Testes de Software

Capítulo 04
SWEBOK

Professor
Wilson Amaral

Sumário

Testes de Software

INTRODUÇÃO

O teste de software consiste na verificação *dinâmica* de que um programa fornece comportamentos *esperados* em um conjunto *finito* de casos de teste, *selecionados* adequadamente a partir do domínio de execução geralmente infinito.
Na definição acima, as palavras em *itálico* correspondem a questões-chave na descrição da área de conhecimento de Teste de Software:

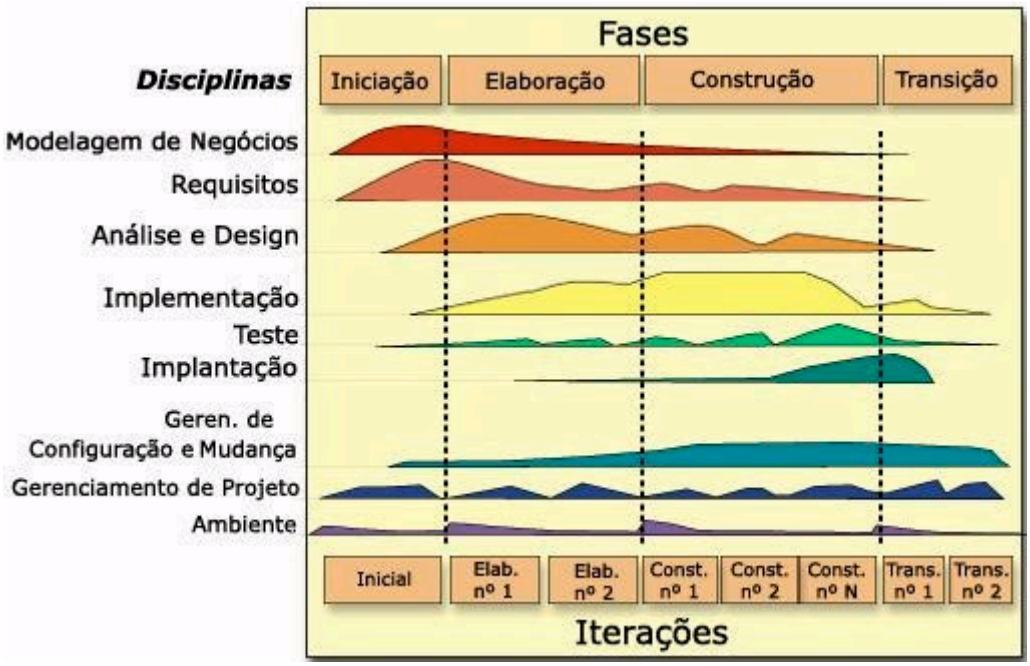
• **Dinâmico**: esse termo significa que o teste sempre implica a execução do programa em entradas selecionadas. Para ser preciso, o valor de entrada sozinho nem sempre é suficiente para especificar um teste, uma vez que um sistema complexo não determinístico pode reagir à mesma entrada com diferentes comportamentos, dependendo do estado do sistema. O termo "entrada" será mantido, com a convenção implícita de que seu significado também inclui um estado de entrada especificado nos casos para os quais é importante.
Técnicas estáticas são diferentes e complementares aos testes dinâmicos. As técnicas estáticas foram abordadas na primeira parte deste documento (Qualidade de Software). Vale a pena notar que a terminologia não é uniforme entre diferentes comunidades e algumas usam o termo "teste" também em referência a técnicas estáticas.

• **Finita**: Mesmo em programas simples, um número infinito de casos de teste são teoricamente possíveis, e testes exaustivos podem exigir meses ou anos para que o conjunto de testes possa ser considerado concluído. O teste é conduzido em um subconjunto de todos os testes possíveis, que é determinado pelos riscos e critérios de priorização.
O teste sempre implica uma troca entre recursos limitados e programações, por um lado, e requisitos de teste inerentemente ilimitados, por outro.

• **Selecionado**: As muitas técnicas de teste propostas diferem essencialmente na forma como o conjunto de teste é selecionado e os engenheiros de software devem estar cientes de que diferentes critérios de seleção podem produzir graus de eficácia muito diferentes. Como identificar o critério de seleção mais adequado sob determinadas condições é um problema complexo; na prática, técnicas de análise de risco e experiência em engenharia de software são aplicadas.

• **Esperado**: Deve ser possível, embora nem sempre fácil, decidir se os resultados observados dos testes do programa são aceitáveis ou não; caso contrário, o esforço de teste é inútil. O comportamento observado pode ser verificado em relação às necessidades do usuário (geralmente chamado de teste de validação), contra uma especificação (teste para verificação) ou, talvez, contra o comportamento previsto de requisitos ou expectativas implícitas[1]. Nos últimos anos, a visão dos testes de software amadureceu em uma forma construtiva. O teste não é mais visto como uma atividade que começa somente após a conclusão da fase de codificação, com o objetivo limitado de detectar falhas. O teste de software é, ou deve ser, abrangente em todo o ciclo de vida de desenvolvimento e manutenção, conforme o gráfico das baleias do RUP abaixo apresentado:

[1] Consulte o item Testes de aceitação no capítulo 01 - Requisitos de Software do SWEBOK



De fato, o planejamento para o teste de software deve começar nos estágios iniciais do processo de requisitos de software e ser continuamente desenvolvido - e possivelmente aperfeiçoado - à medida que o desenvolvimento de software avança. Essas atividades de planejamento de teste e projeto de teste fornecem informações úteis para os projetistas de software e ajudam a destacar possíveis fraquezas, como omissões / contradições de design ou omissões / ambiguidades na documentação.

]Para muitas organizações, a abordagem à qualidade do software é uma prevenção: é obviamente muito melhor prevenir problemas do que corrigi-los. O teste pode ser visto, então, como um meio de fornecer informações sobre os atributos de funcionalidade e qualidade do software e também para identificar falhas nos casos em que a prevenção de erros não foi eficaz. Talvez seja óbvio, mas vale a pena reconhecer que o software ainda pode conter falhas, mesmo após a conclusão de uma extensa atividade de teste. As falhas de software experimentadas após a entrega são tratadas pela manutenção corretiva. Os tópicos de manutenção de software são abordados no capítulo 5 do SWEBOK - Software Maintenance. Na primeira parte deste documento (Qualidade de Software Quality), as técnicas de gerenciamento de qualidade de software são categorizadas em técnicas estáticas (sem execução de código) e técnicas dinâmicas (com execução do software). Ambas categorias são úteis. Este documento se concentra em técnicas dinâmicas. O teste de software também está relacionado à construção de software (consulte Teste de construção no capítulo 03 do SWEBOK - Software Construction). Em particular, os testes de unidade e integração estão intimamente relacionados à construção de software, se não parte dele.

DISCRIMINAÇÃO DE TÓPICOS PARA TESTES DE SOFTWARE

A figura a seguir mostra a divisão dos tópicos para os testes de Software. O primeiro tópico descreve os Fundamentos do Teste de Software. Abrange as definições básicas no campo de testes de software, a terminologia básica e os principais problemas e o relacionamento do teste de software com outras atividades.

O segundo tópico, Níveis de Teste, consiste em dois subtópicos: o primeiro lista os níveis nos quais o teste de softwares de grande porte é tradicionalmente subdividido, e o segundo subtópico considera testes para condições ou propriedades específicas e é referido como Testes Objetivos. Nem todos os tipos de testes se aplicam a todos os produtos de software, nem todos os tipos possíveis serão aqui listados. O objetivo do teste determina como o conjunto de testes é identificado, tanto em relação à sua consistência - *quanto teste é suficiente para alcançar o*

objetivo declarado - quanto à sua composição - *quais casos de teste devem ser selecionados para alcançar o objetivo declarado*, embora geralmente "para alcançar o objetivo declarado" permaneça implícito e apenas a primeira parte das duas perguntas em *itálico* acima é colocada. Critérios para abordar a primeira questão são referidos como critérios de adequação de teste, enquanto aqueles que abordam a segunda questão são os critérios de seleção de teste. Diversas técnicas de teste foram desenvolvidas nas últimas décadas, e novas ainda estão sendo propostas. Técnicas atualmente aceitas são abordadas no terceiro tópico. As Medidas Relacionadas a Testes são tratadas no quarto tópico, enquanto as questões relativas ao Processo de Teste são abordadas no quinto tópico. Finalmente, as ferramentas de teste de software são apresentadas no sexto e último tópico deste documento.

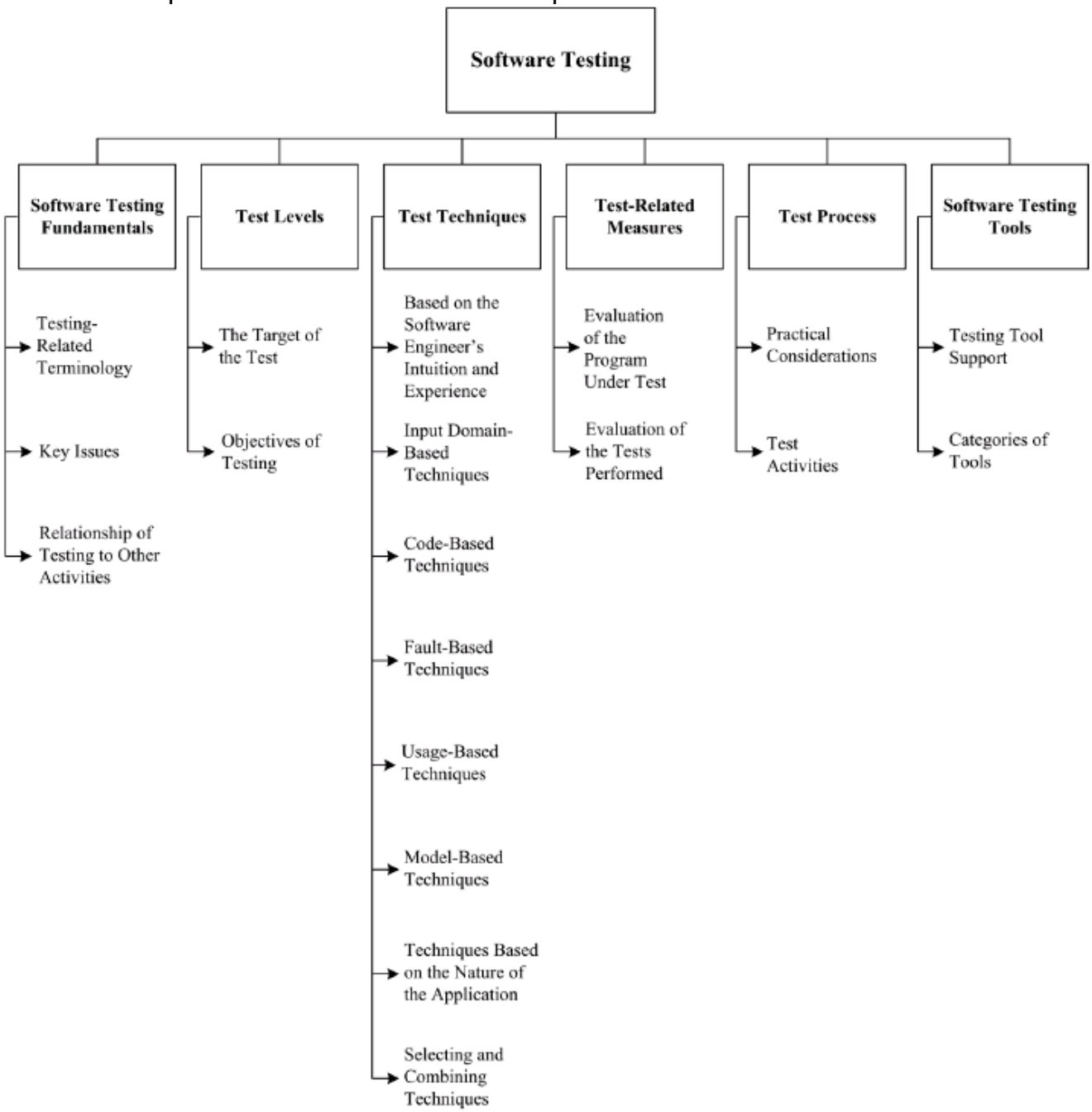


Figure 4.1. Breakdown of Topics for the Software Testing KA

1. Fundamentos do Teste de Software

1.1. Terminologia Relacionada a Testes

1.1.1. Definições de testes e terminologia relacionada

As definições de terminologia relacionada a testes são fornecidas nas referências citadas e resumidas a seguir.

1.1.2. Faltas vs. Falhas

Muitos termos são usados na literatura de engenharia de software para descrever um mau funcionamento: notavelmente falta, falha e erro, entre outros. Essa terminologia é definida com precisão na primeira parte deste documento. É essencial distinguir claramente entre a causa de um mau funcionamento (para o qual o termo falta será usado aqui) e um efeito indesejado observado no serviço entregue do sistema (que será chamado de falha).

Na verdade, pode haver faltas no software que nunca se manifestam como falhas (veja Limitações Teóricas e Práticas de Teste na seção 1.2, Questões-chave). Assim, o teste pode revelar falhas, mas são as faltas que podem e devem ser removidas[2]. O termo genérico "defeito" pode ser usado para se referir a uma falta ou falha, quando a distinção não for tão importante[24]. No entanto, deve ser reconhecido que a causa de uma falha nem sempre pode ser inequivocamente identificada. Não existem critérios teóricos para determinar definitivamente, em geral, a falta que causou uma falha observada. Pode-se dizer que foi a falta que teve que ser modificada para remover a falha, mas outras modificações poderiam ter funcionado tão bem. Para evitar a ambiguidade, poder-se-ia referir a entradas causadoras de falhas, em vez de faltas - isto é, aqueles conjuntos de entradas que causam uma falha na exibição.

1.2. Assuntos chave

1.2.1. Critérios de Seleção de Testes / Critérios de Adequação de Testes (Regras de Parada)

Um critério de seleção de teste é um meio de selecionar casos de teste ou determinar que um conjunto de casos de teste é suficiente para uma finalidade especificada. Os critérios de adequação dos testes podem ser usados para decidir quando testes suficientes serão ou foram realizados[3].

1.2.2. Teste de Eficácia / Objetivos para Testes

A eficácia dos testes é determinada pela análise de um conjunto de execuções de programas. A seleção de testes a serem executados pode ser guiada por diferentes objetivos: é somente à luz do objetivo perseguido que a eficácia do conjunto de testes pode ser avaliada.

1.2.3. Testando a detecção de defeitos

No teste de descoberta de defeitos, um teste bem-sucedido é aquele que faz com que o sistema falhe. Isso é bem diferente do teste para demonstrar que o software atende às suas especificações ou outras propriedades desejadas, caso em que o teste é bem-sucedido se nenhuma falha for observada em casos de teste realistas e ambientes de teste.

1.2.4. O Problema do Oráculo

Um oráculo é qualquer agente humano ou mecânico que decide se um programa se comportou corretamente em um determinado teste e, consequentemente, resulta em um veredicto de "passar" ou "falhar". Existem muitos tipos diferentes de oráculos; por exemplo, especificações de requisitos não ambíguas, modelos comportamentais e anotações de código. A automação de oráculos mecanizados pode ser difícil e cara.

---

[2] M.R. Lyu, ed., Handbook of Software Reliability Engineering, McGraw-Hill and IEEE Computer Society Press, 1996.

[3] H. Zhu, P.A.V. Hall, and J.H.R. May, "Software Unit Test Coverage and Adequacy," ACM Computing Surveys, vol. 29, no. 4, Dec. 1997, pp. 366–427.

1.2.5. Limitações Teóricas e Práticas dos Testes

A Teoria dos Testes adverte contra a atribuição de um nível de confiança injustificado a uma série de testes bem sucedidos. Infelizmente, os resultados mais estabelecidos da teoria de testes são negativos, pois afirmam que os testes nunca podem ser obtidos em oposição ao que é realmente alcançado. A mais famosa citação a esse respeito é o aforismo de Dijkstra de que "o teste de programa pode ser usado para mostrar a presença de erros, mas nunca para mostrar sua ausência"[4]. A razão óbvia para isso é que testes completos não são viáveis em softwares realistas. Por causa disso, o teste deve ser conduzido com base no risco[5] e pode ser visto como uma estratégia de gerenciamento de risco.

1.2.6. O problema dos caminhos inviáveis

Caminhos inviáveis são caminhos de fluxo de controle que possuem um erro de codificação que faz com que aquele trecho de código nunca seja executado independentemente do dado de entrada fornecido. Eles são um problema significativo em testes baseados em caminhos, particularmente na derivação automatizada de entradas de teste para exercer caminhos de fluxo de controle. Veja o exemplo abaixo:

```
if (k<2)
{ if (K > 3)   // Aqui deveria ser -3
   K = K*N;  // Esta linha de código nunca será executada, independente do valor de K
}
```

1.2.7. Testabilidade

O termo "testabilidade de software" tem dois significados relacionados, porém diferentes: por um lado, refere-se à facilidade com que um dado critério de cobertura de teste pode ser satisfeito; por outro lado, é definido como a probabilidade, possivelmente medida estatisticamente, de que um conjunto de casos de teste exporá uma falha se o software estiver com defeito. Ambos os significados são importantes.

1.3. Relacionamento do teste com outras atividades

O teste de software está relacionado, às técnicas de gerenciamento estático de qualidade de software e às provas de correção, depuração e construção de programas, no entanto, vale considerar o teste do ponto de vista de analistas de qualidade de software e de certificadores:

• Testes vs. Técnicas de Gerenciamento de Qualidade de Software Estático (para detalhes, consulte Técnicas de Gerenciamento de Qualidade de Software em "Qualidade de Software" no início deste documento;

• Teste vs. Verificação de Exatidão e Verificação Formal (para detalhes, consulte Modelos e Métodos de Engenharia de Software no capítulo 9, desenvolvimento de Software no capítulo 3 e Ferramentas e Técnicas de Depuração no capítulo 15 do SWEBOK);

• Teste vs. Construção de Programa (para mais detalhes, consulte Desenvolvimento de Teste no capítulo 3 do SWEBOK).

2. Níveis de Teste

O teste de software geralmente é realizado em diferentes níveis durante os processos de desenvolvimento e manutenção. Os níveis podem ser distinguidos com base no objeto de teste, que é chamado de alvo, ou na finalidade, que é chamada de objetivo (do nível de teste).

[4] E.W. Dijkstra, "Notes on Structured Programming," T.H.-Report 70-WSE-03, Technological University, Eindhoven, 1970; http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF.

[5] Concepts and Definitions, ISO/IEC/IEEE, 2012.

2.1. O Alvo do Teste

O alvo do teste pode variar: um único módulo, um grupo de tais módulos (relacionados por finalidade, uso, comportamento ou estrutura) ou um sistema inteiro. Três estágios de teste podem ser distinguidos: unidade, integração e sistema. Esses três estágios de teste não implicam em nenhum modelo de processo, e nenhum deles é considerado mais importante que os outros dois.

2.1.1. Teste de unidade

O Teste Unitário verifica o funcionamento isolado dos elementos de software que são testáveis separadamente. Dependendo do contexto, estes podem ser os subprogramas individuais ou um componente maior feito de unidades altamente coesivas. Normalmente, o teste de unidade ocorre com acesso ao código que está sendo testado e com o suporte de ferramentas de depuração. Os programadores que escreveram o código normalmente (mas nem sempre) conduzem testes unitários.

2.1.2. Teste de Integração

O teste de integração é o processo de verificar as interações entre os componentes de software. Estratégias clássicas de teste de integração, como topdown e bottom-up, são frequentemente usadas com software estruturado hierarquicamente. Estratégias de integração modernas e sistemáticas são tipicamente direcionadas à arquitetura, o que envolve a integração gradual dos componentes ou subsistemas de software com base em segmentos funcionais identificados. O teste de integração geralmente é uma atividade contínua em cada estágio do desenvolvimento, durante o qual os engenheiros de software abstraem as perspectivas de nível inferior e concentram-se nas perspectivas do nível em que estão integrando. Para outros, além do software pequeno e simples, as estratégias de teste de integração incremental geralmente são preferidas para reunir todos os componentes de uma só vez - o que geralmente é chamado de teste "big bang".

2.1.3. Teste do sistema

O teste do sistema está relacionado ao teste do comportamento de todo um sistema. A unidade eficaz e o teste de integração terão identificado muitos dos defeitos do software. O teste do sistema é geralmente considerado apropriado para avaliar os requisitos não funcionais do sistema - como segurança, velocidade, precisão e confiabilidade (consulte Requisitos funcionais e não funcionais no capítulo 01 do SWEBOK). Interfaces externas para outros aplicativos, utilitários, dispositivos de hardware ou os ambientes operacionais também são geralmente avaliados nesse nível.

2.2. Objetivos do teste

Os testes são conduzidos em função de objetivos específicos, que são declarados de forma mais ou menos explícita e com diferentes graus de precisão. Declarar os objetivos do teste em termos precisos e quantitativos suporta a medição e o controle do processo de teste. O teste pode ser destinado a verificar propriedades diferentes. Os casos de teste podem ser projetados para verificar se as especificações funcionais estão corretamente implementadas, o que é referido na literatura como testes de conformidade, testes de correção ou testes funcionais. No entanto, várias outras propriedades não funcionais também podem ser testadas, incluindo desempenho, confiabilidade e usabilidade, entre muitas outras (consulte Modelos e características de qualidade no início deste documento). Outros objetivos importantes para o teste incluem a medição de confiabilidade, identificação de vulnerabilidades de segurança, avaliação de usabilidade e aceitação de software, para os quais diferentes abordagens são adotadas. Observe que, em geral, os objetivos do teste variam de acordo com a meta de teste; diferentes finalidades são abordadas em diferentes níveis de teste. Os subtópicos listados abaixo são os mais citados na literatura. Observe que alguns tipos de testes são mais apropriados para pacotes de software personalizados - testes de instalação, por exemplo - e outros para produtos de consumo, como o teste beta.

2.2.1. Teste de Aceitação / Qualificação

O teste de aceitação (ou qualificação) determina se um sistema satisfaz seus critérios de aceitação, geralmente verificando os comportamentos desejados do sistema em relação aos requisitos do cliente. O cliente ou o representante de um cliente, portanto, especifica ou realiza atividades diretamente para verificar se seus requisitos foram atendidos ou, no caso de um produto de consumo, se a organização atendeu aos requisitos estabelecidos para o mercado-alvo. Esta atividade de teste pode ou não envolver os desenvolvedores do sistema.

2.2.2. Testes de Instalação

Frequentemente, após a conclusão do sistema e do teste de aceitação, o software é verificado após a instalação no ambiente de destino. Os testes de instalação podem ser vistos como testes de sistema realizados no ambiente operacional de configurações de hardware e outras restrições operacionais. Os procedimentos de instalação também podem ser verificados.

2.2.3. Testes Alfa e Beta

Antes do software ser lançado, às vezes é submetido a um pequeno grupo selecionado de usuários para uso experimental (teste alfa), e posteriormente ou para um conjunto maior de usuários representativos (teste beta). Esses usuários relatam problemas com o produto. Os testes alfa e beta geralmente não são controlados e nem sempre são mencionados em um plano de teste.

2.2.4. Teste de Confiabilidade

Teste de avaliação da confiabilidade que identifica e corrige falhas. Além disso, medidas estatísticas de confiabilidade podem ser obtidas gerando aleatoriamente casos de teste de acordo com o perfil operacional do software (para mais detalhes, consulte Perfil Operacional na seção 3.5 - Técnicas Baseadas em Uso, mais abaixo neste documento). A última abordagem é chamada de teste operacional. Usando modelos de crescimento de confiabilidade, ambos os objetivos podem ser alcançados juntos[6] (para mais detalhes, consulte Teste de Vida, Avaliação de Confiabilidade na seção 4.1 - Avaliação do Programa em Teste, mais abaixo neste documento).

2.2.5. Teste de regressão

De acordo com a ISO/IEC/IEEE 24765:2010 [7], o teste de regressão é o "novo teste seletivo de um sistema ou componente para verificar se as modificações não causaram efeitos não intencionais e se o sistema ou componente ainda atende aos requisitos especificados". Objetiva garantir que o software ainda passa nos testes anteriormente realizados após a intervenção. Para desenvolvimento incremental, o objetivo do teste de regressão é mostrar que o comportamento do software não é alterado por mudanças incrementais no software, exceto na medida em que deveria. Em alguns casos, uma compensação deve ser feita entre a garantia dada pelo teste de regressão toda vez que uma alteração é feita e os recursos necessários para executar os testes de regressão, o que pode ser bastante demorado devido ao grande número de testes que podem ser executados. O teste de regressão envolve selecionar, minimizar e / ou priorizar um subconjunto dos casos de teste em um conjunto de testes existente[8]. O teste de regressão pode ser realizado em cada um dos níveis de teste descritos na seção 2.1 - O Alvo do Teste, e pode ser aplicado a testes funcionais e não funcionais.

2.2.6. Teste de desempenho

O teste de desempenho verifica se o software atende aos requisitos de desempenho especificados e avalia as características de performance, por exemplo, capacidade e tempo de resposta.

---

[6] M.R. Lyu, ed., Handbook of Software Reliability Engineering, McGraw-Hill and IEEE Computer Society Press, 1996.
[7] ISO/IEC/IEEE 24765:2010 Systems and Software Engineering—Vocabulary, ISO/ IEC/IEEE, 2010.
[8] S. Yoo and M. Harman, "Regression Testing Minimization, Selection and Prioritization: A Survey," Software Testing Verification and Reliability, vol. 22, no. 2, Mar. 2012, pp. 67–120.

2.2.7. Teste de Segurança
O teste de segurança é focado na verificação de que o software está protegido contra ataques externos. Em particular, o teste de segurança verifica a confidencialidade, integridade e disponibilidade dos sistemas e seus dados. Geralmente, o teste de segurança inclui a verificação contra uso indevido e abuso do software ou sistema (teste negativo).
2.2.8. Teste de estresse
O teste de estresse objetiva submeter o software a carga máxima de acessos e operações com o objetivo de determinar os limites comportamentais e testar os mecanismos de defesa em sistemas críticos.
2.2.9. Teste Back-to-Back
É o teste em que duas ou mais variantes de um programa são executadas com as mesmas entradas, as saídas são comparadas e os erros são analisados em caso de discrepâncias.
2.2.10. Teste de recuperação
O teste de recuperação visa verificar os recursos de reinicialização do software após uma falha do sistema ou outro "desastre".
2.2.11. Teste de Interface
Falhas de Interface são comuns em sistemas complexos. O teste de interface visa verificar se os componentes fazem interface corretamente para fornecer a troca correta de dados e informações de controle. Normalmente, os casos de teste são gerados a partir da especificação da interface. Um objetivo específico do teste de interface é simular o uso de APIs por aplicativos de usuário final. Isso envolve a geração de parâmetros das chamadas da API, a configuração de condições externas do ambiente e a definição de dados internos que afetam a API.
2.2.12. Teste de configuração
Nos casos em que o software é criado para atender a usuários diferentes, o teste de configuração verifica o software em diferentes configurações especificadas.
2.2.13. Teste de Usabilidade
Teste que objetiva avaliar a interação Humano-Computador-Humano. A principal tarefa de usabilidade é testar a interação do usuário com o sistema e avaliar se é fácil para os usuários finais aprender e usar o software. Em geral, pode envolver o teste das funções do software que suportam as tarefas do usuário, a documentação que ajuda os usuários e a capacidade do sistema de se recuperar dos erros do usuário.

3. Técnicas de Teste
Um dos objetivos do teste é detectar tantas falhas quanto possível. Muitas técnicas foram desenvolvidas para fazer isso[9]. Essas técnicas tentam "quebrar" um programa sendo o mais sistemático possível na identificação de entradas que produzirão comportamentos representativos de programas, por exemplo, considerando subclasses do domínio de entrada, cenários, estados e fluxos de dados. A classificação das técnicas de teste apresentadas aqui é baseada em como os testes são gerados: da intuição e experiência do engenheiro de software, as especificações, a estrutura do código, as falhas reais ou imaginadas a serem descobertas, uso previsto, modelos ou a natureza do aplicativo.
Uma categoria lida com o uso combinado de duas ou mais técnicas. Às vezes, essas técnicas são classificadas como caixa branca (também chamada de caixa de vidro), se os testes forem baseados em informações sobre como o software foi projetado ou codificado ou como caixa preta se os casos de teste dependerem apenas da entrada / saída comportamento do software. A lista a seguir inclui as técnicas de teste que são comumente usadas, mas alguns profissionais dependem de algumas técnicas mais do que outras.

[9] ISO/IEC/IEEE P29119-1/DIS Draft Standard for Software and Systems Engineering - Software Testing

3.1. Baseado na Intuição e Experiência do Engenheiro de Software

3.1.1. Ad Hoc

Talvez a técnica mais amplamente praticada seja o teste ad hoc, em que os testes são derivados com base na habilidade, intuição e experiência do engenheiro de software com programas semelhantes. O teste ad hoc pode ser útil para identificar casos de testes que não são facilmente gerados por técnicas mais formalizadas.

3.1.2. Teste exploratório

O teste exploratório é definido como aprendizado simultâneo, desenho de teste e execução de teste[10], ou seja, os testes não são definidos com antecedência em um plano de teste estabelecido, mas são projetados, executados e modificados dinamicamente. A eficácia dos testes exploratórios depende do conhecimento do engenheiro de software, que pode ser derivado de várias fontes: comportamento observado do produto durante o teste, familiaridade com o aplicativo, a plataforma, o processo de falha, o tipo de possíveis faltas e falhas, o risco associado a um determinado produto e assim por diante.

3.2. Técnicas Baseadas no Domínio de Entrada

3.2.1. Particionamento de equivalência

O particionamento de equivalência envolve particionar o domínio de entrada em uma coleção de subconjuntos (ou classes equivalentes) com base em um critério ou relação especificada. Este critério ou relação pode ser resultados computacionais diferentes, uma relação baseada no fluxo de controle ou no fluxo de dados, ou uma distinção feita entre entradas válidas que são aceitas e processadas pelo sistema e entradas inválidas, tais como valores fora do intervalo, que não são aceitas. e deve gerar uma mensagem de erro ou iniciar o processamento de erros. Um conjunto representativo de testes (às vezes apenas um) é geralmente obtido de cada classe de equivalência.

3.2.2. Teste emparelhado

Os casos de teste são derivados combinando valores interessantes para cada par de um conjunto de variáveis de entrada, em vez de considerar todas as combinações possíveis. O teste emparelhado pertence ao teste combinatorial, que em geral também inclui combinações de nível superior aos pares: essas técnicas são chamadas de t-wise, sendo considerada toda combinação possível de variáveis de entrada t.

3.2.3. Análise de valor limite

Os casos de teste são escolhidos nos limites do domínio de entrada das variáveis ou perto deles, com o raciocínio subjacente de que muitas falhas tendem a se concentrar perto dos valores extremos de entradas. Uma extensão desta técnica é o teste de robustez, em que os casos de teste também são escolhidos fora do domínio de entrada das variáveis para testar a robustez do programa no processamento de entradas inesperadas ou erradas.

3.2.4. Teste aleatório

Os testes são gerados de forma puramente aleatória (não confundir com o teste estatístico do perfil operacional, conforme descrito no Perfil Operacional na seção 3.5). Essa forma de teste se enquadra no título do teste de domínio de entrada, pois o domínio de entrada deve ser conhecido para poder escolher pontos aleatórios dentro dele. O teste aleatório fornece uma abordagem relativamente simples para automação de testes; recentemente, formas aprimoradas de testes aleatórios têm sido propostas, nas quais a amostragem de entrada aleatória é direcionada por outros critérios de seleção de entrada[11]. O teste de fuzz ou fuzzing é uma forma especial de teste aleatório destinado a quebrar o software; é mais usado para testes de segurança.

---

[10] ISO/IEC/IEEE P29119-1/DIS Draft Standard for Software and Systems Engineering - Software Testing - Concepts and Definitions, ISO/IEC/IEEE, 2012.

[11] T.Y. Chen et al., "Adaptive Random Testing: The ART of Test Case Diversity," Journal of Systems and Software, vol. 83, no. 1, Jan. 2010, pp. 60–66.

3.3. Técnicas Baseadas em Código

3.3.1. Critérios Baseados em Fluxo de Controle

Os critérios de cobertura baseados em fluxo de controle visam cobrir todas as instruções, blocos de instruções ou combinações especificadas de instruções em um programa. O mais forte dos critérios baseados no fluxo de controle é o teste de caminho, que visa executar todos os caminhos de fluxo de controle de entrada-saída no gráfico de fluxo de controle de um programa. Como o teste de caminho exaustivo geralmente não é viável devido a loops, outros critérios menos rigorosos focam na cobertura de caminhos que limitam as iterações de loop, como cobertura de instrução, cobertura de instruções condicionais / decisão. A adequação de tais testes é medida em porcentagens.

3.3.2. Critérios Baseados em Fluxo de Dados

No teste baseado em fluxo de dados, o gráfico de fluxo de controle é anotado com informações sobre como as variáveis do programa são definidas, usadas e eliminadas. O critério mais forte, todos os caminhos de uso de definição, exige que, para cada variável, todo o segmento de caminho de fluxo de controle, de uma definição dessa variável até o uso dessa definição, seja executado. A fim de reduzir o número de caminhos necessários, são empregadas estratégias mais fracas, como todas as definições e todas as utilizações.

3.3.3. Modelos de referência para testes baseados em código

Embora não seja uma técnica em si, a estrutura de controle de um programa pode ser representada graficamente usando um gráfico de fluxo para visualizar técnicas de teste baseadas em código. Um grafo de fluxo é um grafo direcionado, cujos nós e arcos correspondem aos elementos do programa (para mais detalhes, veja Gráficos e Árvores no capítulo 14 do SWEBOK). Por exemplo, os nós podem representar instruções ou sequências ininterruptas de instruções, e arcos podem representar a transferência de controle entre nós.

3.4. Técnicas Baseadas em Falhas

Com diferentes graus de formalização, as técnicas de teste baseadas em falhas concebem casos de teste especificamente destinados a revelar categorias de falhas prováveis ou predefinidas. Para melhor focar a geração ou seleção do caso de teste, um modelo de falha pode ser introduzido para classificar os diferentes tipos de falhas.

3.4.1. Adivinhação de erro

Na adivinhação de erro, os casos de teste são especificamente projetados por engenheiros de software que tentam antecipar as falhas mais plausíveis em um determinado programa. Uma boa fonte de informações é o histórico de falhas descobertas em projetos anteriores, bem como a experiência do engenheiro de software.

3.4.2. Teste de Mutação

Um mutante é uma versão ligeiramente modificada do programa em teste, diferindo dele por uma pequena alteração sintática. Cada caso de teste exerce o programa original e todos os mutantes gerados: se um caso de teste for bem sucedido na identificação da diferença entre o programa e um mutante, o último é considerado "morto". Originalmente concebido como uma técnica para avaliar conjuntos de teste (ver secção 4.2 Avaliação dos Testes Realizados), o teste de mutação é também um critério de teste em si mesmo: ou os testes são gerados aleatoriamente até que mutantes suficientes tenham sido "mortos", ou testes sejam especificamente projetados para "matar" mutantes sobreviventes. Neste último caso, o teste de mutação também pode ser categorizado como uma técnica baseada em código. A suposição subjacente do teste de mutação, o efeito de acoplamento, é que, procurando por falhas sintáticas simples, falhas mais complexas, mas reais, serão encontradas.

Para que a técnica seja eficaz, um grande número de mutantes deve ser gerado e executado automaticamente de forma sistemática[12].

3.5. Técnica baseada em uso

3.5.1. Perfil Operacional

No teste de avaliação de confiabilidade (também chamado de teste operacional), o ambiente de teste reproduz o ambiente operacional do software ou o perfil operacional, tanto quanto possível. O objetivo é inferir dos resultados dos testes observados a confiabilidade futura do software quando em uso real. Para fazer isso, as entradas são atribuídas a probabilidades ou perfis, de acordo com sua frequência de ocorrência na operação real. Os perfis operacionais podem ser usados durante o teste do sistema para orientar a derivação de casos de teste que avaliarão o alcance dos objetivos de confiabilidade e exercerão o uso relativo e a criticidade de diferentes funções semelhantes ao que será encontrado no ambiente operacional[13].

3.5.2. Observação do usuário Heurística

Os princípios de usabilidade podem fornecer diretrizes para descobrir problemas no design da interface do usuário (consulte Design da interface do usuário no Capítulo 02 do SWEBOK). As heurísticas especializadas, também chamadas de métodos de inspeção de usabilidade, são aplicadas para a observação sistemática do uso do sistema sob condições controladas, a fim de determinar quão bem as pessoas podem usar o sistema e suas interfaces.

As heurísticas de usabilidade incluem orientações cognitivas, análise de reivindicações, observações de campo, pensamento em voz alta e até mesmo abordagens indiretas, como questionários de usuários e entrevistas.

3.6. Técnicas de teste baseadas em modelo

Um modelo neste contexto é uma representação abstrata (formal) do software em teste ou de seus requisitos de software (consulte Modelagem no capítulo 9 do SWEBOK). O teste baseado em modelo é usado para validar requisitos, verificar sua consistência e gerar casos de teste com foco nos aspectos comportamentais do software. Os principais componentes do teste baseado em modelo são[14]: a notação usada para representar o modelo do software ou seus requisitos; modelos de fluxo de trabalho ou modelos similares; a estratégia de teste ou algoritmo usado para geração de casos de teste; a infraestrutura de suporte para a execução do teste; e a avaliação dos resultados dos testes em comparação com os resultados esperados. Devido à complexidade das técnicas, as abordagens de teste baseadas em modelo são frequentemente usadas em conjunto com os softwares de automação de teste.

As técnicas de teste baseadas em modelo incluem:

3.6.1. Tabelas de Decisão

As tabelas de decisão representam relações lógicas entre condições (entradas) e ações (saídas). Os casos de teste são sistematicamente derivados considerando todas as combinações possíveis de condições e suas ações resultantes correspondentes. Uma técnica relacionada é o gráfico de causa e efeito[15].

3.6.2. Máquinas de estados finitos

Ao modelar um programa como uma máquina de estados finitos, os testes podem ser selecionados para cobrir os estados e transições.

3.6.3. Especificações Formais

A descrição das especificações em uma linguagem formal (consulte Métodos Formais no capítulo 9 do SWEBOK) permite a derivação automática de casos de teste funcionais e, ao

[12] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," IEEE Trans. Software Engineering, vol. 37, no. 5, Sep.–Oct. 2011, pp. 649–678.
[13] M.R. Lyu, ed., Handbook of Software Reliability Engineering, McGraw-Hill and IEEE Computer Society Press, 1996.
[14] M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.
[15] S. Naik and P. Tripathy, Software Testing and Quality Assurance: Theory and Practice, Wiley-Spektrum, 2008.

mesmo tempo, fornece um norte para verificar os resultados do teste. O TTCN3 (Testing and Test Control Notation versão 3) é uma linguagem desenvolvida para escrever casos de teste. A notação foi concebida para as necessidades específicas do teste de sistemas de telecomunicações, por isso é particularmente adequado para testar protocolos de comunicação complexos.

3.6.4. Modelos de fluxo de trabalho

Os modelos de fluxo de trabalho especificam uma sequência de atividades realizadas por humanos e / ou aplicativos de software, geralmente representados por meio de notações gráficas. Cada sequência de ações constitui um fluxo de trabalho (também chamado de cenário). Fluxos de trabalho típicos e alternativos devem ser testados[16]. Um foco especial nas funções em uma especificação de fluxo de trabalho é direcionado ao teste de processos de negócios.

3.7. Técnicas baseadas na natureza da aplicação

As técnicas acima se aplicam a todos os tipos de software. Técnicas adicionais para derivação e execução de testes são baseadas na natureza do software que está sendo testado; por exemplo:

- software orientado a objeto
- software baseado em componentes
- software baseado na web
- programas simultâneos
- software baseado em protocolo
- sistemas em tempo real
- sistemas críticos para segurança
- software orientado a serviços
- software de código aberto
- software embarcado

3.8 . Selecionando e Combinando Técnicas

3.8.1. Combinação Funcional e estrutural

A combinação de técnicas de teste baseadas em modelo funcional e estrutural e baseadas em código são frequentemente contrastadas como testes funcionais versus testes estruturais. Essas duas abordagens para testar a seleção não devem ser vistas como alternativas, mas sim como complementos, na verdade, eles usam diferentes fontes de informação e demonstraram destacar diferentes tipos de problemas. Eles poderiam ser usados em combinação, dependendo das considerações orçamentárias.

3.8.2. Determinístico vs. Randômico

Os casos de teste podem ser selecionados de maneira determinística, de acordo com uma das muitas técnicas, ou aleatoriamente, extraídos de alguma distribuição de inputs, como é normalmente feito em testes de confiabilidade. Várias comparações analíticas e empíricas foram realizadas para analisar as condições que tornam uma abordagem mais eficaz que a outra.

4. Medidas Relacionadas a Testes

Às vezes, as técnicas de teste são confundidas com os objetivos do teste. As técnicas de teste podem ser vistas como auxílios que ajudam a garantir a realização dos objetivos do teste[17]. A medição é geralmente considerada fundamental para a análise de qualidade. A medição também pode ser usada para otimizar o planejamento e a execução dos testes. O gerenciamento de teste pode usar várias medidas de processo diferentes para monitorar o

[16] ISO/IEC/IEEE P29119-1/DIS Draft Standard for Software and Systems Engineering - Software Testing
[17] ISO/IEC/IEEE P29119-1/DIS Draft Standard for Software and Systems Engineering - Software Testing

progresso. (Veja seção 5.1, Considerações Práticas, para uma discussão de medidas do processo de teste útil para propósitos de administração.)

4.1. Avaliação do Programa sob Teste

4.1.1. Medidas de Programa que Ajudam no Planejamento e Projeto de Testes

Medidas baseadas em tamanho de software (por exemplo, linhas de código-fonte ou tamanho funcional, ver Requisitos de Medição no capítulo 1 - Requisitos de Software do SWEBOK) ou na estrutura do programa podem ser usadas para guiar o teste. As medidas estruturais também incluem medições que determinam a frequência com que os módulos chamam uns aos outros.

4.1.2. Tipos de falhas, classificação e estatísticas

A literatura de testes é rica em classificações e taxonomias de falhas. Para tornar o teste mais eficaz, é importante saber quais tipos de falhas podem ser encontradas no software em teste e a frequência relativa com a qual essas falhas ocorreram no passado. Essas informações podem ser úteis para fazer previsões de qualidade, bem como para melhoria de processos (consulte Caracterização de defeitos no início deste documento).

4.1.3. Densidade de Falha

Um programa sob teste pode ser avaliado contando as falhas descobertas como a razão entre o número de falhas encontradas e o tamanho do programa.

4.1.4. Teste de Vida, Avaliação da Confiabilidade

Uma estimativa estatística da confiabilidade do software, que pode ser obtida pela observação da confiabilidade obtida, pode ser usada para avaliar um produto de software e decidir se o teste pode ou não ser interrompido (ver seção 2.2 Confiabilidade e Avaliação).

4.1.5. Modelos de Crescimento de Confiabilidade

Os modelos de crescimento de confiabilidade fornecem uma previsão de confiabilidade baseada em falhas. Eles assumem, em geral, que quando as faltas que causaram as falhas observadas foram corrigidas (embora alguns modelos também aceitem correções imperfeitas), a confiabilidade estimada do produto exibe, em média, uma tendência crescente. Existem muitos modelos de crescimento de confiabilidade publicados. Notavelmente, esses modelos são divididos em modelos de contagem de falhas e tempo entre falhas.

4.2. Avaliação dos Testes Realizados

4.2.1. Medidas de Cobertura / Exaustão

Vários critérios de adequação de teste exigem que os casos de teste exercitem sistematicamente um conjunto de elementos identificados no programa ou nas especificações (consulte o tópico 3, Técnicas de Teste). Para avaliar a eficácia dos testes executados, os engenheiros de software podem monitorar os elementos cobertos para que possam medir dinamicamente a proporção entre os elementos cobertos e o número total. Por exemplo, é possível medir a porcentagem de ramificações cobertas no gráfico de fluxo do programa ou a porcentagem de requisitos funcionais exercidos entre os listados no documento de especificações. Critérios de adequação baseados em códigos requerem instrumentação apropriada do programa em teste.

4.2.2. Semeadura de falhas (O teste do teste)

Na semeadura de falhas, algumas falhas são introduzidas artificialmente em um programa antes do teste. Quando os testes são executados, algumas dessas falhas serão reveladas, bem como, possivelmente, algumas falhas que já existiam. Em teoria, dependendo de quais e quantas falhas artificiais são descobertas, a eficácia do teste pode ser avaliada e o número restante de falhas genuínas pode ser estimado.

Na prática, os estatísticos questionam a distribuição e a representatividade das falhas semeadas em relação às falhas genuínas e o pequeno tamanho da amostra em que se baseiam quaisquer extrapolações. Alguns também argumentam que essa técnica deve ser usada com muito cuidado, já que inserir falhas no software envolve o risco óbvio de deixá-las lá.

4.2.3. Escore de mutação

Em testes de mutação (ver Teste de Mutação na seção 3.4, Técnicas Baseadas em Falhas), a razão de mutantes mortos para o número total de mutantes gerados pode ser uma medida da eficácia do conjunto de testes executado.

4.2.4. Comparação e Eficácia Relativa de Diferentes Técnicas

Vários estudos foram realizados para comparar a eficácia relativa de diferentes técnicas de teste. É importante ser preciso quanto à propriedade contra a qual das técnicas estão sendo avaliadas.

O que, por exemplo, é o significado exato dado ao termo "efetividade"?

Possíveis interpretações incluem o número de testes necessários para encontrar a primeira falha, a razão entre o número de falhas encontradas através do teste e todas as falhas encontradas durante e após o teste, e quanta confiabilidade foi melhorada. Comparações analíticas e empíricas entre diferentes técnicas foram conduzidas de acordo com cada uma das noções de eficácia especificadas acima.

5. Processo de Teste

Testar conceitos, estratégias, técnicas e medidas precisam ser integrados em um processo definido e controlado. O processo de teste suporta atividades de teste e fornece orientação a testadores e equipes de teste, desde o planejamento do teste até a avaliação do resultado do teste, de modo a garantir que os objetivos do teste serão atingidos de maneira econômica.

5.1. Considerações Práticas

5.1.1. Atitudes / Programação sem necessidade

Um elemento importante do teste bem-sucedido é uma atitude colaborativa em relação às atividades de teste e garantia de qualidade. Os gerentes têm um papel fundamental na promoção de uma recepção geralmente favorável à descoberta e correção de falhas durante o desenvolvimento e manutenção de software; por exemplo, superando a mentalidade de propriedade de código individual entre programadores e promovendo um ambiente colaborativo com responsabilidade de equipe por anomalias no código.

5.1.2. Guias de teste

As fases de teste podem ser orientadas por vários objetivos. Por exemplo, testes baseados em riscos usam os riscos do produto para priorizar e focar a estratégia de teste casos com base em cenários de software especificados.

5.1.3. Gerenciamento do Processo de Teste

Atividades de teste conduzidas em diferentes níveis (consulte o tópico 2, Níveis de Teste) devem ser organizadas - em conjunto com pessoas, ferramentas, políticas e medidas - em um processo bem definido isso é parte integrante do ciclo de vida.

5.1.4. Documentação de Teste e Produtos de Trabalho

A documentação é parte integrante da formalização do processo de teste[18]. Os documentos de teste podem incluir, entre outros, o plano de teste, especificação de projeto de teste, especificação de procedimento de teste, especificação de caso de teste, registro de teste e relatório de incidente de teste. O software em teste é documentado como o item de teste. A documentação de teste deve ser produzida e atualizada continuamente com o mesmo nível de qualidade que outros tipos de documentação em engenharia de software. A documentação de teste também deve estar sob o controle do gerenciamento de configuração de software (consulte o capítulo 6 - Software Configuration Management do SWEBOK). Além disso, a documentação de teste inclui produtos de trabalho que podem fornecer material para manuais do usuário e treinamento do usuário.

[18] ISO/IEC/IEEE P29119-1/DIS Draft Standard for Software and Systems Engineering - Software Testing

5.1.5. Desenvolvimento Orientado a Testes

O desenvolvimento orientado a testes (TDD) originou-se como uma das principais práticas de XP (programação extrema) e consiste em escrever testes unitários antes de escrever o código a ser testado (veja sobre os Métodos Ágeis no Software no capítulo 9 do SWEBOK). Dessa forma, o TDD desenvolve os casos de teste como um substituto para um documento de especificação de requisitos de software, em vez de uma verificação independente de que o software implementou corretamente os requisitos.

Em vez de uma estratégia de teste, o TDD é uma prática que exige que os desenvolvedores de software definam e mantenham testes de unidade; Assim, também pode ter um impacto positivo na elaboração das necessidades do usuário e das especificações de requisitos de software.

5.1.6. Equipe de teste interna versus independente

A formalização do processo de teste também pode envolver a formalização da organização da equipe de teste. A equipe de testes pode ser composta de membros internos (ou seja, da equipe do projeto, envolvidos ou não na construção de software), de membros externos (na esperança de trazer uma perspectiva imparcial e independente) ou de membros internos e externos. Considerações de custo, cronograma, níveis de maturidade das organizações envolvidas e criticidade do aplicativo podem orientar a decisão.

5.1.7. Estimativa de Custo / Esforço e Medidas de Processo de Teste

Diversas medidas relacionadas aos recursos gastos em testes, bem como à eficácia relativa de detecção de falhas das várias fases de teste, são usadas pelos gerentes para controlar e melhorar o processo de teste. Essas medidas de teste podem abranger aspectos como o número de casos de teste especificados, o número de casos de teste executados, o número de casos de teste aprovados e o número de casos de teste com falha, entre outros.

A avaliação dos relatórios da fase de teste pode ser combinada com a análise da causa-raiz para avaliar a eficácia do processo de teste em encontrar falhas o mais cedo possível. Tal avaliação pode ser associada à análise de riscos. Além disso, os recursos que valem a pena serem gastos em testes devem ser proporcionais ao uso / criticidade da aplicação: diferentes técnicas têm custos diferentes e geram diferentes níveis de confiança na confiabilidade do produto.

5.1.8. Rescisão

Deve ser tomada uma decisão sobre quanto teste é suficiente e quando um estágio de teste pode ser finalizado. Medidas de profundidade, como cobertura de código ou cobertura funcional alcançada, bem como estimativas de densidade de falhas ou de confiabilidade operacional, fornecem suporte útil, mas não são suficientes em si mesmas. A decisão também envolve considerações sobre os custos e riscos incorridos por possíveis falhas remanescentes, em oposição aos custos incorridos por continuar a testar (consulte Critérios de seleção de teste / Critérios de adequação de teste na seção 1.2).

5.1.9. Reutilização de teste e padrões de teste

Para realizar testes ou manutenção de maneira organizada e econômica, os meios usados para testar cada parte do software devem ser reutilizados sistematicamente. Um repositório de materiais de teste deve estar sob o controle do gerenciamento de configuração de software, de modo que as alterações nos requisitos de software ou no projeto possam ser refletidas nas alterações nos testes realizados. As soluções de teste adotadas para testar alguns tipos de aplicativos sob certas circunstâncias, com as motivações por trás das decisões tomadas, formam um padrão de teste que pode ser documentado para posterior reutilização em projetos semelhantes.

5.2. Atividades de teste

Conforme mostrado na descrição a seguir, o gerenciamento bem-sucedido das atividades de teste depende fortemente do processo de gerenciamento de configuração de software (consulte o capítulo 6 - Software Configuration Management do SWEBOK).

5.2.1. Planejamento

Como todos os outros aspectos do gerenciamento de projetos, as atividades de teste devem ser planejadas. Os principais aspectos do planejamento de testes incluem a coordenação de pessoal, a disponibilidade de instalações e equipamentos de teste, a criação e manutenção de toda a documentação relacionada a testes e o planejamento de possíveis resultados indesejáveis. Se mais de uma linha de base do software estiver sendo mantida, uma grande consideração de planejamento é o tempo e o esforço necessários para garantir que o ambiente de teste esteja configurado corretamente.

5.2.2. Geração de Casos de Teste

A geração de casos de teste é baseada no nível de teste a ser realizado e nas técnicas de teste específicas. Os casos de teste devem estar sob o controle do gerenciamento de configuração de software e incluir os resultados esperados para cada teste.

5.2.3. Desenvolvimento de ambiente de teste

O ambiente usado para teste deve ser compatível com as outras ferramentas de engenharia de software adotadas. Deve facilitar o desenvolvimento e o controle de casos de teste, bem como o registro e a recuperação dos resultados esperados, scripts e outros materiais de teste.

5.2.4. Execução

A execução de testes deve incorporar um princípio básico de experimentação científica: tudo o que é feito durante o teste deve ser realizado e documentado com clareza suficiente para que outra pessoa possa replicar os resultados. Portanto, os testes devem ser realizados de acordo com os procedimentos documentados, usando uma versão claramente definida do software em teste.

5.2.5. Avaliação dos Resultados do Teste

Os resultados do teste devem ser avaliados para determinar se o teste foi bem-sucedido ou não. Na maioria dos casos, "bem-sucedido" significa que o software foi executado como esperado e não teve nenhum resultado inesperado importante. Nem todos os resultados inesperados são necessariamente falhas, mas são determinados em algum momento como um ruído. Antes que uma falha possa ser removida, é necessário um esforço de análise e depuração para isolá-lo, identificá-lo e descrevê-lo. Quando os resultados dos testes são particularmente importantes, um comitê de revisão formal pode ser convocado para avaliá-los.

5.2.6. Relatório de Problemas / Registro de Testes

As atividades de teste podem ser inseridas em um registro de testes para identificar quando um teste foi realizado, quem realizou o teste, qual configuração de software foi usada e outras informações de identificação relevantes. Resultados de testes inesperados ou incorretos podem ser registrados em um sistema de relatório de problemas, cujos dados formam a base para depuração e correção posteriores dos problemas que foram observados como falhas durante o teste. Além disso, anomalias não classificadas como falhas poderiam ser documentadas caso posteriormente se tornem mais graves do que se pensava inicialmente. Os relatórios de teste também são entradas para o processo de solicitação de gerenciamento de mudança (consulte Controle de configuração de software no capítulo 6 do SWEBOK que trata do gerenciamento de configuração de software).

5.2.7. Rastreamento de defeitos

Os defeitos podem ser rastreados e analisados para determinar quando eles foram introduzidos no software, por que eles foram criados (por exemplo, requisitos mal definidos, declaração incorreta de variáveis, vazamento de memória, erro de sintaxe de programação) e eles poderiam ter sido observados pela primeira vez no software. As informações de rastreamento

de defeitos são usadas para determinar quais aspectos do teste de software e de outros processos precisam ser aprimorados e como as abordagens anteriores foram eficazes.

6. Ferramentas de Teste de Software

6.1. Suporte à ferramenta de teste

O teste exige muitas tarefas que exigem muito trabalho, executa inúmeras execuções de programas e manipula uma grande quantidade de informações. Ferramentas apropriadas podem aliviar o fardo de operações clericais e tediosas e torná-las menos propensas a erros. Ferramentas sofisticadas podem suportar design de teste e geração de casos de teste, tornando-o mais eficaz.

6.1.1. Seleção de ferramentas

Orientação para gerentes e testadores sobre como selecionar ferramentas de teste que serão mais úteis para sua organização e processos é um tópico muito importante, já que a seleção de ferramentas afeta muito a eficiência e a efetividade do teste. A seleção de ferramentas depende de diversas evidências, como opções de desenvolvimento, objetivos de avaliação, instalações de execução e assim por diante. Em geral, pode não haver uma ferramenta exclusiva que satisfaça necessidades específicas, portanto, um conjunto de ferramentas pode ser uma opção apropriada.

6.2. Categorias de Ferramentas

Categorizamos as ferramentas disponíveis de acordo com sua funcionalidade:

• Testar chamadas (drivers, stubs) prover um ambiente controlado no qual os testes podem ser iniciados e as saídas de teste podem ser registradas. Para executar partes de um programa, drivers e stubs são fornecidos para simular os módulos de chamada e chamado, respectivamente.

• Geradores de teste fornecem assistência nos casos de teste de geração. A geração pode ser aleatória, baseada em caminho, baseada em modelo ou ambos.

• Ferramentas de captura / reprodução

Reexecutam ou repetem automaticamente testes executados anteriormente que gravaram entradas e saídas (por exemplo, telas).

• As ferramentas de verificação de asserção / comparadores de arquivos

Ajudam a decidir se um resultado de teste é bem-sucedido ou não.

• Analisadores de cobertura e instrumentadores

Trabalham juntos. Os analisadores de cobertura avaliam quais e quantas entidades do gráfico de fluxo do programa foram exercidas entre todas as exigidas pelo critério de cobertura de teste selecionado. A análise pode ser feita graças aos instrumentadores do programa que inserem sondas de gravação no código.

• Rastreadores

Registram o histórico dos caminhos de execução de um programa.

• Ferramentas de teste de regressão

Suportam a reexecução de um conjunto de testes após uma seção de software ter sido modificada. Eles também podem ajudar a selecionar um subconjunto de teste de acordo com a alteração feita.

• Ferramentas de avaliação de confiabilidade

Suportam a análise de resultados de testes e a visualização gráfica, a fim de avaliar medidas relacionadas à confiabilidade de acordo com modelos selecionados.

7. Bibliografia

- S. Naik and P. Tripathy, Software Testing and Quality Assurance: Theory and Practice, Wiley-Spektrum, 2008.

I. Sommerville, Software Engineering, 9th ed., Addison-Wesley, 2011.

- M.R. Lyu, ed., Handbook of Software Reliability Engineering, McGraw-Hill and IEEE Computer Society Press, 1996.
- H. Zhu, P.A.V. Hall, and J.H.R. May, "Software Unit Test Coverage and Adequacy," ACM Computing Surveys, vol. 29, no. 4, Dec. 1997, pp. 366–427.
- E.W. Dijkstra, "Notes on Structured Programming," T.H.-Report 70-WSE-03, Technological University, Eindhoven, 1970; http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF.
- ISO/IEC/IEEE P29119-1/DIS Draft Standard for Software and Systems Engineering Software Testing—Part 1: Concepts and Definitions, ISO/IEC/IEEE, 2012.
- ISO/IEC/IEEE 24765:2010 Systems and Software Engineering—Vocabulary, ISO/IEC/IEEE, 2010.
- S. Yoo and M. Harman, "Regression Testing Minimization, Selection and Prioritization: A Survey," Software Testing Verification and Reliability, vol. 22, no. 2, Mar. 2012, pp. 67–120.
- S.H. Kan, Metrics and Models in Software Quality Engineering, 2nd ed., Addison Wesley, 2002.
- J. Nielsen, Usability Engineering, Morgan Kaufmann, 1993.
- T.Y. Chen et al., "Adaptive Random Testing: The ART of Test Case Diversity," Journal of Systems and Software, vol. 83, no. 1, Jan. 2010, pp. 60–66.
- Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," IEEE Trans. Software Engineering, vol. 37, no. 5, Sep.–Oct. 2011, pp. 649–678.
- M. Utting and B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.