mas não tão adequados quanto outras técnicas para atingir níveis específicos de cobertura de teste ou gerar procedimentos de teste reutilizáveis.

Quando a documentação do sistema for ruim, o tempo de teste for bem limitado ou a equipe de teste tiver um ótimo conhecimento do sistema a ser testado, o teste baseado na experiência pode ser uma boa alternativa a abordagens mais estruturadas.

O teste baseado na experiência pode ser inadequado em sistemas que exigem detalhamento na documentação de testes, altos níveis de repetibilidade ou a capacidade de avaliar a cobertura de teste com precisão.

Ao utilizar abordagens dinâmicas e heurísticas, os testadores normalmente usam testes baseados na experiência e o teste é mais reativo a eventos do que abordagens de teste pre-planejadas. Além disso, a execução e a avaliação são tarefas concomitantes.

Algumas abordagens estruturadas aos testes baseados na experiência não são totalmente dinâmicas, isto é, os testes não são totalmente criados ao mesmo tempo em que o testador executa o teste.

Repare que, embora algumas ideias sobre a cobertura mantenham relação com as técnicas ora discutidas, as técnicas baseadas na experiência não possuem critérios formais de cobertura.

3.4.1 Suposição de erros

Ao utilizar a técnica de suposição de erros, o *Test Analyst (TA)* utiliza a experiência para supor os possíveis erros que poderiam ter sido cometidos quando o código foi projetado e desenvolvido. Quando os erros esperados são identificados, o *Test Analyst (TA)* determina os melhores métodos para descobrir os defeitos resultantes.

Por exemplo, se o *Test Analyst (TA)* espera que o *software* exiba falhas quando uma senha inválida for digitada, os testes serão modelados para que vários valores diferentes sejam digitados no campo de senha para comprovar se o erro foi realmente cometido e resultou em um defeito que pode ser considerado uma falha quando os testes forem executados.

Além de ser utilizada como técnica de teste, a suposição de erros também é útil durante a análise de risco para identificar possíveis modos de falha. [Myers79]

Aplicabilidade

A suposição de erros é realizada principalmente durante o teste de integração e de sistema, mas pode ser utilizada em qualquer nível de teste. Esta técnica é frequentemente usada com outras técnicas e contribui para a ampliação da abrangência dos casos de teste existentes.

A suposição de erros também pode ser utilizada com eficácia ao testar os erros comuns de um novo *release* de *software* antes de dar início a testes mais rigorosos com *script*. As *checklists* e as taxonomias podem ser úteis na hora de orientar o teste.

Limitações / dificuldades

É difícil avaliar a cobertura e ela varia bastante dependendo da capacidade e da experiência do *Test Analyst (TA)*. É mais bem utilizada por testadores experientes que conhecem os tipos de defeitos que normalmente são introduzidos no tipo de código testado.

A suposição de erros é normalmente utilizada, mas, frequentemente, não é documentada e pode ser menos reprodutível do que outras formas de teste.

Cobertura

Quando uma taxonomia é utilizada, a cobertura é determinada pelas falhas de dados e os tipos de defeitos adequados. Sem uma taxonomia, a cobertura é limitada pela experiência e pelo conhecimento do testado e pelo tempo disponível. Os frutos desta técnica dependerão de como o testador consegue objetivar as áreas com problemas.

Tipos de defeitos

Os defeitos comuns costumam ser os definidos na taxonomia específica ou "supostos" pelo *Test Analyst (TA)* que não foram descobertos durante os testes baseados em especificações.

3.4.2 Teste baseado em checklist

Ao aplicar a técnica de teste baseado em *checklist*, o *Test Analyst (TA)* experiente utiliza uma lista generalizada de alto nível de itens a serem anotados, verificados ou lembrados ou um conjunto de regras ou critérios de acordo com o qual o produto precisa ser verificado. As *checklists* são elaboradas com base em um conjunto de normas, experiência e outras considerações. Uma *checklist* de normas de interface de usuário que serve de base para testar um aplicativo é um exemplo de teste baseado em *checklist*.

Aplicabilidade

O teste baseado em *checklist* é utilizado com maior eficácia em projetos com uma equipe de teste experiente que conhece o *software* testado ou a área coberta pela *checklist* (por exemplo, para aplicar a *checklist* da interface de usuário com sucesso, o *Test Analyst (TA)* pode conhecer o teste de interface de usuário, mas não o *software* testado em específico).

Como as *checklists* são de alto nível e tendem a não contar com etapas detalhadas encontradas normalmente em casos de teste e procedimentos de teste, o conhecimento do testador é utilizado para preencher as lacunas. Ao remover as etapas detalhadas, as *checklists* são simples e podem ser aplicadas a vários *releases* parecidos. As *checklists* podem ser utilizadas em qualquer nível de teste. As *checklists* também são utilizadas em testes de regressão e testes básicos.

Limitações / dificuldades

A natureza de alto nível das *checklists* pode afetar a reprodutibilidade dos resultados dos testes. É possível que vários testadores interpretem as *checklists* de maneira diferente e sigam abordagens diferentes para atender aos itens da *checklist*. Isto pode provocar resultados diferentes, embora a mesma *checklist* seja utilizada.

Isto pode resultar em uma cobertura maior, mas, às vezes, a reprodutibilidade é sacrificada. Além disso, as *checklists* podem resultar em um exagero de confiança a respeito do nível de cobertura alcançado, já que o

teste depende do parecer do testador. As *checklists* podem ser obtidas de casos de teste ou listas detalhadas e tendem a crescer com o tempo. A manutenção é necessária para garantir que as *checklists* cubram os aspectos importantes do *software* testado.

Cobertura

A cobertura é tão boa quanto a *checklist*, mas, por conta da natureza de alto nível da *checklist*, os resultados variarão com base no *Test Analyst (TA)*, que executa a *checklist*.

Tipos de defeitos

Os defeitos mais comuns encontrados nesta técnica incluem falhas que decorrem da variação dos dados, da sequência das etapas ou do fluxo de trabalho geral durante o teste. A utilização das *checklists* pode ajudar a manter o teste renovado já que novas combinações de dados e processos são permitidas durante o teste.

3.4.3 Teste exploratório

O teste exploratório é caracterizado pelo testador que estuda o produto e os defeitos ao mesmo tempo, planejando o teste a ser realizando, modelando e executando os testes e divulgando os resultados.

O testador ajusta dinamicamente as metas do teste durante a execução e prepara apenas documentos simples. [Whittaker09]

Aplicabilidade

Um bom teste exploratório é planejado, interativo e criativo. Exige pouca documentação sobre o sistema a ser testado e, frequentemente, é utilizado em situações em que a documentação não está disponível ou não é propícia para outras técnicas de teste. O teste exploratório é utilizado frequentemente para aprimorar outros testes e servir de base para o desenvolvimento de outros casos de teste.

Limitações / dificuldades

Pode ser difícil gerenciar e agendar os testes exploratórios. A cobertura pode ser esporádica e a reprodutibilidade é difícil. A utilização de cartas para a designação de áreas de cobertura em uma sessão de teste e a definição de um cronograma para determinar o tempo de teste são os métodos utilizados para gerenciar o teste exploratório. No final de uma sessão de teste ou um conjunto de testes, o *Test Manager (TM)* pode realizar uma reunião para fazer um balanço e coletar os resultados dos testes e determinar as cartas das próximas sessões. As sessões de balanço são difíceis de escalar para equipes de teste ou projetos grandes.

Outra dificuldade nas sessões exploratórias consiste em rastreá-las precisamente em um sistema de gestão de testes. Às vezes, isto é feito com a criação de casos de teste que, na verdade, são sessões exploratórias. Isto permite que o tempo alocado para o teste exploratório e a cobertura planejada sejam rastreados com outros testes.

Como a reprodutibilidade pode ser difícil com o teste exploratório, isto também pode causar problemas quando for necessário se lembrar das etapas para reproduzir uma falha. Algumas organizações usam a ferramenta de captura e execução de uma ferramenta de automação de testes para gravar as etapas realizadas de um testador exploratório. Isto cria um registro completo de todas as atividades durante a

sessão exploratória (ou qualquer sessão de teste baseado na experiência). Explorar os detalhes para detectar a causa real da falha pode ser tedioso, mas, pelo menos há um registro de todas as etapas envolvidas.

Cobertura

As cartas podem ser criadas para a especificação de tarefas, objetivos e entregáveis. Então, as sessões exploratórias são planejadas para atingir tais objetivos. Além disso, a carta também pode identificar onde concentrar o teste, o que está dentro e fora da abrangência da sessão de teste e quais recursos devem ser alocados para a conclusão dos testes planejados. Uma sessão pode ser utilizada para se focar em tipos específicos de defeitos e outras áreas possivelmente problemáticas que podem ser tratadas sem a formalidade do teste com *script*.

Tipos de defeitos

Os defeitos comuns encontrados nos testes exploratórios são problemas baseados em cenários que foram ignorados durante os testes funcionais com *script*, problemas que recaem entre limites funcionais e problemas relacionados ao fluxo de trabalho. Às vezes, problemas de desempenho e segurança também são descobertos durante os testes exploratórios.

3.4.4 Aplicação da melhor técnica

As técnicas baseadas em defeitos e na experiência exigem a aplicação do conhecimento dos defeitos e outras experiências de teste para objetivar o teste a fim de aumentar a detecção de defeitos. Vão de testes rápidos em que o testador não possui atividades formalmente pré-planejadas para realizar a sessões pré-planejadas, passando por sessões com *script*. Quase sempre são úteis, mas têm um valor específico nas seguintes circunstâncias:

- Nenhuma especificação está disponível;
- Há pouca documentação do sistema testado;
- O tempo para modelar e criar testes detalhados é insuficiente;
- Os testadores têm experiência no domínio e / ou na tecnologia;
- A diversidade do teste com *script* é uma meta para a maximização da cobertura de teste;
- As falhas operacionais precisam ser analisadas.

As técnicas baseadas em defeitos e na experiência também são úteis quando utilizadas junto com técnicas baseadas em especificações, já que preenchem as lacunas da cobertura do teste decorrentes de pontos fracos sistemáticos em tais técnicas. À maneira das técnicas baseadas em especificações, não há uma técnica perfeita para todas as situações. É importante que o *Test Analyst (TA)* entenda os prós e os contras de cada técnica e consiga escolher a melhor técnica ou o melhor conjunto de técnicas para determinada situação, considerando o tipo do projeto, o cronograma, o acesso à informação, a habilidade do testador e outros fatores que podem influenciar a escolha.

4. Teste de características de qualidade do software (120 min)

Palavras-chave

Teste de acessibilidade, teste de acurácia, atratividade, avaliação heurística, teste de interoperabilidade, assimilabilidade, operabilidade, teste de adequação, SUMI, entendibilidade, teste de usabilidade, WAMMI

Objetivos de aprendizagem do teste com características de qualidade do software

4.2 Características de qualidade para teste de domínio de negócios

TA-4.2.1 (K2): Explica-se com exemplos quais são as técnicas de teste adequadas para testar as características de acurácia, adequação, interoperabilidade e complacência.

TA-4.2.2 (K2): Em termos de características de acurácia, adequação e interoperabilidade, definem-se os defeitos comuns objetivados.

TA-4.2.3 (K2): Em termos de características de acurácia, adequação e interoperabilidade, define-se quando a característica deve ser testada no ciclo de vida.

TA-4.2.4 (K4): Em determinado contexto de projeto, descrevem-se as abordagens que seria adequado confirmar e validam-se tanto a implementação dos requisitos de usabilidade quanto a materialização das expectativas do usuário.

4.1 Introdução

Embora o capítulo anterior tenha descrito as técnicas específicas disponíveis para o testador, este capítulo considera a aplicação de tais técnicas na avaliação das características principais utilizadas para descrever a qualidade dos aplicativos ou sistemas do *software*.

Este *syllabus* discute as características de qualidade que podem ser avaliadas pelo *Test Analyst (TA)*. Os atributos que serão avaliados pelo *Technical Test Analyst (TTA)* são considerados no *syllabus* do *Advanced Technical Test Analyst (TTA)*. A descrição das características de qualidade do produto estipuladas na ISO 9126 serve de guia para a descrição das características.

Outras normas, como a série ISO 25000 [ISO25000] (que suplantou a ISO 9126), também podem ser úteis. As características de qualidade da ISO são divididas em características de qualidade de produtos (atributos) e cada uma delas pode ter subcaracterísticas (subatributos). Aparecem na tabela abaixo, juntamente com uma indicação de qual característica / subcaracterística é coberta pelos *syllabus* do *Test Analyst (TA)* e do *Technical Test Analyst (TTA)*.

Característica	Subcaracterísticas	*TA*	*TTA*
Funcionalidade	Acurácia, adequação, interoperabilidade, complacência	X	
Segurança			X
Confiabilidade	Maturidade (robustez), tolerância a falhas, recuperabilidade, complacência		X
Usabilidade	Entendibilidade, assimilabilidade, operabilidade, atratividade, complacência	X	
Eficiência	Desempenho (comportamento relacionado a tempo), utilização de recursos, complacência		X
Manutenibilidade	Analisabilidade, modificabilidade, estabilidade, testabilidade, complacência		X
Portabilidade	Adaptabilidade, instalabilidade, coexistência, substitutibilidade, complacência		X

O *Test Analyst (TA)* deve se concentrar nas características de funcionalidade e usabilidade na qualidade do *software*. O teste de acessibilidade deve ser realizado pelo *Test Analyst (TA)*. Embora não apareça como subcaracterística, a acessibilidade é frequentemente considerada parte do teste de usabilidade. O teste com as demais características de qualidade costuma ser considerado responsabilidade do *Technical Test Analyst (TTA)*. Embora tal alocação de trabalho possa variar em organizações diferentes, é seguida nestes *syllabus* do ISTQB.

A subcaracterística da complacência aparece em cada uma das características de qualidade. Em caso de certos ambientes de segurança crítica ou regulados, cada característica de qualidade pode ter que cumprir com normas e regulamentos específicos (por exemplo, a complacência da funcionalidade pode indicar que a funcionalidade precisa cumprir com uma norma específica, como a utilização de um protocolo de comunicação em particular, para conseguir enviar / receber dados de um *chip*).

Como as normas podem variar bastante dependendo do setor, não serão discutidas profundamente aqui. Se o *Test Analyst (TA)* estiver trabalhando em um ambiente afetado pelos requisitos de complacência, é importante compreendê-los e garantir que tanto o teste quanto a documentação do teste atendam aos requisitos de complacência.

Em relação a todas as características e subcaracterísticas de qualidade discutidas nesta seção, os riscos comuns devem ser reconhecidos para que uma estratégia de teste adequada possa ser formada e documentada.

O teste com características de qualidade exige foco específico no tempo do ciclo de vida, na disponibilidade necessária de ferramentas, *software* e documentos e na *expertise* técnica. Sem o planejamento de uma estratégia para lidar com cada característica e suas necessidades de teste únicas, o testador pode não ter tempo suficiente para o planejamento, o reforço e a execução de testes no cronograma.

Alguns destes testes, por exemplo, o teste de usabilidade, podem exigir a alocação de recursos humanos especiais, muito planejamento, laboratórios dedicados, ferramentas específicas, habilidades especializadas em testes e, na maioria dos casos, bastante tempo.

Em alguns casos, o teste de usabilidade pode ser realizado por um grupo diferente de especialistas em usabilidade ou experiência do usuário.

Os testes com características e subcaracterísticas de qualidade devem ser integrados ao cronograma geral de teste e recursos suficientes devem ser alocados aos trabalhos. Cada uma das áreas possui necessidades específicas, objetiva problemas específicos e pode acontecer em momentos diferentes durante o ciclo de vida de desenvolvimento de *software*, como se discute nas seções abaixo.

Embora o *Test Analyst (TA)* possa não ser responsável pelas características de qualidade que exigem uma abordagem mais técnica, é importante que o *Test Analyst (TA)* esteja ciente das outras características e entenda a sobreposição das áreas para o teste. Por exemplo, um produto que for reprovado no teste de desempenho provavelmente também será reprovado no teste de usabilidade se o usuário levar tempo demais para utilizá-lo com eficácia.

Igualmente, um produto com problemas de interoperabilidade em alguns componentes provavelmente não está pronto para o teste de portabilidade, já que isto tenderá a obscurecer os problemas mais básicos quando o ambiente for alterado.

4.2 Características de qualidade para teste de domínio de negócios

O teste funcional é o foco principal do *Test Analyst (TA)*. O teste funcional se concentra no que o produto faz. A base de teste dos testes funcionais geralmente consiste em um documento com requisitos ou especificações, na *expertise* referente ao domínio ou em uma necessidade implícita.

Os testes funcionais variam de acordo com o nível do teste em que são realizados e também podem ser influenciados pelo ciclo de vida de desenvolvimento de *software*. Por exemplo, um teste funcional realizado durante o teste de integração testará a funcionalidade de módulos de interface que implementam uma única função definida.

No nível do teste de sistema, os testes funcionais incluem o teste da funcionalidade do aplicativo como um todo. Em relação a sistemas de sistemas, o teste funcional se concentrará principalmente no teste integral

em sistemas integrados. Em um ambiente ágil, o teste funcional costuma ficar limitado à funcionalidade disponibilizada na iteração ou no *sprint* específico, embora o teste de regressão com uma iteração possa cobrir todas as funcionalidades lançadas.

Uma ampla variedade de técnicas de teste é empregada durante os testes funcionais (*vide* o capítulo 3). Os testes funcionais podem ser realizados por um testador dedicado, um especialista em domínios ou um desenvolvedor (normalmente no nível de componentes).

Além dos testes funcionais cobertos nesta seção, existem também duas características de qualidade que são parte da área de responsabilidade do *Test Analyst (TA)* e são consideradas áreas de teste não funcionais (com foco em *como* o produto realiza a funcionalidade). Estes dois atributos não funcionais são a usabilidade e a acessibilidade.

As seguintes características de qualidade são consideradas nesta seção:

Subcaracterísticas funcionais de qualidade

- Acurácia;
- Adequação;
- Interoperabilidade.

Características não funcionais de qualidade

- Usabilidade;
- Acessibilidade.

4.2.1 Teste de acurácia

A acurácia funcional envolve testar a adesão do aplicativo aos requisitos especificados ou implícitos e também pode incluir a acurácia computacional. O teste de acurácia emprega muitas das técnicas de teste explicadas no capítulo 3 e frequentemente utiliza as especificações ou um sistema legado como oráculo de teste. O teste de acurácia pode ser realizado em qualquer etapa do ciclo de vida e procura detectar o tratamento incorreto de dados ou situações.

4.2.2 Teste de adequação

O teste de adequação envolve a avaliação e a validação da aptidão de um conjunto de funções para as tarefas especificadas. O teste pode ser baseado em casos de uso. O teste de adequação costuma ser realizado durante o teste de sistema, mas também pode ser realizado durante as etapas posteriores do teste de integração. Os defeitos descobertos neste teste são indícios de que o sistema não conseguirá atender às necessidades do usuário de uma maneira que seja considerada aceitável.

4.2.3 Teste de interoperabilidade

O teste de interoperabilidade verifica até que ponto dois ou mais sistemas ou componentes podem trocar informações e, em seguida, utilizar as informações que foram trocadas. O teste deve cobrir todos os ambientes-alvo pretendidos (inclusive variações no *hardware*, no *software*, no *middleware*, no sistema operacional *etc.*) para garantir a troca adequada de dados. Na realidade, talvez isto apenas seja factível para

um número relativamente pequeno de ambientes. Em tal caso, o teste de interoperabilidade pode ficar limitado a um grupo representativo seleto de ambientes.

A especificação de testes de interoperabilidade exige que as combinações dos ambientes-alvo sejam combinadas, configuradas e disponibilizadas à equipe de teste. Então, os ambientes são testados com uma seleção de casos de teste funcionais que realizam os diversos pontos de troca de dados presentes no ambiente.

A interoperabilidade diz respeito a como diferentes sistemas de *software* interagem uns com os outros. O *software* com boas características de interoperabilidade pode ser integrado a uma série de outros sistemas sem a necessidade de grandes alterações. Uma série de mudanças e o esforço necessário para realizar tais alterações podem ser usados como medida de interoperabilidade.

O teste de interoperabilidade de *software* pode, por exemplo, se concentrar nas seguintes características de modelagem:

- Normas de comunicação do setor, como XML;
- A capacidade de detectar automaticamente as necessidades de comunicação dos sistemas com os quais interage e ajustá-las adequadamente.

O teste de interoperabilidade pode ser particularmente significativo para organizações que desenvolvem *software* e ferramentas comerciais de prateleira (*commercial off-the-shelf* ou COTS, na sigla em inglês) e sistemas de sistemas.

Este tipo de teste é realizado durante o teste de componente, integração e sistema com foco na interação do sistema com o ambiente. No nível da integração de sistemas, este tipo de teste é feito para determinar se o sistema totalmente desenvolvido interage bem com os outros sistemas. Como os sistemas podem interoperar em vários níveis, o *Test Analyst (TA)* deve compreender tais interações e conseguir criar as condições que realizarão as diversas interações.

Por exemplo, se os dois sistemas trocarem dados, o *Test Analyst (TA)* deve conseguir criar os dados e as transações necessários para realizar a troca de dados. Vale lembrar que talvez todas as interações não estejam claramente especificadas nos documentos de requisitos. Em vez disso, muitas destas interações serão definidas apenas nos documentos sobre a arquitetura e a modelagem do sistema.

O *Test Analyst (TA)* deve conseguir examinar tais documentos e estar preparado para isso a fim de determinar os pontos de troca de informações entre sistemas e entre o sistema e seu ambiente para garantir que todos sejam testados. Técnicas como as tabelas de decisão, os diagramas de transição de estado, os casos de uso e os testes combinatórios valem para o teste de interoperabilidade. Entre os defeitos comuns encontrados estão a troca de dados incorretos entre componentes que interagem entre si.

4.2.4 Teste de usabilidade

É importante entender por que os usuários podem ter dificuldades na hora de utilizar o sistema. Para entender isto, primeiro é necessário ter em mente que o termo usuário pode ser utilizado para indicar uma ampla variedade de tipos diferentes de pessoas, de especialistas em TI e pessoas com deficiências a crianças.

Algumas instituições nacionais (por exemplo, o Real Instituto Nacional de Cegos da Grã-Bretanha) recomenda que os *sites* sejam acessíveis para usuários com deficiências, cegos, que possuem visão parcial, com mobilidade reduzida, surdos e com deficiências cognitivas.

A verificação da usabilidade de aplicativos e *sites* por parte dos usuários supracitados também pode aprimorar a usabilidade para todos os demais. A acessibilidade é discutida abaixo.

O teste de usabilidade verifica a facilidade com a qual o usuário consegue usar ou aprender a usar o sistema para atingir uma meta específica em um contexto específico. O teste de usabilidade procura medir o seguinte:

- **Eficácia**: a capacidade que o *software* tem de permitir que o usuário atinja metas específicas com exatidão e integralidade em um contexto de uso específico;
- **Eficiência**: a capacidade que o produto tem de permitir que o usuário invista quantidades adequadas de recursos em relação à eficácia alcançada em um contexto de uso específico;
- **Satisfação**: a capacidade que o *software* tem de satisfazer o usuário em um contexto de uso específico.

Entre os atributos que podem ser medidos estão:

- **Entendibilidade**: os atributos do *software* que afetam o esforço exigido pelo usuário para reconhecer o conceito lógico e sua aplicabilidade;
- **Assimilabilidade**: os atributos do *software* que afetam o esforço exigido pelo usuário para conhecer o aplicativo;
- **Operabilidade**: os atributos do *software* que afetam o esforço exigido pelo usuário para realizar tarefas com eficácia e eficiência;
- **Atratividade**: a capacidade que o *software* tem de ser apreciado pelo usuário.

O teste de usabilidade é normalmente realizado em duas etapas:

- **Teste de usabilidade formativa**: teste realizado iterativamente durante as etapas de modelagem e prototipagem para ajudar a orientar (ou "formar") a modelagem através da identificação de defeitos de modelagem de usabilidade;
- **Teste de usabilidade cumulativo**: teste realizado após a implementação para medir a usabilidade e identificar problemas com um componente ou sistema concluído.

Entre as habilidades do testador de usabilidade devem estar a *expertise* ou o conhecimento nas seguintes áreas:

- Sociologia;
- Psicologia;
- Conformidade com normas nacionais (inclusive normas de acessibilidade);
- Ergonomia.

4.2.4.1 Realização de testes de usabilidade

A validação da implementação real deve ser realizada sob condições mais próximas possíveis daquelas em que o sistema será utilizado. Isto pode implicar a montagem de um laboratório de usabilidade com câmeras, escritórios de mentira, mesas examinadoras, usuários *etc.*, para que o pessoal de desenvolvimento consiga observar o efeito do sistema real sobre pessoas de verdade.

Frequentemente, um teste de usabilidade formal exige alguma preparação dos "usuários" (podem ser usuários reais ou representantes de usuários), que consiste em lhes dar roteiros ou instruções para que sigam. Outros testes livres permitem que o usuário explore o *software* para que os observadores possam determinar se o usuário tem facilidade ou dificuldade na hora de descobrir como realizar as tarefas.

Muitos testes de usabilidade podem ser executados pelo *Test Analyst (TA)* como parte de outros testes, por exemplo, durante um teste de sistema funcional. Para uma abordagem coerente à detecção e à notificação de defeitos de usabilidade em todas as etapas do ciclo de vida, as diretrizes de usabilidade podem ser úteis. Sem diretrizes de usabilidade, talvez seja difícil determinar o que uma usabilidade "inaceitável". Por exemplo, se o usuário precisar de 10 cliques para acessar um aplicativo, isso é admissível?

Sem diretrizes específicas, o *Test Analyst (TA)* pode se encontrar na posição incômoda de defender relatórios de defeito que o desenvolvedor deseja encerrar porque o *software* funciona "como foi projetado". É muito importante ter as especificações de usabilidade verificáveis definidas nos requisitos e um conjunto de diretrizes de usabilidade aplicado a todos os projetos parecidos.

As diretrizes devem incluir itens como a acessibilidade das instruções, a clareza dos *prompts*, o número de cliques para realizar uma atividade, a notificação de erros, os indicadores de processamento (algum tipo de indício para o usuário de que o sistema está realizando o processamento e não consegue aceitar outras entradas no momento), diretrizes de *layout* da tela, o uso das cores e dos sons e outros fatores que afetam a experiência do usuário.

4.2.4.2 Especificações do teste de usabilidade

Entre as técnicas principais do teste de usabilidade estão:

- Inspeção, avaliação ou revisão;
- Interação dinâmica com protótipos;
- Verificação e validação da implementação real;
- Realização de pesquisas e questionários.

Inspeção, avaliação ou revisão

A inspeção ou a revisão dos requisitos, das especificações e das modelagens, do ponto de vista da usabilidade, que aumentam o nível de envolvimento do usuário podem ser rentáveis se os problemas forem detectados antecipadamente.

A avaliação heurística (a inspeção sistemática da modelagem da usabilidade de uma interface de usuário) pode ser utilizada para detectar problemas de usabilidade na modelagem para que possam ser dirimidos em um processo de modelagem iterativa. Isto implica fazer um pequeno número de avaliadores examinar a interface e julgar sua complacência com os princípios aceitos de usabilidade (a "heurística").

As avaliações são mais eficazes quando a interface de usuário é mais visível. Por exemplo, capturas de tela são mais fáceis de entender e interpretar do que um memorial descritivo da funcionalidade fornecido por uma tela específica. A visualização é importante para a devida avaliação de usabilidade da documentação.

Interação dinâmica com protótipos

Quando os protótipos forem desenvolvidos, o *Test Analyst (TA)* deve trabalhar com os protótipos e ajudar os desenvolvedores a aprimorá-los mediante a incorporação do *feedback* dos usuários à modelagem. Assim,

os protótipos podem ser refinados e o usuário consegue ver, de maneira mais realista, o *look and feel* do produto acabado.

Verificação e validação da implementação real

Quando os requisitos especificarem as características de usabilidade do *software* (por exemplo, o número de cliques para atingir uma meta específica), devem ser criados casos de teste para verificar se a implementação do *software* inclui tais características.

Para a validação da implementação real, os testes especificados para o teste de sistema funcional podem ser desenvolvidos como cenários de teste de usabilidade. Os cenários de teso medem características específicas de usabilidade, como a assimilabilidade ou a operabilidade, em vez de consequências funcionais.

Os cenários de teste de usabilidade podem ser desenvolvidos para testar a sintaxe e a semântica especificamente. A sintaxe é a estrutura ou a gramática da interface (por exemplo, o que pode ser digitado em um campo de entrada), enquanto a semântica descreve o significado e o propósito (por exemplo, mensagens e saídas razoáveis e significativas do sistema para o usuário) da interface.

Técnicas caixa-preta (por exemplo, as descritas na seção 3.2), particularmente os casos de uso que podem ser definidos em texto simples ou com *Unified Modeling Language* (UML, na sigla em inglês), às vezes são empregadas nos testes de usabilidade.

Os cenários de teste dos testes de usabilidade também precisam incluir as instruções de usuário, a alocação de tempo para antes e depois das entrevistas de teste para dar instruções e receber *feedback* e um protocolo acordado para a realização das sessões. Este protocolo inclui uma descrição de como o teste será realizado, as oportunidades, os apontamentos, os registros das sessões e os métodos de entrevista e pesquisa que serão utilizados.

Realização de pesquisas e questionários

Técnicas de pesquisa e questionário podem ser aplicados para coletar observações e *feedback* relacionados ao comportamento do usuário com o sistema. Pesquisas padronizadas publicamente disponíveis, como o *Software Usability Measurement Inventory* (SUMI) e o *Website Analysis and MeasureMent Inventory* (WAMMI), permitem a criação de *benchmarks* contra um banco de dados de medições de usabilidade anteriores. Além disso, como o SUMI faz medições concretas da usabilidade, isto pode fornecer um conjunto de critérios de conclusão / aceitação.

4.2.5 Teste de acessibilidade

É importante levar em consideração a acessibilidade ao *software* para as pessoas que possuem necessidades específicas ou restrições de uso. Isto inclui as pessoas com deficiência. O teste de acessibilidade deve considerar as normas, como as Diretrizes de Acessibilidade ao Conteúdo na Web, e as leis pertinentes, como as Leis sobre a Discriminação contra as Pessoas com Deficiência (Reino Unido e Austrália) e a Seção 508 (Estados Unidos).

A acessibilidade, à maneira da usabilidade, deve ser levada em consideração durante as fases de modelagem. Frequentemente, o teste ocorre durante os níveis de integração e continua ao longo do teste de sistema até os níveis de teste de aceitação. Normalmente, os defeitos são determinados quando o *software* deixa de atender aos regulamentos designados ou às normas definidas para o *software*.

5. Revisões (165 min)

Palavras-chave

nenhuma

Objetivos de aprendizagem das revisões

5.1 Introdução

TA-5.1.1 (K2): Explica-se por que a preparação da revisão é importante para o *Test Analyst (TA)*.

5.2 Utilização de checklists em revisões

TA-5.2.1 (K4): Analisa-se um caso de uso ou uma interface de usuário e identificam-se problemas de acordo com as informações da *checklist* fornecidas no *syllabus*.

TA-5.2.2 (K4): Analisam-se especificações de requisitos ou estórias de usuário e identificam-se problemas de acordo com as informações da *checklist* fornecidas no *syllabus*.

5.1 Introdução

Um processo de revisão bem-sucedido exige planejamento, participação e acompanhamento. O *Test Analyst (TA)* deve participar ativamente do processo de revisão e oferecer seus pontos de vista únicos. O *Test Analyst (TA)* deve contar com treinamento formal em revisão para compreender melhor suas funções respectivas em qualquer processo de revisão; Todos os participantes de revisões devem estar comprometidos com os benefícios de uma revisão bem feita. Quando bem-feitas, as revisões podem ser a maior e mais rentável contribuição para a qualidade geral proporcionada.

Independentemente do tipo de revisão realizada, o *Test Analyst (TA)* deve reservar tempo o suficiente para os preparativos. Isto inclui tempo para revisar o produto de trabalho, verificar os documentos cruzados para comprovar sua coerência e determinar o que pode estar faltando no produto de trabalho. Sem tempo suficiente para os preparativos, o *Test Analyst (TA)* pode ficar restrito apenas à edição do que já aparece no documento em vez de participar de uma revisão eficiente que maximiza o uso do tempo da equipe de revisão e proporciona o melhor *feedback* possível.

Uma boa revisão inclui a compreensão do que está escrito, a determinação do que está faltando e a comprovação de que o produto descrito mantém a coerência com outros produtos que ou já foram desenvolvidos ou estão em desenvolvimento. Por exemplo, ao revisar um plano de teste de integração, o *Test Analyst (TA)* também deve levar em consideração os itens que foram integrados. Quais são as condições necessárias para que estejam prontos para a integração?

Existem dependências que precisam ser documentadas? Existem dados disponíveis para testar os pontos da integração? A revisão não se limita ao produto de trabalho revisado. Também deve levar em consideração a interação daquele item com os outros no sistema.

O autor de um produto revisado pode facilmente se sentir criticado. O *Test Analyst (TA)* deve fazer de tudo para abordar quaisquer comentários sobre a revisão partindo da premissa de colaboração com o autor para a criação do melhor produto possível.

Ao usar esta abordagem, os comentários serão feitos de maneira construtiva e ficarão voltados para o produto de trabalho e não para o autor. Por exemplo, se uma declaração for ambígua, é melhor afirmar “Não sei o que devo testar para comprovar se este requisito foi implementado corretamente. Pode me ajudar a entender isto?” do que “Este requisito é ambíguo e ninguém conseguirá decifrá-lo”.

Em uma revisão, o trabalho do *Test Analyst (TA)* consiste em garantir que as informações fornecidas no produto de trabalho sejam suficientes para fundamentar o teste. Se as informações inexistirem, não forem claras ou não fornecerem o nível necessário de detalhamento, então, provavelmente, isto é um defeito que precisa ser corrigido pelo autor. Ao manter uma abordagem positiva em vez de uma abordagem crítica, os comentários serão mais bem recebidos e a reunião será mais produtiva.

5.2 Utilização de checklists em revisões

As *checklists* são usadas durante as revisões para lembrar os participantes de que devem verificar pontos específicos durante a revisão. Além disso, as *checklists* podem ajudar a despersonalizar a revisão, por exemplo: “É a mesma *checklist* que usamos em todas as revisões e não só em seu produto de trabalho”. As *checklists* podem ser genéricas e são utilizadas em todas as revisões ou podem se focar em características e

áreas de qualidade ou tipos de documentos específicos. Por exemplo, uma *checklist* genérica pode verificar as propriedades gerais dos documentos, como ter um identificador único, sem referências a definir, uma formatação adequada e itens de conformidade parecidos.

Uma *checklist* específica referente a um documento com requisitos pode conter verificações do uso adequado dos termos deverá e deve, da testabilidade de cada requisito declarado e assim por diante. O formato dos requisitos também pode indicar o tipo de *checklist* a ser utilizado. Um documento de requisitos que assumir um formato de texto narrativo terá critérios de revisão diferentes dos de um documento baseado em diagramas.

As *checklists* também podem estar orientadas para o conjunto de habilidades de um programador / arquiteto ou testador. No caso do *Test Analyst (TA)*, a *checklist* do conjunto de habilidades do testador seria a mais adequada. As *checklists* podem incluir os itens abaixo.

As *checklists* utilizadas em requisitos, casos de uso e estórias de usuários geralmente têm um foco diferente do das utilizadas em códigos ou arquiteturas. As *checklists* orientadas para requisitos podem incluir os seguintes itens:

- A testabilidade de cada requisito;
- Os critérios de aceitação de cada requisito;
- A disponibilidade de uma estrutura de casos de uso, onde couber;
- A identificação única de cada requisito / caso de uso / estória de usuário;
- O versionamento de cada requisito / caso de uso / estória de usuário;
- A rastreabilidade de cada requisito a partir de requisitos de negócios / de *marketing*;
- A rastreabilidade entre requisitos e casos de uso.

Isto deve servir apenas de exemplo. Vale lembrar que, se um requisito não for testável, o que significa que foi definido de maneira que o *Test Analyst (TA)* não consegue nem determinar como testá-lo, há um defeito nele. Por exemplo, um requisito que declarar “O *software* deve ser muito fácil de usar” não é testável. Como o *Test Analyst (TA)* pode determinar se o *software* é fácil de usar ou até mesmo muito fácil de usar?

Se, pelo contrário, o requisito disser “O *software* precisa cumprir com as normas de usabilidade declaradas no documento de normas de usabilidade” e se o documento de normas de usabilidade realmente existir, eis um requisito testável. Além disso, trata-se de um requisito principal porque corresponde a todos os itens da interface. Neste caso, este requisito pode facilmente gerar muitos casos de teste individuais em um aplicativo não trivial.

A rastreabilidade a partir deste requisito ou talvez a partir do documento de normas de usabilidade até os casos de teste também é crucial porque se a especificação de usabilidade mencionada mudar, todos os casos de teste precisarão ser revisados e atualizados quando necessário.

Um requisito também não é testável se o testador não conseguir determinar se o teste o aprovou ou o reprovou ou se não conseguir elaborar um teste que consiga aprová-lo ou reprová-lo. Por exemplo, não é possível testar um requisito que declare “O sistema deverá ficar disponível o tempo todo, 24 horas por dia, sete dias por semana, 365 (ou 366) dias por ano”. Uma *checklist* simples para revisões de casos de uso pode incluir as seguintes perguntas:

- O caminho (cenário) principal foi claramente definido?
- Todos os caminhos (cenários) alternativos foram identificados, inclusive o tratamento de erros?

- As mensagens da interface de usuário foram definidas?
- Existe apenas um caminho principal (deve existir, senão existem vários casos de uso)?
- Todos os caminhos são testáveis?

Uma *checklist* simples referente à usabilidade da interface de usuário de um aplicativo pode incluir o seguinte:

- Todos os campos e suas funções foram definidos?
- Todas as mensagens de erro foram definidas?
- Todos os *prompts* de usuário foram definidos e são coerentes?
- A ordem de abas dos campos foi definida?
- Existem alternativas no teclado para as ações do *mouse*?
- Existem combinações de teclas de atalho definidas para o usuário (por exemplo, copiar e colar)?
- Existem dependências entre os campos (como certa data precisa ser posterior a outra data)?
- Há um *layout* de tela?
- O *layout* da tela atende aos requisitos especificados?
- Há um indicador para o usuário que aparece quando o sistema está processando dados?
- A tela atende ao requisito mínimo de um clique (se definido)?
- A navegação flui logicamente para o usuário com base nas informações de caso de uso?
- A tela atende aos requisitos de assimilabilidade?
- Há um texto de ajuda disponível para o usuário?
- Há um texto disponível para o usuário quando o cursor do *mouse* é colocado sobre um item?
- O usuário considerará isto “atrativo” (avaliação subjetiva)?
- O uso das cores combina com outros aplicativos e com as normas da organização?
- Os efeitos sonoros são utilizados adequadamente e são configuráveis?
- A tela atende aos requisitos de localização?
- O usuário consegue saber o que fazer (entendibilidade) (avaliação subjetiva)?
- O usuário conseguirá se lembrar do que fazer (entendibilidade) (avaliação subjetiva)?

Em um projeto ágil, os requisitos costumam assumir a forma de estórias de usuário. Estas estórias representam pequenas unidades de funcionalidade demonstrável. Se, por um lado, um caso de uso é uma transação de usuário que se estende a várias áreas de funcionalidade, uma estória de usuário é mais isolada e geralmente entra em determinada abrangência quando é elaborada. Uma *checklist* referente a uma estória de usuário pode incluir:

- A estória é adequada para a iteração / o *sprint* almejado?
- Os critérios de aceitação são definidos e testáveis?
- A funcionalidade foi claramente definida?
- Existem dependências entre esta estória e as outras?
- A estória foi priorizada?
- A estória contém um item de funcionalidade?

Claro, se a estória definir uma nova interface, a utilização de uma *checklist* genérica referente a estórias (como a supracitada) e uma *checklist* detalhada referente a interfaces de usuário seria adequada.

Uma *checklist* pode ser ajustada com base no seguinte:

- Organização (por exemplo, pensar nas políticas, nas normas e nas convenções da empresa);
- Projeto / trabalhos de desenvolvimento (por exemplo, foco, normas técnicas, riscos);
- Objeto revisado (por exemplo, revisões de códigos podem ser ajustadas a linguagens de programação específicas).

Checklists boas encontrarão problemas e ajudarão a iniciar discussões sobre outros itens que poderiam não ter sido mencionados especificamente nelas. A utilização de uma combinação de *checklists* é uma ótima maneira de garantir que uma revisão realize o produto de trabalho da mais alta qualidade. A utilização de *checklists* padrão, como as mencionadas no *syllabus* do nível fundamental, e o desenvolvimento de *checklists* organizacionalmente específicas, como as supracitadas, ajudarão o *Test Analyst (TA)* a ser eficaz nas revisões.

Para obter mais informações sobre revisões e inspeções, *vide* [Gilb93] e [Wiegers03].

6. Gestão de defeitos (120 min)

Palavras-chave

Taxonomia de defeitos, contenção de fase, análise de causa-raiz

Objetivos de aprendizagem da gestão de defeitos

6.2 Quando é possível detectar um defeito?

TA-6.2.1 (K2): Explica-se como a contenção de fase pode reduzir custos.

6.3 Campos dos relatórios de defeito

TA-6.3.1 (K2): Explicam-se as informações que podem ser necessárias ao documentar um defeito não funcional.

6.4 Classificação de defeitos

TA-6.4.1 (K4): Identificam-se, coletam-se e registram-se as informações de classificação de determinado defeito.

6.5 Análise de causa-raiz

TA-6.5.1 (K2): Explica-se o propósito da análise de causa-raiz.

6.1 Introdução

O *Test Analyst (TA)* avalia o comportamento do sistema em termos de necessidades do negócio e do usuário, por exemplo, o usuário saberia o que fazer quando se deparasse com esta mensagem ou este comportamento? Com a comparação dos resultados reais com os esperados, o *Test Analyst (TA)* determina se o sistema está se comportando corretamente. Uma anomalia (também chamada de incidente) é uma ocorrência inesperada que exige uma maior investigação. Uma anomalia pode ser uma falha causada por um defeito. Uma anomalia pode levar à elaboração de um relatório de defeito. Um efeito é todo problema real que precise ser resolvido.

6.2 Quando é possível detectar um defeito?

Um defeito pode ser detectado através de testes estáticos e os sintomas do defeito, a falha, podem ser detectados através de testes dinâmicos. Cada fase do ciclo de vida de desenvolvimento do *software* deve contar com métodos para a detecção e a eliminação de possíveis falhas. Por exemplo, durante a fase de desenvolvimento, as revisões de códigos e modelagens devem ser utilizadas para detectar defeitos. Durante o teste dinâmico, os casos de teste são utilizados para detectar falhas.

Quanto antes o defeito for detectado e corrigido, menores serão os custos de qualidade para o sistema como um todo. Por exemplo, o teste estático consegue encontrar defeitos antes que o teste dinâmico seja possível. É um dos motivos pelo qual o teste estático é uma abordagem rentável para a produção de *software* de alta qualidade.

O sistema de rastreamento de defeitos permite que o *Test Analyst (TA)* registre a fase do ciclo de vida na qual o defeito foi introduzido e a fase em que foi detectado. Se as duas fases forem a mesma, a contenção de fase perfeita foi alcançada. Isto significa que o defeito foi introduzido e detectado na mesma fase e não "fugiu" para uma fase posterior. Um exemplo disto seria um requisito incorreto identificado durante a revisão de requisitos e corrigido durante ela.

Não se trata apenas do uso eficiente da revisão de requisitos, mas também o defeito é impedido de provocar trabalho adicional, o que o encareceria para a organização. Se um requisito incorreto "fugir" da revisão de requisitos e for implementado depois pelo desenvolvedor, testado pelo *Test Analyst (TA)* e detectado pelo usuário durante o teste de aceitação do usuário, todo o trabalho realizado em tal requisito foi um desperdício de tempo (sem falar que o usuário pode ter perdido a confiança no sistema).

A contenção de fase é uma forma eficaz de reduzir os custos dos defeitos.

6.3 Campos dos relatórios de defeito

Os campos (parâmetros) em um relatório de defeito pretendem fornecer informações suficientes para que o relatório de defeito sirva de base para a tomada de providências. Um relatório de defeito que provoca uma ação é:

- **Completo**: todas as informações necessárias aparecem no relatório;
- **Conciso**: não existem informações irrelevantes no relatório;

- **Exato**: as informações no relatório são corretas e declaram claramente os resultados esperados e reais e as etapas adequadas que devem ser reproduzidas;
- **Objetivo**: o relatório é uma declaração de fatos escrita profissionalmente.

As informações registradas em um relatório de defeito devem ser divididas em campos de dados. Quanto mais bem definidos forem os campos, mais fácil é divulgar defeitos individuais e produzir relatórios de tendências e outros resumos.

Quando um número definido de opções estiver disponível para um campo, ter listas suspensas dos valores disponíveis pode diminuir o tempo necessário ao registrar um defeito. As listas suspensas são apenas eficazes quando o número de opções é limitado e o usuário não precisa repassar uma longa lista para encontrar a opção correta.

Tipos diferentes de relatórios de defeito exigem informações diferentes e a ferramenta de gestão de defeitos deve ser flexível o bastante para gerar *prompts* para os campos adequados dependendo do tipo de defeito.

Os dados devem ser registrados em campos diferentes, idealmente suportados pelas validações de dados para evitar falhas na entrada de dados e garantir uma divulgação eficaz.

Os relatórios de defeito tratam de falhas descobertas durante os testes funcionais e não funcionais. As informações em um relatório de defeito sempre devem estar orientadas para a identificação clara do cenário em que o problema foi detectado, inclusive as etapas e os dados necessários para a reprodução do cenário, e os resultados reais e esperados.

Os relatórios de defeito não funcionais podem exigir mais detalhes do ambiente, outros parâmetros de desempenho (por exemplo, o tamanho da carga), a sequência de etapas e os resultados esperados. Ao documentar uma característica de usabilidade, é importante declarar o que o usuário esperava que o *software* fizesse. Por exemplo, se a norma de usabilidade disser que uma operação deve ser concluída em menos de quatro cliques, o relatório de defeito deve mostrar quantos cliques foram necessários e comparar esse número com a norma declarada.

Quando a norma não estiver disponível e os requisitos não cobrirem os aspectos de qualidade não funcionais do *software*, o testador pode utilizar o teste de pessoa razoável para determinar se a usabilidade é inaceitável. Nesse caso, as expectativas de tal “pessoa razoável” devem estar claramente contidas no relatório de defeito. Como os requisitos não funcionais às vezes não aparecem nos documentos de requisitos, a documentação de falhas não funcionais apresenta mais desafios para o testador na documentação do comportamento “esperado” do que do comportamento “real”.

Embora, normalmente, a meta seja a redigir um relatório de defeito para consertar o problema, as informações sobre os defeitos também devem ser fornecidas para reforçar uma classificação precisa, uma análise de risco e uma melhoria de processo.

6.4 Classificação de defeitos

Existem vários níveis de classificação que um relatório de defeito pode receber ao longo de seu ciclo de vida. A classificação adequada de defeitos é parte integrante da notificação adequada de defeitos. As classificações são utilizadas para agrupar defeitos, avaliar a eficácia do teste, avaliar a eficácia do ciclo de vida de desenvolvimento de terminar tendências interessantes.

Entre as informações sobre as classificações comuns de defeitos recém-identificados estão:

- A atividade do projeto que resultou na detecção do defeito (por exemplo, revisão, auditoria, inspeção, codificação, teste);
- A fase do projeto em que o defeito foi introduzido (se conhecido) (por exemplo, requisitos, modelagem, modelagem detalhada, código);
- A fase do projeto em que o defeito foi detectado (por exemplo, requisitos, modelagem, modelagem detalhada, código, revisão de código, teste de unidade, teste de integração, teste de sistema, teste de aceitação);
- Suposta causa do defeito (por exemplo, requisitos, modelagem, interface, código, dados);
- Repetibilidade (por exemplo, uma vez, intermitente, reprodutível);
- Sintoma (por exemplo, pane, interrupção, erro na interface de usuário, erro no sistema, desempenho).

Assim que o defeito for investigado, outras classificações podem ser possíveis:

- **Causa-raiz**: o erro cometido que levou ao defeito, por exemplo, processo, erro de codificação, erro de usuário, erro de teste, problema de configuração, problema de dados, *software* de terceiros, problema de *software* externo, problema de documentação;
- **Fonte**: o produto de trabalho no qual o erro foi cometido (por exemplo, requisitos, modelagem, modelagem detalhada, arquitetura, modelagem de banco de dados, documentação do usuário, documentação do teste);
- **Tipo** (por exemplo, problema de lógica, problema computacional, problema de tempo, tratamento de dados, aprimoramento).

Quando o defeito for consertado (ou tiver sido delegado ou tiver sido reprovado na confirmação), ainda mais informações de classificação podem estar disponíveis, como:

- **Solução** (por exemplo, alteração no código, alteração na documentação, delegação, inexistência de problema, duplicação);
- **Ação corretiva** (por exemplo, revisão de requisitos, revisão do código, teste de unidade, documentação de configuração, preparação de dados, ausência de alteração).

Além das categorias de classificação, os defeitos também são frequentemente classificados com base na gravidade e na prioridade. Além disso, dependendo do projeto, talvez faça sentido classificá-los com base no impacto sobre a segurança da missão, no impacto sobre o cronograma do projeto, o projeto, nos custos do projeto, no risco do projeto e no impacto sobre a qualidade do projeto. Estas classificações podem ser levadas em consideração em contratos que estipulem a rapidez de execução de um reparo.

A área final de classificação é a resolução final. Frequentemente, os defeitos são agrupados com base na resolução, por exemplo, reparado / verificado, fechado / sem problema, delegado, em aberto / não resolvido. Esta classificação costuma ser utilizada ao longo de um projeto, já que os defeitos são rastreados ao longo do ciclo de vida.

Os valores de classificação utilizados por uma organização são frequentemente customizados. Os valores supracitados são apenas exemplos de alguns dos valores comuns utilizados no setor. É importante que os valores da classificação sejam utilizados de maneira coerente para que sejam úteis. Campos de classificação em excesso deixarão a abertura e o processamento de defeitos um pouco lentos, então é importante cruzar

o valor dos dados coletados com o custo incremental de cada defeito processado. A capacidade de personalizar os valores de classificação coletados por uma ferramenta é frequentemente um fator importante na seleção de ferramentas.

6.5 Análise de causa-raiz

A finalidade da análise de causa-raiz consiste em determinar o que provocou o defeito e fornecer dados para reforçar alterações nos processos que removerão as causas-raiz responsáveis por boa parte dos defeitos. A análise de causa-raiz costuma ser realizada pela pessoa que investiga e ou corrige o problema ou determina que o problema não deve ou não pode ser corrigido. Normalmente, é o desenvolvedor.

A definição de um valor preliminar de causa-raiz é normalmente realizada pelo *Test Analyst (TA)*, que, com base em seus conhecimentos, suporá o que provavelmente ocasionou o problema. Ao confirmar o reparo, o *Test Analyst (TA)* verificará a configuração de causa-raiz feita pelo desenvolvedor. Quando a causa-raiz for determinada, também é comum determinar ou confirmar a fase em que o defeito foi introduzido.

Entre as causas-raiz comuns estão:

- Requisito confuso;
- Requisito ausente;
- Requisito errado;
- Implementação de modelagem incorreta;
- Implementação de interface incorreta;
- Erro de lógica de código;
- Erro de cálculo;
- Erro de *hardware*;
- Erro de interface;
- Dado inválido.

As informações sobre a causa-raiz são agregadas para determinar problemas comuns que resultam na criação de defeitos. Por exemplo, se um grande número de defeitos for causado por requisitos confusos, faz sentido se empenhar mais para realizar avaliações de requisitos eficazes. Igualmente, se a implementação de interfaces for um problema em grupos de desenvolvimento, sessões conjuntas de modelagem podem ser necessárias.

A utilização de causas-raiz para o aprimoramento de processos ajuda a organização a monitorar as vantagens de mudanças eficazes em processos e a quantificar os custos dos defeitos que podem ser atribuídos a uma causa-raiz específica. Isto pode ajudar a financiar mudanças em processos que podem exigir a aquisição de ferramentas e equipamentos adicionais e a alteração do cronograma. O *syllabus Melhoramento do processo de teste* do nível especialista do ISTQB [ISTQB_EL_ITP] examina a análise de causa-raiz com maior profundidade.

7. Ferramentas de teste (45 min)

Palavras-chave

Teste orientado para palavras-chaves, ferramenta de preparação de dados de teste, ferramenta de modelagem de teste, ferramenta de execução de teste

Objetivos de aprendizagem das ferramentas de teste

7.2 Ferramentas e automação de testes

TA-7.2.1 (K2): Explicam-se as vantagens da utilização de ferramentas de preparação de dados de teste, de modelagem de teste e de execução de teste.

TA-7.2.2 (K2): Explica-se a função do *Test Analyst (TA)* na automação orientada para palavras-chave.

TA-7.2.3 (K2): Explicam-se as etapas para a solução de uma falha de execução em um teste automatizado.

7.1 Introdução

As ferramentas de teste podem melhorar bastante a eficiência e a exatidão dos trabalhos de teste, mas apenas se as ferramentas adequadas forem devidamente implementadas. As ferramentas de teste têm que ser gerenciadas como outro aspecto de uma organização de teste bem administrada. A sofisticação e a aplicabilidade das ferramentas de teste variam bastante e o mercado de ferramentas muda constantemente. Normalmente, as ferramentas podem ser obtidas com fornecedores de ferramentas comerciais e diversos *sites* de ferramentas de *freeware* ou *shareware*.

7.2 Ferramentas e automação de testes

Boa parte do trabalho do *Test Analyst (TA)* exige o uso eficaz de ferramentas. Saber quais ferramentas devem ser utilizadas e quando elas devem ser utilizadas pode aumentar a eficiência do *Test Analyst (TA)* e ajudar a estipular a melhor cobertura de teste no tempo alocado.

7.2.1 Ferramentas de modelagem de testes

As ferramentas de modelagem de testes são utilizadas para a criação de casos de teste e dados de teste que serão aplicados no teste. As ferramentas podem funcionar a partir de formatos específicos de documentos de requisitos, modelos (por exemplo, UML) ou entradas fornecidas pelo *Test Analyst (TA)*. As ferramentas de modelagem de testes são frequentemente projetadas e desenvolvidas para funcionar com formatos e produtos específicos, como ferramentas específicas de gestão de requisitos.

As ferramentas de modelagem de testes podem fornecer informações para que o *Test Analyst (TA)* as utilize na hora de determinar os tipos de testes necessários para a obtenção do nível desejado de cobertura de teste, confiança no sistema ou mitigação de riscos de produto. Por exemplo, as ferramentas de classificação por árvore geram (e exigem) o conjunto de combinações necessário para alcançar a cobertura total com base em um critério de cobertura escolhido. Então, estas informações podem ser utilizadas pelo *Test Analyst (TA)* para determinar os casos de teste que precisam ser executados.

7.2.2 Ferramentas de preparação de dados de testes

As ferramentas de preparação de dados de testes dispõem de vários benefícios. Algumas ferramentas de preparação de dados de testes conseguem analisar documentos como um documento de requisitos ou até mesmo o código-fonte para determinar os dados necessários durante o teste para alcançar um nível de cobertura.

Outras ferramentas de preparação de dados de teste podem obter um conjunto de dados de um sistema de produção e “depurá-lo” ou “anonimizá-lo” para remover quaisquer informações pessoais ao mesmo tempo em que mantém a integridade interna dos dados.

Então, os dados depurados podem ser utilizados para a realização de testes sem o risco de uma falha na segurança ou o uso indevido de informações pessoais. Isto é particularmente importante quando houver necessidades de grandes volumes de dados realistas. Outras ferramentas de geração de dados podem ser utilizadas para gerarem dados de testes de certos conjuntos de parâmetros de entrada (isto é, a utilização

em testes aleatórios). Algumas delas analisarão a estrutura do banco de dados para determinas as entradas necessárias do *Test Analyst (TA)*.

Ferramentas de execução de testes automatizados

As ferramentas de execução de testes são mais utilizadas por *Test Analyst (TA)s* em todos os níveis de teste para a realização de testes e a verificação do resultado dos testes. A finalidade de usar uma ferramenta de execução de testes normalmente consiste em um ou mais dos seguintes:

- Reduzir custos (em termos de esforço e / ou tempo);
- Executar mais testes;
- Executar o mesmo teste em muitos ambientes;
- Tornar a execução de testes mais repetível.
- Executar testes que seria impossível executar manualmente (isto é, testes de validação de muitos dados).

Frequentemente, os objetivos se sobrepõem aos objetivos principais de aumentar a cobertura ao mesmo tempo em que os custos são reduzidos.

7.2.2.1 Aplicabilidade

O retorno do investimento das ferramentas de execução de testes costuma ser mais alto quando os testes de regressão são automatizados por conta do baixo nível de manutenção esperada e a repetição da execução dos testes. A automação de testes básicos também pode ser um uso eficaz da automação devido ao uso frequente dos testes, à necessidade de um resultado rápido e, embora o custo da manutenção possa ser mais alto, à capacidade de ter uma forma automatizada de avaliar um novo pacote em um ambiente de integração contínua.

As ferramentas de execução de testes são normalmente utilizadas durante os níveis de teste de sistema e de integração. Algumas ferramentas, especialmente as ferramentas de teste API, podem ser utilizadas no nível do teste de componente também. O aproveitamento das ferramentas quando elas forem relevantes contribuirá para a melhoria do retorno sobre o investimento.

7.2.2.2 Fundamentos das ferramentas de automação de testes

As ferramentas de execução de testes funcionam com a execução de um conjunto de instruções redigidas em uma linguagem de programação, frequentemente chamada de linguagem *script*. As instruções para a ferramenta são de um nível muito detalhado que especifica entradas, ordem da entrada, valores específicos utilizados nas entradas e saídas esperadas. Isto pode deixar *scripts* detalhados suscetíveis a alterações no *software* em teste (SUT, na sigla em inglês), particularmente quando a ferramenta interage com a interface gráfica de usuário (GUI, na sigla em inglês).

A maioria das ferramentas de execução de testes vem com um comparador, que permite a comparação de um resultado real com um resultado esperado armazenado.

7.2.2.3 Implementação da automação de testes

A tendência na automação de execução de testes (em programação) é a de ir de instruções detalhadas de baixo nível a linguagens de nível mais alto com a utilização de bibliotecas, macros e subprogramas. As técnicas de modelagem, como a captura orientada para palavras-chave e comandos, captam uma série de instruções e mencionam as que têm uma palavra-chave ou um comando específico. Isto permite que o *Test*

Analyst (TA) escreva casos de teste em um idioma humano e desconsidere a linguagem de programação subjacente e as funções de nível menor.

A utilização desta técnica de redação modular permite uma manutenibilidade mais fácil durante mudanças na funcionalidade e na interface do *software* em teste. [Bath08] O uso de palavras-chave em *scripts* automatizados é discutido abaixo.

Os modelos podem ser utilizados para guiarem a criação de palavras-chave ou comandos. Ao examinar os modelos de processos de negócios, que frequentemente são incluídos nos documentos de requisitos, o *Test Analyst (TA)* consegue determinar os principais processos de negócios que precisam ser testados. Então, as etapas dos processos podem ser determinadas, inclusive os pontos de decisão que podem ocorrer durante os processos.

Os pontos de decisão podem virar comandos que a automação de testes consegue obter e utilizar a partir de planilhas com palavras-chave ou comandos. A modelagem de processos de negócios é um método de documentação de processos de negócios e pode ser utilizada para identificar os principais processos e pontos de decisão. A modelagem pode ser realizada manualmente ou com ferramentas que se fundamentarão em entradas com base em regras de negócios e descrições de processos.

7.2.2.4 Melhoria do sucesso dos trabalhos de automação

Ao determinar os testes que precisam ser automatizados, todo caso de teste apto ou suíte de testes aptos deve ser avaliado para ver se merece ser automatizado. Muitos projetos fracassados de automação se baseiam na automação de casos de teste manuais prontos e não verificam se há alguma vantagem real na automação. Pode ser ideal para certo conjunto de casos de teste (uma suíte) para conter testes manuais, semi-automatizados e totalmente automatizados.

Os seguintes aspectos devem ser levados em consideração ao implementar um projeto de automação de execução de testes:

Possíveis vantagens:

- O tempo de execução de testes automatizados ficará mais previsível;
- O teste de regressão e a validação de defeitos com testes automatizados serão mais rápidos e mais confiáveis no final do projeto;
- A situação e o crescimento técnico do testador ou da equipe de teste precisam ser melhorados com a utilização de ferramentas automatizadas;
- A automação pode ser particularmente útil com ciclos de vida de desenvolvimento iterativos e incrementais para a realizar de testes de regressão melhores para cada pacote ou iteração;
- A cobertura de certos tipos de teste só pode ser possível com ferramentas automatizadas (por exemplo, validação de muitos dados);
- A automação na execução de testes pode ser mais rentável do que o teste manual referente a testes de entrada, conversão e comparação de muitos dados ao proporcionar entradas e verificações rápidas e coerentes.

Possíveis riscos:

- Um teste manual incompleto, ineficaz ou incorreto pode ser automatizado no estado em que se encontra (*as is*);

- O *testware* pode ser difícil de manter, exigindo várias mudanças quando o *software* em teste for alterado;
- O envolvimento direto do testador na execução dos testes pode ser reduzido, o que leva a uma menor detecção de defeitos;
- A equipe de teste pode ter habilidades insuficientes para utilizar as ferramentas automatizadas de maneira eficaz;
- Testes irrelevantes que não contribuam com a cobertura geral do teste podem ser automatizados porque existem e são estáveis;
- Os testes podem ficar improdutivos à medida que o *software* se estabilizar (paradoxo do pesticida).

Durante a implantação de uma ferramenta de automação de execução de testes, nem sempre é bom automatizar os casos de teste manuais no estado em que se encontram.

É melhor redefinir os casos de teste para um melhor uso da automação. Isto inclui a formação de casos de teste, a reutilização de padrões, a expansão das entradas através do uso de variáveis em vez de valores pré-programados e a utilização de todas as vantagens da ferramenta de teste.

As ferramentas de execução de testes costumam realizar vários testes, agrupar testes, repetir testes e mudar a ordem de execução ao mesmo tempo em que fazem análises e divulgam os resultados.

Em muitas ferramentas de automação de execução de testes, as habilidades de programação são necessárias para criar testes (*scripts*) e suítes de testes eficientes e eficazes. É comum que seja bem difícil atualizar e gerenciar grandes suítes de testes automatizados se não forem modeladas com cuidado.

O treinamento adequado em ferramentas de teste, programação e técnicas de modelagem é valioso para garantir o aproveitamento de todas as vantagens das ferramentas.

Durante o planejamento do teste, é importante reservar tempo para a execução manual periódica de casos de teste automatizados para aprender como o teste funciona, verificar o funcionamento correto e revisar a validade dos dados de entrada e a cobertura.

7.2.2.5 Automação orientada para palavras-chave

As palavras-chave (às vezes denominadas comandos) são, em sua maioria, mas não exclusivamente, utilizadas para representar interações de negócios de alto nível com um sistema (por exemplo, um pedido de cancelamento).

Cada palavra-chave é normalmente utilizada para representar um número de interações detalhadas entre um ator e o sistema em teste. As sequências de palavras-chave (inclusive os dados de teste relevantes) são utilizadas para especificar casos de teste. [Buwalda01]

Na automação de testes, a palavra-chave é implementada como um ou mais *scripts* de teste executáveis. As ferramentas leem casos de teste escritos como uma sequência de palavras-chave que executa os *scripts* de teste adequados, os quais implementam a funcionalidade da palavra-chave.

Os *scripts* são implementados de maneira altamente modular para possibilitar o fácil mapeamento de palavras-chave específicas. As habilidades de programação são necessárias para implementar tais *scripts* modulares.

As principais vantagens da automação de testes orientados para palavras-chave são:

- Palavras-chave que dizem respeito a um aplicativo ou a um domínio de negócios específico podem ser definidas por especialistas em domínios. Isto pode deixar a tarefa da especificação do caso de teste mais eficiente;
- Uma pessoa com *expertise* principal em domínios pode se beneficiar com a execução automática de casos de teste (assim que as palavras-chave foram implementadas como *scripts*) sem ter que entender o código de automação subjacente;
- É mais fácil fazer a manutenção dos casos de teste escritos com palavras-chave porque é menos provável que precisem de modificações se os detalhes no *software* em teste mudarem;
- As especificações de casos de teste independem da implementação. As palavras-chave podem ser implementadas com diversas linguagens *script* e ferramentas.

Os *scripts* de automação (o verdadeiro código de automação) que utilizam as informações da palavra-chave / do comando costumam ser escritos por desenvolvedores ou *Technical Test Analysts (TTA)*, enquanto o *Test Analyst (TA)* normalmente cria dados de palavras-chave / comandos e faz a manutenção deles. Embora a automação orientada para palavras-chave costume ser executada durante a fase de teste de sistema, o desenvolvimento do código pode começar ainda nas fases de integração. Em um ambiente iterativo, o desenvolvimento de automação de testes é um processo contínuo.

Assim que as palavras-chave e os dados de entrada forem criados, normalmente, o *Test Analyst (TA)* assume a responsabilidade de executar os casos de teste orientados para palavras-chave e analisar quaisquer falhas que possam ocorrer.

Quando uma anomalia é detectada, o *Test Analyst (TA)* deve investigar a causa do fracasso para determinar se o problema está nas palavras-chave, nos dados de entrada, no próprio *script* de automação ou no aplicativo testado. Normalmente, a primeira etapa na resolução de problemas consiste em executar o mesmo teste com os mesmos dados manualmente para ver se a falha está no próprio aplicativo.

Se nenhuma falha for indicada, o *Test Analyst (TA)* deve revisar a sequência de testes que levou à falha para determinar se o problema ocorreu em uma etapa anterior (talvez através da produção de dados incorretos), mas só veio à tona em um momento posterior do processamento.

Se o *Test Analyst (TA)* não conseguir determinar a causa da falha, as informações da resolução de problemas devem ser proporcionadas ao *Technical Test Analyst (TTA)* ou ao desenvolver para a realização de outras análises.

7.2.2.6 Causas de falhas nos trabalhos de automação

Frequentemente, os projetos de automação de execução de testes não conseguem atingir suas metas. As falhas podem acontecer por conta de uma flexibilidade insuficiente na utilização da ferramenta de teste, de habilidades insuficientes de programação na equipe de teste ou de uma expectativa nada realista de que os problemas podem ser resolvidos com a automação de execução de testes.

Vale ressaltar que qualquer automação de execução de testes exige gestão, empenho, habilidade e atenção, assim como qualquer projeto de desenvolvimento de *software*.

Dedicou-se tempo à criação de uma arquitetura sustentável de acordo com as práticas de modelagem adequadas, possibilitando o gerenciamento de configurações, e de acordo com boas práticas de codificação. Os *scripts* de testes automatizados precisam ser testados porque provavelmente contêm defeitos. Talvez o desempenho dos *scripts* precise ser ajustado.

A usabilidade das ferramentas pode ser levada em consideração, não só para o desenvolvedor, mas também para as pessoas que utilizarão a ferramenta para a execução de *scripts*. Talvez seja necessário projetar uma interface entre a ferramenta e o usuário que proporciona acesso aos casos de teste de uma maneira logicamente organizada para o testador, mas que também propicie a acessibilidade de que a ferramenta precisa.

8. Referências

8.1 Normas

- [ISO25000] ISO/IEC 25000:2005, Software Engineering – Software Product Quality Requirements and Evaluation (SQuaRE). Capítulos 1 e 4.
- [ISO9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality. Capítulos 1 e 4.
- [RTCA DO-178B/ED-12B]: Software Considerations in Airborne Systems and Equipment Certification, RTCA/EUROCAE ED12B.1992. Capítulo 1.

8.2 Documentos do ISTQB

- [BSTQB_AL_OVIEW] *BSTQB Advanced Level Overview*, versão 1.0
- [BSTQB_ALTM_SYL] *BSTQB Advanced Level Test Manager (TM) Syllabus*, versão 1.0
- [BSTQB_ALTTA_SYL] *BSTQB Advanced Level Technical Test Analyst (TTA) Syllabus*, versão 1.0 [BSTQB_FL_SYL] *BSTQB Foundation Level Syllabus*, versão 2011
- [BSTQB_GLOSSARY] *Glossary de Termos de Teste de Software*, versão 3.0

8.3 Livros

- [Bath08] BATH, Graham; MCKAY, Judy. *The Software Test Engineer's Handbook* – Rocky Nook, 2008, ISBN 978-1-933952-24-6.
- [Beizer95] BEIZER, Boris. *Black-Box Testing* – John Wiley & Sons, 1995, ISBN 0-471-12094-4.
- [Black02] BLACK, Rex. *Managing the Testing Process* – 2ª edição, Nova Iorque: John Wiley & Sons, 2002, ISBN 0-471-22398-0.
- [Black07] BLACK, Rex. *Pragmatic Software Testing* – John Wiley and Sons, 2007, ISBN 978-0-470-12790-2.
- [Buwalda01] BUWALDA, Hans. *Integrated Test Design and Automation* – Addison-Wesley Longman, 2001, ISBN 0-201-73725-6.
- [Cohn04] COHN, Mike. *User Stories Applied: For Agile Software Development* – Addison-Wesley Professional, 2004, ISBN 0-321-20568-5.
- [Copeland03] COPELAND, Lee. *A Practitioner's Guide to Software Test Design* – Artech House, 2003, ISBN 1-58053-791-X.
- [Craig02] CRAIG, Rick David; JASKIEL, Stefan P. *Systematic Software Testing* – Artech House, 2002, ISBN 1-580-53508-9.
- [Gerrard02] GERRARD, Paul; THOMPSON, Neil. *Risk-Based e-Business Testing* – Artech House, 2002, ISBN 1-580-53314-0.
- [Gilb93] DOROTHY, Graham; GILB, Tom. *Software Inspection* – Addison-Wesley, 1993, ISBN 0-201-63181-4.
- [Graham07] BLACK, Rex; EVANS, Isabel; GRAHAM, Dorothy; VEENENDAAL, Erik van. *Foundations of Software Testing* – Thomson Learning, 2007, ISBN 978-1-84480-355-2.
- [Grochmann94] GROCHMANN, M. "Test Case Design Using Classification Trees," in: *Conference Proceedings of STAR 1994*, 1994.

- [Koomen06] AALST, Leo van der; BROEKMAN, Bart; KOOMEN, Tim; VROON, Michiel. *TMap NEXT, For Result Driven Testing* – UTN Publishers, 2006, ISBN 90-72194-80-2.
- [Myers79] MYERS, Glenford J. *The Art of Software Testing* – John Wiley & Sons, 1979, ISBN 0-471-46912-2.
- [Splaine01] JASKIEL, Stefan P.; SPLAINE, Steven. *The Web-Testing Handbook* – STQE Publishing, 2001, ISBN 0-970-43630-0.
- [vanVeenendaal12] VEENENDAAL, Erik van. *Practical Risk-Based Testing – The PRISMA Approach* – Holanda, UTN Publishers, 2012, ISBN 9789490986070.
- [Wiegers03] WIEGERS, Karl. *Software Requirements 2* – Microsoft Press, 2003, ISBN 0-735-61879-8.
- [Whittaker03] WHITTAKER, James. *How to Break Software* – Addison-Wesley, 2003, ISBN 0-201-79619-8.
- [Whittaker09] WHITTAKER, James. *Exploratory Software Testing* – Addison-Wesley, 2009, ISBN 0-321-63641-4.

8.4 Outras referências

As seguintes referências contêm informações disponíveis na Internet e em outros lugares. Embora estas referências tenham sido acessadas até o momento da publicação deste *syllabus* de nível avançado, o ISTQB não pode ser responsabilizada se as referências não estiverem mais disponíveis.

Capítulo 3:

- CZERWONKA, Jacek: www.pairwise.org;
- Taxonomia de *bugs*: www.testingeducation.org/a/bsct2.pdf;
- Taxonomia de amostras de *bugs* com base na obra de Boris Beizer: inet.uni2.dk/~vinter/bugtaxst.doc;
- Um bom apanhado das diversas taxonomias: testingeducation.org/a/bugtax.pdf;
- *Heuristic Risk-Based Testing* de James Bach;
- *Exploratory & Risk-Based Testing* (2004): www.testingeducation.org;
- *Exploring Exploratory Testing*, Cem Kaner e Andy Tikam, 2003;
- PETTICHORD, Bret. *An Exploratory Testing Workshop Report*, - www.testingcraft.com/exploratorypettichord.

Capítulo 4:

- www.testingstandards.co.uk