**PS-MCTIC**

**Guia Operacional – Políticas de Teste**

**Versão 1.2**

|  |
| --- |
| **Histórico de Revisões** |
| **Versão** | **Data** | **Descrição** | **Autor** |
| 1.0 | 19/01/2015 | Elaboração do Artefato | RSI Informática |
| 1.1 | 28/01/2015 | Inclusão da referência do PS-MCTI | RSI Informática |
| 1.2 | 28/03/2017 | Atualização da sigla MCTI para MCTIC e descrição pela fusão do Ministério das Comunicações. | CTIS |

Sumário

[1. INTRODUÇÃO 5](#_Toc478460030)

[2. FUNDAMENTOS DO TESTE 5](#_Toc478460031)

[2.1 Causa dos Defeitos de Software 5](#_Toc478460032)

[3. NÍVEIS DE TESTE 6](#_Toc478460033)

[3.1. Teste de Componente 6](#_Toc478460034)

[3.2. Teste de Integração 6](#_Toc478460035)

[3.3. Teste de Sistema 8](#_Toc478460036)

[3.4. Teste de Aceite 9](#_Toc478460037)

[4. TIPOS DE TESTE 9](#_Toc478460038)

[4.1. Testes Funcionais 9](#_Toc478460039)

[4.1.1. Testes de Controle Interno e Segurança 10](#_Toc478460040)

[4.2. Testes Não-Funcionais 11](#_Toc478460041)

[4.3. Testes Estruturais 11](#_Toc478460042)

[4.4. Testes relacionados a mudanças (confirmação e regressão) 12](#_Toc478460043)

[5. TÉCNICAS ESTÁTICAS 12](#_Toc478460044)

[5.1. Revisões e o Processo de Teste 13](#_Toc478460045)

[5.2. Processo de Revisão 13](#_Toc478460046)

[5.2.1. Fases de um Processo de Revisão 13](#_Toc478460047)

[5.2.2. Funções e Responsabilidades 15](#_Toc478460048)

[5.2.3. Tipos de Revisão 15](#_Toc478460049)

[5.3. Ferramentas de Análise Estática 16](#_Toc478460050)

[6. Técnicas de Modelagem de Teste 17](#_Toc478460051)

[6.1. Baseado em Especificação (ou baseada em comportamento ou caixa-preta) 17](#_Toc478460052)

[6.1.1. Particionamento de Equivalência 17](#_Toc478460053)

[6.1.2. Análise de Valor Limite 18](#_Toc478460054)

[6.1.3. Tabela de Decisão 19](#_Toc478460055)

[6.1.4. Teste de Transição de Estados 21](#_Toc478460056)

[6.2. Baseado em Estrutura (ou caixa-branca) 22](#_Toc478460057)

[6.2.1. Teste e Cobertura de Comandos 23](#_Toc478460058)

[6.2.2. Teste e Cobertura de Decisão 24](#_Toc478460059)

[6.2.3. Outras Técnicas Baseadas em Estrutura 25](#_Toc478460060)

[6.3. Baseado em Defeitos 26](#_Toc478460061)

[6.4. Baseado em Experiência 26](#_Toc478460062)

[7. Planejamento & Controle de Teste 27](#_Toc478460063)

[8. Modelagem & Análise de Teste 29](#_Toc478460064)

[8.1. Identificação de Condições de Teste 29](#_Toc478460065)

[8.2. Criação de Casos de Teste 30](#_Toc478460066)

[9. Implementação e Execução de Teste 32](#_Toc478460067)

[9.1. Implementação de Teste 32](#_Toc478460068)

[9.2. Execução de Teste 33](#_Toc478460069)

[9.3. Avaliação do Critério de Saída e Relatório 35](#_Toc478460070)

[10. REFERÊNCIAS 36](#_Toc478460071)

# INTRODUÇÃO

Este guia tem por finalidade descrever abordagens, requisitos e procedimentos necessários para as atividades de Teste de Software do Processo de Software (PS-MCTIC).

# FUNDAMENTOS DO TESTE

A grande maioria das pessoas já teve uma experiência com algum software que não funcionou conforme o esperado: um erro em uma fatura, um atraso no processamento de um pagamento realizado com cartão de crédito, um site que não foi carregado corretamente. Todos estes são exemplos comuns que podem acontecer por causa de problemas em softwares.

Nem todos os sistemas / softwares nos trazem o mesmo nível de risco e nem todos os problemas têm o mesmo impacto quando ocorrem. Um risco é algo que não aconteceu ainda e, que pode nunca acontecer. É um problema em potencial. Estamos preocupados com esses problemas em potencial, pois, se um deles acontecer, nós sentiremos um impacto negativo. Quando discutimos os riscos, é preciso considerar qual a possibilidade que o problema ocorra e o impacto gerado caso isso aconteça.

Alguns dos problemas que encontramos quando testamos um software são bastante triviais, mas outros podem ser caros e prejudiciais - com perda de tempo, dinheiro ou de reputação nos negócios - e até resultar em ferimentos ou morte.

## 2.1 Causa dos Defeitos de Software

As pessoas cometem erros - que são as falhas. Se alguém comete um erro ou erra no uso do software, isso pode levar diretamente a um problema - o software é usado incorretamente e, portanto, não se comporta como esperávamos. No entanto, as pessoas também podem cometer erros durante o projeto e construção do software. Esses erros significam que existem falhas no próprio software. Estas são chamadas defeitos, ou ainda, erros ou falhas. Lembre-se que o software não é apenas o código. Quando o código do software foi construído, ele é executado e, em seguida, todos os defeitos podem fazer com que o sistema deixe de fazer o que deve fazer (ou fazer algo que não deve), causando uma falha. Nem todos os defeitos resultam em falhas, alguns defeitos ficam dormentes no código, podendo nunca serem observados.

Os erros humanos podem causar defeitos ou falhas a serem introduzidos em qualquer fase do ciclo de vida de desenvolvimento de software e, dependendo das conseqüências do erro, os resultados podem ser insignificantes ou catastróficos. Testes rigorosos são necessários durante o desenvolvimento e manutenção para identificar defeitos, a fim de reduzir as falhas. Execução de testes nos ajuda a avançar para a melhoria da qualidade do produto e serviço, mas isso é apenas um dos métodos de verificação e validação aplicados aos produtos. Uma variedade de métodos e técnicas pode ser utilizada.

# NÍVEIS DE TESTE

1.

## Teste de Componente

O Teste de Componente, também conhecido como teste unitário, de módulo ou de programa, pesquisa por defeitos no componente e verifica o funcionamento do software (por exemplo, módulos, programas, objetos, classes, etc) nos itens que são separadamente testáveis.

Teste de componente pode ser feito de forma isolada do resto do sistema, dependendo no contexto do ciclo de vida do desenvolvimento e do sistema. Na maioria das vezes stubs e os drivers são usados para substituir o software que está faltando e simular a interface entre os componentes de software de maneira simples. Um stub é chamado a partir do componente de software a ser testado; um driver chama um componente a ser testado.

Teste de componente pode incluir testes funcionais e não funcionais específicos para algumas características, tais como, o comportamento do recurso (vazamentos de memória, por exemplo), desempenho ou testes de resistência, bem como testes estruturais (por exemplo, decisão, cobertura). Os casos de teste são derivados de produtos de trabalho, como o desenho do software ou modelo de dados.

Normalmente, testes de componentes ocorrem com o acesso ao código que está sendo testado e com o apoio do ambiente de desenvolvimento. Na prática, geralmente envolve o programador que escreveu o código. Às vezes, dependendo do nível de risco aplicável, o teste de componente é realizado por diferentes programadores, introduzindo assim a independência.

Os defeitos normalmente são corrigidos logo que são encontrados, sem registro formal dos incidentes.

## Teste de Integração

O Teste de Integração pode referir-se tanto a integração entre os componentes, bem como interações com diferentes partes de um sistema, como um arquivo do sistema operacional, com o hardware ou, até mesmo, interfaces entre diversos sistemas. O teste de integração deve ser diferenciado das atividades de integração. O teste de integração é normalmente realizado pelos desenvolvedores, mas também pode ser realizado por um testador ou uma equipe independente de testes.

Pode haver mais de um nível de testes de integração e estes podem ser realizados em objetos de teste de tamanhos variáveis. Por exemplo:

* Testes de integração de componentes testam a interação entre componentes do software e são realizados após o teste de componente;
* Teste de integração de sistema testa a interação entre diferentes sistemas e pode ser realizado depois dos testes do sistema.

Quanto maior o escopo da integração, mais difícil torna-se isolar falhas de uma interface específica, o que pode levar a um maior risco. Isso leva à abordagens diferentes em relação ao teste de integração. Um extremo é que todos os componentes ou os sistemas são integrados simultaneamente, após a qual tudo é testado como um todo. Isso é chamado de teste de integração "big bang". Testes Big-bang tem a vantagem de que tudo está acabado antes do início do teste de integração. Não há necessidade de simular peças. A principal desvantagem é que, em geral, é demorado e difícil de identificar a causa de falhas com esta integração tardia. Assim, a integração big-bang pode parecer uma boa idéia quando o planejamento do projeto é otimista e não espera encontrar nenhum problema. Se achar que o teste de integração vai encontrar defeitos, é uma boa prática a considerar que, pode-se ganhar tempo se o processo de integração for feito em partes menores.

Outra forma de testar integração seria integrar todos os programas, um por um, e realizar um teste após cada etapa (teste incremental). Entre esses dois extremos, há uma gama de variantes. A abordagem incremental tem a vantagem de ter os defeitos encontrados no início de uma pequena montagem, quando é relativamente fácil detectar a causa. Uma desvantagem é que pode ser demorado, pois stubs e drivers têm de ser desenvolvidos e utilizados no teste. Dentro dos testes de integração incremental, existem algumas possibilidades, dependendo do sistema arquitetura:

* Top-down: o teste ocorre de cima para baixo, seguindo o fluxo de controle ou estrutura arquitetônica (por exemplo, a partir da GUI ou do menu principal). Componentes ou sistemas são substituídos por stubs.
* Bottom-up: o teste ocorre de baixo para cima do controle de fluxo. Componentes ou sistemas são substituídos por drivers.
* Funcional Incremental (backbone): a integração e os testes ocorrem com base nas funções ou funcionalidades, como documentadas na especificação funcional.

A sequência de integração preferencial e o número de etapas de integração necessárias dependem da localização da arquitetura das interfaces de alto risco. A melhor opção é começar a integração com as interfaces em que são esperados mais problemas. Isso impede defeitos maiores no final do teste de integração. A fim de reduzir o risco da descoberta tardia de defeitos, a integração, preferencialmente, deve ser incremental em vez de "big-bang". Idealmente, testadores devem compreender a arquitetura e sua influência no planejamento de integração. Se os testes de integração estão previstos antes que os componentes ou sistemas estejam construídos, podem ser desenvolvidos, na ordem necessária para a maioria dos testes eficientes.

Em cada fase da integração, os testadores se concentram apenas na integração em si, seja ela qual for. O teste de integração pode ser realizado por os desenvolvedores, mas também, pode ser feito por uma equipe independente de testadores especialistas em integração, ou por um grupo de especialistas desenvolvedores / integradores, incluindo especialistas não funcionais.

## Teste de Sistema

O teste de sistema está relacionado ao comportamento de todo o sistema / produto definido pelo escopo de um projeto de desenvolvimento ou produto. Podem incluir testes baseados em riscos e / ou especificação de requisitos, processos de negócios, casos de uso, etc. O teste do sistema é o mais frequentemente utilizado para verificar se o sistema a ser entregue atende às especificações. O seu objetivo é encontrar tantos defeitos quanto for possível. Na maioria das vezes, é realizado por testadores especializados, dentro da própria equipe de desenvolvimento ou através de equipes de teste independentes.

O Teste de Sistema deve investigar tanto os requisitos funcionais quanto os não-funcionais. Testes não-funcionais típicos incluem desempenho, segurança, confiabilidade, etc.

O Teste de Sistema baseia-se na especificação dos sistemas e é realizado utilizando a técnica de teste baseada em especificação (caixa preta) que for mais adequada. Por exemplo, uma tabela de decisão pode ser criada para as combinações dos efeitos descritos em regras de negócio. Técnicas de Testes Estruturais (caixa branca) também podem ser utilizadas para avaliar a profundidade dos elementos desses testes.

O teste do sistema requer um ambiente de teste controlado que respeite, entre outras coisas, o controle de versões do software, o testware e os dados de teste.

## Teste de Aceite

Após a realização dos testes de sistema, tendo-se corrigido todos, ou pelo menos, a maioria dos defeitos inicia-se o Teste de Aceite. O teste de aceite deve responder a perguntas tais como: “O sistema pode ser liberado?”, “Ainda existem riscos?” e “O time de desenvolvimento cumpriu com suas tarefas?”. O teste de aceite é responsabilidade do usuário ou cliente, que podem solicitar o auxílio da equipe de testes, embora outras partes interessadas possam estar envolvidas também. O objetivo do teste de aceite é o de estabelecer a confiança no sistema, parte do sistema ou características não funcionais.

Para que o teste de aceite seja realizado de forma apropriada, o ambiente de testes deve ser o mais próximo possível do ambiente de produção.

# TIPOS DE TESTE

Tipos de testes são apresentados como um meio de definir claramente o objetivo de um nível de teste para um programa ou projeto. Precisamos pensar sobre diferentes tipos de testes, pois testar a funcionalidade do componente ou sistema pode não ser suficiente em cada nível para atender os objetivos do teste global. Um tipo de teste é focado em um objetivo do teste particular, que poderia ser o teste de uma função a ser executada pelo componente ou sistema, uma característica não funcional, a estrutura ou a arquitetura do componente ou sistema, ou relacionados a mudanças, ou seja, confirmando que os defeitos foram corrigidos (teste de confirmação, ou re-teste) e à procura de alterações não intencionais (testes de regressão). Dependendo dos seus objetivos, o teste será organizado de forma diferente.

1.

## Testes Funcionais

A função de um sistema (ou componente) é “o que ele faz”. Isto é tipicamente descrito em uma especificação de requisitos, especificação funcional, ou em casos de uso. Pode haver algumas funções que não estão documentadas, mas que também fazem parte da exigência de um sistema, embora seja difícil de testar em relação aos requisitos não documentados e implícitos.

Testes funcionais baseiam-se nestas funções, descritas em documentos ou compreendidas pelos testadores e podem ser realizadas em todos os níveis de teste (por exemplo, testes de componentes podem ser baseados em uma especificação de componentes).

Testes funcionais consideram o comportamento especificado e são também muitas vezes referidos como testes de caixa-preta. Isso não é inteiramente verdadeiro, pois testes de caixa-preta também incluem testes não-funcionais.

Devemos priorizar os requisitos com base em critérios de risco (se isso não for feito já na especificação) e usar isso para priorizar os testes. Isso irá garantir que os testes mais importantes e mais críticos estão incluídos no esforço de teste.

### Testes de Controle Interno e Segurança

Uma questão-chave para teste de software é o teste de controle interno. A segurança é um componente do controle interno que merece uma atenção especial dos testadores. Interesses em controle interno tem sido destacados por penetrações de segurança divulgadas e o aumento da importância dos sistemas de informação e dos dados contidos nesses sistemas.

Os softwares, geralmente, são controlados por um sistema de controle embutido neles próprios. Desde que o software incorpore controles, os testadores devem testar os controles existentes e executar conforme especificado.

Os objetivos dos controles de transações são prevenir, detectar ou corrigir processamentos incorretos.

As melhores práticas para testar os controles em um software envolvem duas tarefas:

* Realizar avaliação de risco.
* Testar o controle dos processamentos das transações.

Testadores não são especialistas em segurança. No entanto, a segurança é tão importante para as organizações que os testadores não podem ignorar. As tarefas de teste a seguir podem agregar valor às atividades do testador:

1. Entender os pontos onde a segurança é mais freqüentemente vazada, e compreender a diferença entre a perda acidental e intencional.
2. Criar uma matriz de ponto de penetração para identificar as vulnerabilidades do sistema e então, investigar a adequação dos controles de segurança no ponto de maior penetração potencial. Essa matriz ajuda a identificar vulnerabilidades de segurança. A criação de uma matriz de ponto de penetração responde às perguntas sobre onde a segurança é necessária e se os controles de segurança existem nos pontos de maior probabilidade de penetração.
3. Avaliar o programa de treinamento de conscientização de segurança para garantir que os interessados na segurança estão conscientes das suas responsabilidades de segurança.
4. Compreender os atributos de um controle de segurança eficaz.
5. Entender o processo de seleção de técnicas para testar a segurança.

## Testes Não-Funcionais

Um segundo alvo para o teste é o teste das características de qualidade, ou atributos não-funcionais do sistema (ou componente ou grupo de integração). Aqui estamos interessados em quão bem ou quão rápido algo seja feito. Estamos testando algo que precisamos para medir em uma escala de medição, por exemplo, tempo para responder.

Testes não-funcionais, assim como testes funcionais, são realizados em todos os níveis de teste. Testes não-funcionais incluem testes de desempenho, carga, estresse, usabilidade, manutenção, confiabilidade e portabilidade.

O *International Organization for Standardization* (ISO) definiu um conjunto de características de qualidade [ISO / IEC 9126, 2001]. Este conjunto reflete um grande passo para um consenso na indústria de TI e, assim, aborda a noção geral de qualidade de software. O padrão ISO 9126 define seis características de qualidade:

* Funcionalidade;
* Confiabilidade;
* Usabilidade;
* Eficiência;
* Manutenibilidade;
* Portabilidade.

## Testes Estruturais

Teste estrutural é muitas vezes referido como “caixa branca”, porque nós estamos interessados no que está acontecendo “dentro da caixa”. Teste estrutural é o mais frequentemente usado como uma forma de medir o rigor de testes pela cobertura de um conjunto de elementos estruturais (comandos e decisões, uma hierarquia chamada, estrutura de menus).

Análise da Cobertura é a medição da cobertura alcançada por um item de cobertura específico durante a execução do teste com relação aos critérios pré-determinados feita para determinar se há necessidade de se fazer teste adicional; e, em caso positivo, quais casos de teste serão necessários.

Se a cobertura não atinge 100%, então mais testes devem ser construídos a fim de testar aqueles itens que não foram contemplados para, desta forma, aumentar a cobertura.

Pode ocorrer em qualquer nível de teste, embora seja verdade dizer que ele tende a ser aplicado principalmente nos testes de componentes e integração.

## Testes relacionados a mudanças (confirmação e regressão)

Quando um teste falha e nós determinamos que a causa é um defeito, este deve ser relatado e, após sua correção, será necessário executar o teste novamente para confirmar se o defeito foi realmente corrigido. Isto é conhecido como teste de confirmação (também chamado de re-teste). Ao fazer testes de confirmação, é importante garantir que o teste é executado exatamente da mesma maneira como foi realizado a primeira vez, usando as mesmas entradas de dados e ambientes.

A realização de manutenções, sejam elas corretivas ou evolutivas, pode introduzir defeitos em partes do software que haviam sido testadas anteriormente sem falhas. O caminho para detectar esses “efeitos colaterais inesperados” é realizar testes de regressão.

Assim como nos testes de confirmação, os testes de regressão envolvem casos de teste que já foram executadas antes. A diferença é que, para testes de regressão, os casos de teste, provavelmente, foram executados com sucesso na última vez em que eles foram exercitados.

Mais especificamente, a finalidade dos testes de regressão é verificar que as modificações no software ou no ambiente não tenham causado efeitos adversos não intencionais e que o sistema ainda corresponde às suas exigências.

É apropriado ter um conjunto de testes de regressão para todos os níveis de testes (componente, integração, sistema, etc.) Tanto podemos reexecutar todos os testes realizados previamente como também, analisar a real necessidade de regressão e reduzir o escopo deste teste, utilizando para isso, análises de risco e impacto.

# TÉCNICAS ESTÁTICAS

O objetivo fundamental do teste estático é melhorar a qualidade do software e dos produtos de trabalho, auxiliando os envolvidos a reconhecer e corrigir seus próprios defeitos no início do processo de desenvolvimento de software. Embora as técnicas de testes estáticos não resolvam todos os problemas, elas são muito eficazes. Técnicas estáticas podem melhorar tanto a qualidade quanto a produtividade e não devem ser consideradas como substitutas para o teste dinâmico, mas sim com um complemento em todos os aspectos principais de seu trabalho, incluindo requisitos, projeto, implementação, teste e manutenção.

1.

## Revisões e o Processo de Teste

Durante os testes estáticos, os produtos de trabalho são examinados manualmente, ou com um conjunto de ferramentas, mas não são executados.

Testes dinâmicos e testes estáticos são métodos complementares, e tendem a encontrar diferentes tipos de defeitos de forma eficaz e eficiente. Tipos de defeitos que são mais fáceis de encontrar durante os testes estáticos são: o desvio de normas falta de requisitos, defeitos de projeto, código não sustentável e especificações de interface inconsistentes.

Note-se que, em contraste com o teste dinâmico, testes estáticos encontram defeitos, ao invés de falhas.

Estudos têm demonstrado que, como resultado de revisões, um aumento significativo na produtividade e qualidade do produto pode ser alcançado, reduzindo o número de defeitos no início do ciclo de vida do produto também significa que menos tempo tem de ser gasto em testes e manutenção.

## Processo de Revisão

Um processo de revisão pode variar de muito informal até formal (ou seja, bem estruturada e regulamentada). A formalidade de um processo de revisão é relacionada a fatores como a maturidade do processo de desenvolvimento, qualquer questão legal, requisitos regulamentares ou a necessidade de uma trilha de auditoria. Na prática, a revisão informal é talvez o tipo mais comum de análise. Opiniões informais são aplicadas em vários momentos durante as fases iniciais do ciclo de vida de um documento. Uma equipe de duas pessoas pode conduzir uma revisão informal, como o autor pode pedir a um colega a revisão de um documento ou código. Em fases posteriores essas opiniões muitas vezes envolvem mais pessoas e uma reunião. Isso normalmente envolve colegas do autor, que tentam encontrar defeitos no documento em análise e discutir esses defeitos em uma reunião de revisão. O objetivo é ajudar o autor e melhorar a qualidade do documento.

### Fases de um Processo de Revisão

Geralmente, um processo de revisão típico consiste em seis etapas principais:

**1 - Planejamento:** Fase em que se deve selecionar a equipe, alocar as funções, definir os critérios de entrada e de saída para os diversos tipos de revisão formal (ex: inspeção), e selecionar quais as partes dos documentos serão vistos. Durante o planejamento geral, a gerência deve decidir quais documentos estarão sujeitos à revisão. No planejamento de cada revisão, o líder da revisão seleciona participantes competentes para formar a equipe de revisão. Juntamente com o autor, a documentação deve ser avaliada para verificar se está pronta para a revisão. Em revisões formais, devem ser avaliados os critérios de entrada e saída. Para a revisão, o melhor é usar várias perspectivas e diferentes pontos de vista. Pode também ser decidido não analisar o documento inteiro, somente algumas amostragens para verificar a qualidade geral do documento.

**2 - Kick-off:** distribuir os documentos, explicar os objetivos, processos e documentos para os participantes; e checar os critérios de entrada (para os diversos tipos de revisão). Esse pontapé inicial serve para fornecer as informações necessárias a todos os envolvidos na revisão. Além de compartilhar informações relativas ao documento e à revisão, se os envolvidos não tiverem familiaridade com o domínio da aplicação, uma descrição da aplicação ou do ambiente também será necessária. Juntamente com os documentos que serão revisados, documentos de apoio devem ser distribuídos (para os participantes saberem o que é certo ou errado); essa documentação pode ser chamada de documentação de base ou linha de base.

**3 - Preparação:** trabalho feito por cada participante antes da reunião de revisão, tomando nota dos defeitos em potenciais, questões e comentários.

**4 - Reunião de Revisão:** discussão ou registro, com resultados documentados ou anotações (para os tipos de revisões mais formais). Os participantes da reunião podem simplesmente anotar os defeitos, fazer recomendações para o tratamento de defeitos ou tomar decisões sobre os defeitos. Na norma IEEE 1028 é chamado de exame. A reunião de revisão é conduzida por um líder ou um moderador; esse papel requer grande habilidade para proteger os participantes e motivá-los a contribuir com a revisão. Normalmente, essas reuniões têm um tempo limite. Seu objetivo é decidir se o objeto sendo revisado satisfaz os requisitos e está de acordo com os padrões, bem como encontrar defeitos. O resultado é uma recomendação de aceite, reparo, ou reescrita do documento.

**5 - Retrabalho / Reparo:** Resolver defeitos encontrados, tipicamente feitos pelo autor. É decidido como seguir as recomendações ou outras abordagens indicadas na reunião de revisão.

**6 - Acompanhamento:** Checar se os defeitos foram encaminhados, obtendo métricas e checando o critério de saída (para tipos de revisões formais). Geralmente feito pelo gerente, moderador ou alguém designado especialmente para o acompanhamento da correção dos defeitos. Se o resultado da reunião foi um não aceite, outra reunião deverá ser programada.

### Funções e Responsabilidades

Os participantes em qualquer tipo de revisão formal devem possuir conhecimentos adequados do processo de revisão. Dentro de uma equipe de revisão, quatro tipos de participantes podem ser distinguidos: o moderador, o autor, o revisor e o redator.

* **Gerente:** toma decisão durante a realização da revisão, aloca tempo nos cronogramas de projeto e determina se o objetivo da revisão foi atingido. Seleciona os objetos a serem revisados, e confirma que os documentos de base estão disponíveis. Escolhe os participantes da revisão.
* **Moderador:** a pessoa que lidera a revisão do documento ou conjunto de documentos, incluindo o planejamento da revisão, e o acompanhamento após a reunião. Se necessário, o moderador mediará entre os vários pontos de vista e é muitas vezes quem responderá pelo sucesso da revisão. O moderador é crucial para o sucesso da revisão; deve ser um bom líder, conduzindo a reunião de forma eficiente e diplomática. Não deve expressar sua própria opinião sobre o objeto de revisão.
* **Autor:** é a pessoa que escreveu ou que possui a responsabilidade pelos documentos que serão revisados. Se mais de uma pessoa estava envolvida no processo de criação, então uma pessoa será apontada para assumir a responsabilidade. É importante que o autor não tome as questões levantadas como críticas pessoais.
* **Revisores:** indivíduos com conhecimento técnico ou de negócio (também chamados inspetores), que, após a preparação necessária, identificam e descrevem os defeitos encontrados no produto sob revisão. Revisores podem ser escolhidos para representar diferentes funções e perspectivas no processo de revisão, e é parte integrante de qualquer reunião de revisão. Em geral, são alguns (no máximo cinco) especialistas técnicos. É necessário que eles realizem a preparação individual antes da reunião de revisão.
* **Secretário (ou redator):** documenta todo o conteúdo da reunião, problemas e itens em aberto que foram identificados durante a reunião.

### Tipos de Revisão

As revisões podem ser divididas em quatro tipos específicos:

1. **Informal**: nenhum processo formal é usado (gráficos no corredor, testes camaradas, programação por par), mesmo assim útil, barata, popular.
2. **Técnica**: processo documentado e definido de remoção de defeitos, envolvendo pares e especialistas técnicos, mas não gerentes.
3. **Acompanhamento** (*walkthroughs*): o autor “conduz” seus pares “através” do documento ou código.
4. **Inspeções**: um moderador treinado (que não seja o autor) lidera a equipe de inspeção (com papéis definidos) através de um processo formal de inspeção (regras, listas de checagem, critérios de entrada e saída), que inclui obtenção de métricas de remoção de defeitos.

## Ferramentas de Análise Estática

Existe um tipo diferente de teste estático, onde examinamos cuidadosamente requisitos, diagramas e códigos, que geralmente é realizado com o apoio automatizado para descobrir defeitos adicionais antes que o código seja realmente executado. Assim, o que é chamado de análise estática é apenas outra forma de teste.

Análise estática difere do teste dinâmico em uma série de maneiras importantes:

* A análise estática é realizada em requisitos de projeto, ou código sem realmente executar o artefato, ou software a ser examinado.
* A análise estática é idealmente realizada antes dos tipos mais formais de revisão.
* A análise estática não está relacionada a propriedades dinâmicas dos requisitos e código, como a cobertura de teste.
* Análises estáticas encontram defeitos ao invés de falhas.

Para análise estática existem muitas ferramentas, sendo que a maioria delas se concentra nos códigos. Ferramentas de análise estática são geralmente usadas por desenvolvedores antes e, às vezes, durante os testes de componente e de integração, e por designers durante a modelagem de software.

Em resumo, o maior valor da análise estática refere-se especialmente a:

* Detecção precoce de defeitos antes da execução do teste;
* Alerta precoce sobre desvios em aspectos predefinidos do código ou design;
* Identificação de defeitos que não são facilmente encontrados em testes dinâmicos;
* Melhoria na manutenção do código e design de acordo com padrões documentados e regras;
* Prevenção de defeitos, desde que os responsáveis estejam dispostos a aprender com os seus erros, aplicando melhoria contínua.

# Técnicas de Modelagem de Teste

Antes que possamos realmente realizar um teste, nós precisamos saber o que estamos tentando testar, as entradas, os resultados que devem ser produzidos por essas entradas, e como nós realmente devemos nos preparar para executar os testes.

Existem algumas técnicas que nos auxiliam na transformação das expectativas em torno da qualidade em casos e procedimentos de teste relevantes para o projeto. Estas técnicas serão apresentadas nos tópicos a seguir.

1.

## Baseado em Especificação (ou baseada em comportamento ou caixa-preta)

As primeiras técnicas de teste dinâmico que trataremos são as técnicas baseadas em especificação, ou técnicas de caixa-preta, que são assim chamadas por verem o software como uma caixa preta com entradas e saídas, mas sem conhecimento de como o sistema ou componente está estruturado dentro da caixa. Em essência, o testador está se concentrando sobre o que o software faz e não como ele faz isso.

Nesse momento cobriremos as quatro técnicas baseadas em especificações que são mais utilizadas e difundidas:

* Particionamento de Equivalência;
* Análise de Valor Limite;
* Tabelas de Decisão;
* Gráficos de Transição de Estados.

Existem ainda outras técnicas, que poderão vir a ser contempladas neste documento em um momento futuro.

### Particionamento de Equivalência

Particionamento de equivalência é uma técnica de caixa-preta que pode ser aplicada em qualquer nível de teste e é muitas vezes uma boa técnica para ser a primeira a ser utilizada. É uma abordagem de senso comum para testes, tanto que a maioria dos testadores pratica, informalmente, mesmo que não tenha consciência disso.

A ideia por trás da técnica é dividir (ou seja, a particionar) um conjunto de condições de teste em grupos ou conjuntos que podem ser considerados semelhantes (ou seja, o sistema deve lidar com eles de forma equivalente), portanto, "particionamento de equivalência". Partições de equivalência são também conhecidos como classes de equivalência - os dois termos significam exatamente a mesma coisa.

A técnica de particionamento de equivalência diz então que precisamos testar apenas uma condição de cada partição. Isto é porque estamos supondo que todas as condições em uma partição serão tratadas da mesma forma pelo software.

As partições são dividas em válidas e inválidas. As partições válidas representam os resultados que são esperados como aceitos para cada grupo, ou divisão. Já as partições inválidas, representariam os desvios ou, colocando de outra forma, os valores que não seriam esperados para o grupo em questão.

O software, geralmente, deve lidar corretamente com os valores da partição inválida, respondendo com uma mensagem de erro.

### Análise de Valor Limite

Análise de valor limite é baseada em testes nos limites localizados entre as partições. Se você já fez uma “verificação de intervalo”, você provavelmente estava utilizando a técnica de análise de valor limite, mesmo se você não estivesse ciente disso. Note que nós temos tanto limites válidos (nas partições válidas) como limites inválidos (nas partições inválidas).

Como exemplo, considere uma impressora que tem uma opção de entrada do número de cópias a serem feitas, 1-99. Ao aplicar a análise de valor limite, vamos utilizar os valores mínimo e o máximo (Limites) a partir da partição válida (1 e 99 neste caso), juntamente com o primeiro valor ou o último, respectivamente, em cada uma das partições inválidas adjacentes à partição válida (0 e 100 neste caso). Neste exemplo, teríamos três testes para particionamento de equivalência (um para cada uma das três partições) e quatro testes para os valores limite.

Também podemos aplicar as técnicas de particionamento de equivalência em outras coisas que não tratem diretamente de números como, por exemplo, uma reserva de vôo, na qual você tem uma escolha entre classe Econômica, Executiva ou Primeira Classe. Cada uma destas opções é uma partição de equivalência, e deve ser testada, mas não faz sentido falar de limites para este tipo de partição. A partição inválida seria uma tentativa de digitar qualquer outro tipo de classe de vôo (por exemplo, “Pessoal”).

Podemos aplicar o particionamento de equivalência e análise de valor limite para todos os níveis de teste. No nível de sistema, por exemplo, podemos ter três configurações básicas que nossos usuários podem escolher ao configurar seus sistemas, com um número de opções para cada configuração. As configurações básicas do sistema poderiam ser administrador, gestor e de ligação do cliente. Estes representam três partições de equivalência que poderiam ser testadas. Nós poderíamos ter sérios problemas se esquecêssemos de testar a configuração para o administrador do sistema, por exemplo. Podemos também aplicar o particionamento de equivalência e análise de valor limite mais de uma vez para o item de mesma especificação.

Partições também podem ser identificadas durante a configuração de dados de teste. Se existem diferentes tipos de registro, o teste será mais representativo se você incluir um registro de dados de cada tipo. O tamanho de um registro é também uma partição com limites, para que pudéssemos incluir registros de tamanho máximo e mínimo na base de dados de teste.

Tendo identificado as condições que você deseja testar, neste caso, usando particionamento de equivalência e análise de valor limite, o próximo passo é projetar os casos de teste. Quanto mais condições de teste que puderem ser cobertas em um único caso de teste, menor o número de casos de teste que serão necessários para cobrir todas as condições. Isto é geralmente a melhor abordagem a adotar para os testes positivos e para os testes que você está razoavelmente confiante que vão passar.

### Tabela de Decisão

As técnicas de particionamento de equivalência e análise de valor limite são frequentemente aplicadas a situações específicas ou insumos. No entanto, se diferentes combinações de insumos resultarem em ações diferentes das que estão sendo tomadas, isso pode ser mais difícil mostrar usando o particionamento de equivalência e análise de valor limite, que tendem a ser mais focadas na interface do usuário. As outras duas técnicas baseadas em especificação, tabelas de decisão e testes de transição de estado são mais focadas em negócios.

Uma tabela de decisão é uma boa maneira de lidar com combinações de coisas (por exemplo, entradas). Esta técnica é por vezes também referida como uma tabela de "causa-efeito". A razão para isto é que existe uma técnica lógica associada à diagramação chamada de "gráfico de causa-efeito", que às vezes é usada para ajudar a criar a tabela de decisão.

Testar combinações pode ser um desafio, como o número de combinações muitas vezes pode ser enorme. Testar todas as combinações pode ser impraticável, se não impossível. Temos que estar satisfeitos com o teste de apenas um pequeno subconjunto de combinações, mas fazer a escolha de quais combinações testar e o que deixar de fora é não trivial.

É uma técnica que funciona bem em conjunto com o particionamento de equivalência. A combinação de condições pode ser explorada em combinações de partições de equivalência.

A primeira tarefa é identificar uma função adequada ou subsistema que tem um comportamento que reage de acordo com uma combinação de insumos ou eventos. Depois de ter identificado os aspectos que precisam ser combinados, então você coloca-os em uma tabela listando todas as combinações verdadeiras e falsas para cada um dos aspectos. Tomemos como mostra a lógica de negócio para um caixa eletrônico:

* Note que os traços “-” indicam condições que não serão atingidas como parte desta regra. As regras são mutuamente exclusivas, e somente aquela única regra pode ser aplicada em qualquer momento único de tempo. Note que a camada de lógica de negócio está usualmente sob a camada de interface com o usuário, portanto neste ponto a checagem básica da sanidade das entradas deveria ter sido feita.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Condição*** | 1 | 2 | 3 | 4 | 5 |
| Cartão válido | N | S | S | S | S |
| Senha válida | - | N | N | S | S |
| Senha inválida=3 | - | N | S | N | N |
| Saldo OK | - | - | - | N | S |
|  |  |  |  |  |  |
| ***Ação*** |  |  |  |  |  |
| Rejeita cartão | S | N | N | N | N |
| Redigita senha | N | S | N | N | N |
| Prende cartão | N | N | S | N | N |
| Redigita opção | N | N | N | S | N |
| Fornece dinheiro | N | N | N | N | S |

Uma tabela de decisão apresenta duas partes. Na metade superior são listadas as entradas (causas); na metade inferior são listados os efeitos. Cada coluna é um caso de teste, ou seja, a combinação de condições e dos efeitos esperados ou das saídas. As condições podem influenciar e excluir umas às outras, fazendo com que nem todas as combinações sejam plausíveis.

Para nosso exemplo então, precisaríamos construir 05 (cinco) casos de teste para cobrir as condições do negócio.

### Teste de Transição de Estados

Teste de transição de estados é usado quando algum aspecto do sistema pode ser descrito no que é chamado de “modelo de estado finito”. Isto significa simplesmente que o sistema pode ter um número finito de estados diferentes, e que as transições de um estado para outro são determinadas pelas regras do modelo. Este é o modelo em que o sistema e os testes se baseiam. Um sistema de estados finitos é frequentemente mostrado como um diagrama de estado

Um modelo de transição de estado tem quatro partes básicas:

* Os estados que o software pode ocupar;
* As transições de um estado para outro;
* Os eventos que causam uma transição;
* As ações que resultam de uma transição.

Em qualquer estado dado, um evento pode causar apenas uma ação, mas que o mesmo evento - a partir de um estado diferente - pode causar uma ação diferente e um diferente estado final.

Nos diagramas, os estados são mostrados como círculos, as transições como linhas com setas e os eventos como texto perto das transições.

Uma das vantagens da técnica de transição de estado é que o modelo pode ser tão detalhado ou tão abstrato quanto você precise que ele seja. Quando uma parte do sistema é mais importante (isto é, necessita de mais testes) o diagrama pode ser modelo com uma profundidade maior de detalhes. Onde o sistema é menos importante (requer menos testes), o modelo pode usar um único estado para significar o que seria uma série de diferentes Estados.

Além dos Diagramas de Transição de Estados, possuímos também Tabelas de Transição de Estados.

Testes para decorrentes apenas a partir de um gráfico de estado são muito bons para ver as transições válidas, mas nós não podemos ver facilmente os testes negativos, onde tentamos gerar transições inválidas. Para ver o número total de combinações de estados e transições, válidos e inválidos, uma tabela de estado é útil.

A tabela de estado lista todos os estados e todos os eventos que as transições causam ao longo do topo (ou vice-versa).

|  |  |  |  |
| --- | --- | --- | --- |
| **Estado atual** | **Evento [Condição]** | **Ação** | **Estado novo** |
| Dentro do sistema | Senha [inválida] | Erro | Dentro do sistema |
| Dentro do sistema | Senha [válida] | Abre registro | Esperando |
| Dentro do sistema | Cliente | [Indefinido] | [Indefinido] |
| Dentro do sistema | Busca [qualquer] | [Indefinido] | [Indefinido] |
| Dentro do sistema | Pagamento [qualquer] | [Indefinido] | [Indefinido] |
| Dentro do sistema | Finaliza | [Indefinido] | [Indefinido] |

Uma tabela de transição de estado pode representar transições de estado complexas que não caberiam num gráfico. Entretanto, a complexidade pode indicar projeto mal feito. Ela também pode revelar uma situação indefinida, como faz esta porção da tabela para o gráfico do exemplo anterior.

## Baseado em Estrutura (ou caixa-branca)

Técnicas de teste baseadas na estrutura (que também são dinâmicas e não estáticas) utilizam a estrutura interna do software para derivar casos de teste. Elas são comumente chamadas de “caixa branca” uma vez que exigem conhecimento de como o software é implementado, isto é, como ele foi construído.

As técnicas de testes baseadas em estrutura são freqüentemente utilizadas para avaliar a cobertura dos testes realizados utilizando técnicas baseadas em especificação. Após esta avaliação, elas são usadas para projetar testes adicionais com o objetivo de aumentar a cobertura dos testes.

Técnicas baseadas em estrutura é uma boa maneira de gerar casos de teste adicionais diferentes dos já existentes. Eles podem ajudar a garantir mais amplitude de testes, no sentido que os casos de teste possam atingir 100% de cobertura em qualquer medida, ou seja, exercer todas as partes do software a partir do ponto de vista dos itens a serem cobertos.

Cobertura de teste pode ser medida com base em diferentes elementos estruturais de um sistema ou componente. Cobertura pode ser medida tanto em testes de componentes, quanto em testes de integração, sistema ou aceite. Por exemplo, ao nível do teste de sistema ou aceite, os itens de cobertura podem ser requisitos, opções de menu, as telas, ou transações típicas. Outras medidas de cobertura incluem coisas como elementos estruturais do banco de dados (registros, campos) e arquivos. No nível de integração, poderíamos medir a cobertura de interfaces ou específicas interações que foram testadas.

### Teste e Cobertura de Comandos

Esta técnica de teste foca em cada comando do objeto de teste. Os casos de teste devem executar um número mínimo predefinido de comandos, ou até todos os comandos do objeto de teste.

Vejamos o seguinte código abaixo, no qual é lido um número inteiro e realizada a impressão do seu valor absoluto:

**#include <stdio.h>**

**int main () {**

 **int n;**

 **printf ("Digite o numero: ");**

 **scanf ("%d", &n);**

 **if (n < 0)**

 **n = -n;**

 **printf ("Valor absoluto: %d\n",n);**

 **return 0;**

**}**

Graficamente, teríamos a seguinte estrutura:



Para este código, basta utilizarmos um caso de teste, com n < 0 e teremos a garantia de que todos os nossos comandos foram cobertos.

Os resultados e comportamentos esperados do objeto de teste devem ser identificados previamente pela especificação. Depois da execução, o comportamento do objeto de teste deve ser comparado para encontrar alguma diferença ou falha.

Como critério de finalização do teste, é definido a cobertura alcançada: Cobertura de comando = (número de comandos exercitados/número total de comandos) \*100.

Nem sempre é possível cobrir 100% do código. Além disso, essa técnica pode indicar trechos de código inatingíveis, também chamados de código morto.

### Teste e Cobertura de Decisão

Nesta técnica, as arestas do gráfico de fluxo do programa estão no centro das atenções. A execução de cada comando não é mais considerada, mas sim a execução de decisões. O resultado da decisão determina qual comando será executado em seguida. Os testes devem garantir que cada decisão seja executada com as duas possibilidades de saídas (verdadeiro e falso).

Utilizando o mesmo código do exemplo anterior, podemos perceber, conforme demonstrado abaixo que, para o caso de teste elaborado anteriormente, temos um trecho do código que não foi exercitado.



Para 100% de cobertura de desvio devemos passar por todas as setas do fluxograma. Cuidado deve ser tomado quando existe while/for, pois é necessário passar uma vez sem entrar no laço, o que não acrescenta novas “setas” cobertas.

Neste exemplo, para termos 100% de cobertura de desvio seriam necessários dois casos de teste, n < 0 e n >= 0.

O critério de finalização do teste é análogo ao do teste de cobertura de comando. Não é relevante se um trecho for executado mais de uma vez. Esse critério é verificado com a cobertura necessária definida (qual a porcentagem dos comandos que foram exercitados).

A cobertura de decisão não é uma técnica adequada a sistemas orientados a objetos, visto que a complexidade nesses sistemas reside no relacionamento entre as classes, e o fluxo de controle nas funções da classe são geralmente não muito complexos.

### Outras Técnicas Baseadas em Estrutura

Existem outras técnicas baseadas em estrutura que podem ser utilizadas para alcançar diferentes graus de rigor nos testes. Algumas técnicas são mais fortes (necessários mais testes para conseguir 100% de cobertura e, portanto, têm uma chance maior de detecção de defeitos) e outras são mais fracas.

A cobertura de decisão considera exclusivamente o valor lógico do resultado da condição (verdadeiro ou falso). Usando esse valor, é então decidido qual caminho no gráfico de fluxo de controle que é escolhido, e qual comando é o próximo a ser executado no programa, como visto na seção anterior. Se a decisão é baseada em diversas condições unidas por operadores lógicos, então a complexidade da condição pode ser considerada no teste de cobertura de condição.

Teste de condição de decisão: cada parte indivisível que forma a condição do teste deve assumir os valores verdadeiros e falsos. Uma parte indivisível é tal que não é unida por operadores lógicos (e, não, ou).

Teste de cobertura de múltiplas condições: requer que todas as combinações de verdadeiro-falso das partes indivisíveis sejam exercitadas pelo menos uma vez. Além de ser uma técnica cara por gerar muitos casos de teste, existe a possibilidade de não ser possível encontrar valores para testar todas as combinações possíveis (por exemplo, para uma variação como 3<=x E x<5, como encontrar x tal que as duas partes sejam falsas?).

Cobertura de caminho: requer a execução de todos os diferentes caminhos através do objeto de teste. Um caminho descreve a possível ordem de uma só parte do programa em um fragmento do programa. Contrário a isso, as ramificações da decisão são vistos independentemente, cada uma por si. As dependências dos caminhos considerados, por exemplo, como os laços, nos quais as ramificações conduzem de volta ao início de outra ramificação.

Para as técnicas de caixa-branca diferentes fragmentos do código são executados, e condições assumem valores lógicos diferentes. Para ser possível avaliar os testes, é necessário determinar quais fragmentos foram ou não executados. Para esse fim é necessário instrumentar o código e utilizar ferramentas de suporte. Essas ferramentas são muito importantes para os testes de caixa-branca, pois aumentam a produtividade e melhoram diretamente a qualidade do objeto de teste.

Cobertura LCSAJ (*Linear Code Sequence and Jump Coverage*): trata-se de uma variação da cobertura de caminho. Consideramos somente subcaminhos que podem ser facilmente representados no código-fonte do programa, sem a necessidade de um gráfico de fluxo. É a sequência de linhas de código que são executadas sequencialmente.

## Baseado em Defeitos

Teste baseado em defeitos é uma técnica que deve ser sempre utilizada como um complemento para outras técnicas mais formais. O sucesso para adivinhar um erro depende muito da habilidade do testador. Geralmente, bons testadores conseguem saber onde alguns dos defeitos estão. Algumas pessoas têm bastante experiência com determinadas situações negociais ou com alguns sistemas. Esse é um fator determinante na utilização deste tipo de técnica, pois essas pessoas têm uma maior facilidade em localizar os defeitos mais comuns. É por isso que uma abordagem de teste baseado em defeito, se utilizada após o emprego de técnicas mais formais, costuma ser muito eficaz.

Não existem regras para adivinhar o erro. O testador é encorajado a pensar em situações em que o software pode não ser capaz de lidar. Condições típicas podem ser tentar incluir uma divisão por zero, entrada (ou não) em branco, vazia e arquivos de dados do tipo errado (por exemplo, onde os caracteres alfa numéricos são necessários).

Uma abordagem estruturada da técnica baseada em defeitos é uma lista de possíveis defeitos ou falhas, que pode ser utilizada para projetar testes que tentam produzi-los. Estas listas de defeitos e falhas podem ser construídas com base na própria experiência do testador ou de outras pessoas, defeitos disponíveis e dados de falha, e a partir do conhecimento comum sobre o porquê do software falhar.

## Baseado em Experiência

Nas técnicas baseadas em experiência, o conhecimento das pessoas e suas competências são atributos fundamentais para as condições e casos de teste. A experiência de ambos, o pessoal técnico e de negócios é importante, pois eles trazem diferentes perspectivas para a análise do teste e do processo de modelagem. Devido à experiência anterior com sistemas semelhantes, eles podem ter diversas ideias sobre o que poderia dar errado, o que é muito útil para o teste.

Geralmente, técnicas puramente baseadas em experiência são também conhecidas como testes exploratórios. É uma abordagem livre na qual os testadores estão envolvidos em um nível mínimo de planejamento e máximo de execução de teste. O planejamento envolve a criação de um plano de teste, uma curta declaração do escopo, objetivos e abordagens a serem usados em um curto período de execução de testes (1 a 2 horas).

O projeto dos testes e as atividades de execução são realizados em paralelo, geralmente sem documentação formal das condições, casos ou scripts de teste. Isso não significa que outras técnicas de teste mais formais não serão utilizadas. Algumas notas serão escritas durante a sessão de teste exploratório, de modo que um relatório pode ser produzido ao final da atividade.

O registro de teste é realizado da mesma forma que a execução, documentando os aspectos chave que estão sendo testados, eventuais defeitos encontrados e qualquer pensamento sobre eventuais novas possibilidades de testes. Um aspecto fundamental do teste exploratório é a aprendizagem por parte do testador sobre o software, o seu uso, seus pontos fortes e suas fraquezas. Como o próprio nome indica, teste exploratório serve para explorar, descobrir sobre o software, o que faz, o que não deve fazer, o que funciona e o que não funciona. O testador está em constante tomada de decisões sobre o próximo teste e onde gastar o tempo (limitado).

Esta é uma abordagem que é mais útil quando não há especificações ou quando elas são pobres, e quando o tempo é muito limitado. Ele também pode servir para complementar outras técnicas de teste mais formais, ajudando a estabelecer uma maior confiança no software. Desta forma, testes exploratórios podem ser usados como um controle sobre o processo de teste formal, ajudando a garantir que os defeitos mais graves foram encontrados.

# Planejamento & Controle de Teste

A maior parte do planejamento de teste ocorre no início do esforço de teste, e envolve a identificação e implementação de todas as atividades e recursos requeridos para corresponder à missão e objetivos identificados na estratégia de teste.

O teste baseado em risco é usado para informar ao processo de planejamento de teste sobre as atividades de mitigação necessárias para reduzir os riscos de produto identificados. O teste baseado em risco informará também ao processo de planejamento de teste em relação às prioridades relativas à atividade de teste.

Relacionamentos complexos podem existir entre a base de teste, condições de teste, casos de teste e procedimentos de teste, de tal forma que relacionamentos de muitos para muitos podem existir entre esses produtos de trabalho. Eles devem ser compreendidos para permitir que o planejamento e controle de teste sejam efetivamente implementados.

O controle de teste é uma atividade contínua. Ele envolve a comparação do progresso real mediante o plano e relato do status, incluindo desvios do plano. O controle de teste conduz o teste para satisfazer a missão, estratégias e objetivos, incluindo rever as atividades de planejamento de teste conforme necessário.

O controle de teste deve tratar das informações geradas pelo teste, assim como das mudanças de condições nas quais um projeto ou um objetivo existe.

As métricas para monitorar o planejamento e controle de teste devem incluir:

* Risco e cobertura de teste;
* Descoberta de defeito e informação;
* Horas planejadas versus horas reais para desenvolver o testware e executar os casos de teste.

O planejamento do teste não está preocupado apenas com o planejamento de um único nível de testes, mas deve abordar todos os níveis de teste que devem ocorrer. Em outras palavras, deve haver um plano de teste mestre que abrangerá os diversos níveis e garantirá que a quantidade correta de teste acontece no momento que realmente deve ocorrer. Por exemplo, se considerarmos uma área de risco especial de qualidade, muito perigosa, o plano de teste mestre deve incluir a cobertura dessa área de risco no início do ciclo de vida e, provavelmente, em vários outros níveis do ciclo de vida.

Durante o planejamento de teste estabelecem-se os cenários operacionais envolvidos no sistema e também, um quadro para derivar casos de teste, condições e procedimentos de teste. A base de teste pode incluir as especificações de requisitos, as especificações do projeto, os riscos, qualidade e outros itens. Um bom plano de teste, executado corretamente, ajuda a garantir que nós realmente testamos o que se propõe a testar durante a execução do teste. Isso parece simples, mas o plano tem que lidar com a complexa relação entre, por um lado, a base de teste, uma vez que existe durante o planejamento de teste para o projeto e, por outro lado, os casos de teste, condições e procedimentos existentes durante a execução do teste para o projeto.

Uma vez que temos estabelecido o plano de teste, inicia-se o controle de teste. Controle de teste é uma atividade contínua. No controle de teste, podemos comparar o progresso atual contra o plano. Nós relatamos o status de execução do plano, incluindo eventuais desvios que possam surgir. Porque os projetos não são estáticos, mas estão evoluindo, controle é necessário para orientar os testes no cumprimento da missão, estratégias e objetivos do teste.

O controle de teste deve responder às mudanças na missão, estratégias e objetivos do teste, ou as alterações no projeto. Por exemplo, poderíamos descobrir que nossa análise de risco estava errada em termos de probabilidade de defeito em uma área, que nos levaria a realocar o esforço restante e priorizar os testes. Ou, talvez tenhamos que lidar com uma quantidade reduzida de tempo para execução do teste, levando-nos a utilização de risco para orientar uma redução no número total de testes.

# Modelagem & Análise de Teste

Durante o planejamento de teste, um conjunto de objetivos de testes será identificado. O processo de análise e modelagem de teste tem como objetivos:

* Identificar as condições de teste;
* Criar casos de teste que exercitem as condições de teste identificadas.

A atribuição de prioridade para cada condição de teste geralmente consiste em determinar a probabilidade e os impactos associados a cada item de risco de qualidade, ou seja, nós avaliamos o nível de risco para cada item de risco. A prioridade determina a repartição do esforço de teste (durante todo o processo de teste) e o fim do projeto, implementação e execução dos testes relacionados.

Ao longo do processo, as condições específicas de teste e das correspondentes prioridades podem mudar conforme as reais necessidades e também, com a nossa compreensão das necessidades do projeto e a evolução dos participantes do projeto.

Essa priorização, o uso de priorização e redefinição de prioridades ocorre regularmente no processo de teste. Tudo começa durante a análise de risco e planejamento de testes, é claro. Ele continua durante todo o processo, desde a análise e design até a implementação e execução. Ela influencia a avaliação dos critérios de saída e relatórios de resultados do teste.

1.

## Identificação de Condições de Teste

Condições de teste são identificadas através da análise da base de teste e de objetivos para determinar o que testar usando técnicas de teste identificadas na Estratégia de Teste e/ou o Plano de Teste.

A decisão de determinar o nível e estrutura das condições de teste pode ser baseada em características funcionais e não-funcionais dos itens de teste usando o seguinte:

1. Granulosidade da base de teste: Exemplo: requisitos de alto nível podem gerar inicialmente condições de alto nível. Exemplo: prove que a tela X funciona, a partir da qual é possível derivar condições de teste de baixo nível. Exemplo: prove que a tela X rejeita um número que é um dígito menor que o comprimento correto;
2. Riscos de produtos correspondidos: Exemplo: para uma característica de alto nível de risco, condições de teste detalhadas devem ser um objetivo definido;
3. Requisitos para relatório gerencial e rastreabilidade de informação;
4. Quando a decisão for tomada para trabalhar com condições de teste somente e não desenvolver caso de teste. Por exemplo, usando condições de teste para foco de teste sem script.

## Criação de Casos de Teste

Casos de teste são projetados por uma elaboração passo a passo e refinamento de condições de teste identificadas usando técnicas de teste discriminadas na estratégia de teste.

Eles podem ser repetíveis, verificáveis e rastreáveis aos requisitos.

A modelagem de caso de teste inclui a identificação de:

* Pré-condições tais como o projeto ou os requisitos do ambiente de teste localizados e os planos para sua entrega;
* Os requisitos de dados de teste;
* Os resultados esperados e as pós-condições.

Um desafio particular é geralmente a definição do resultado esperado de um teste; isto é, a identificação de um ou mais oráculos de teste que podem ser usados para o teste. Na identificação de resultados esperados, os testadores se concentram não somente nas saídas na tela, mas também com as pós-condições dos dados e do ambiente.

Caso a base de teste esteja claramente definida, isso pode ser teoricamente simples. No entanto, bases de teste são frequentemente vagas, contraditórias, com lacunas na cobertura de áreas importantes, ou completamente inexistentes. Em tais casos, o testador deve ter, ou ter acesso a, conhecimento no assunto. Também, mesmo quando a base de teste é bem especificada, interações complexas de estímulos e respostas podem dificultar a definição de resultados esperados, então um oráculo de teste é essencial. A execução de teste sem nenhuma forma de determinar a precisão dos resultados tem um valor adicional ou benefícios bem baixos, gerando relatórios de incidentes ilegítimos e uma falsa confiança no sistema.

As atividades descritas acima podem ser aplicadas a todos os níveis de teste, ainda que a base de teste varie. Por exemplo, testes de aceite de usuário podem ser baseados primariamente na especificação de requisitos, casos de uso e processos de negócios definidos, enquanto que testes de componentes podem ser baseados primariamente em especificações de modelagem de baixo nível.

Durante o desenvolvimento de condições e casos de teste, comumente alguma documentação é feita resultando nos produtos do trabalho de teste. Um padrão para tal documentação é encontrado na IEEE 829. Essa norma discute os principais tipos de documentos aplicáveis à Análise e Modelagem de teste, Especificação de Modelagem de Teste e Especificação de Caso de Teste, assim como implementação de teste. Na prática, a extensão de quais produtos de teste são documentados varia consideravelmente. Como exemplo, isso pode ser influenciado por:

* Riscos de projeto (o que deve/não deve ser documentado);
* O “valor agregado” que a documentação traz ao projeto;
* Padrões a serem seguidos;
* Modelo de ciclo de vida usado (por exemplo, uma abordagem ágil tenta minimizar a documentação garantindo comunicação próxima e freqüente da equipe);
* Os requisitos para rastreabilidade da base de teste, através de análise e modelagem de teste.

Métricas para monitorar a análise e modelagem de teste podem incluir:

* Porcentagem dos requisitos cobertos pelas condições de teste;
* Porcentagem das condições de teste cobertas pelos casos de teste;
* Número de defeitos encontrados durante a análise e modelagem de teste.

# Implementação e Execução de Teste

1.

## Implementação de Teste

A implementação de teste inclui a organização dos casos de teste em procedimentos (scripts de teste), finalizando os dados e o ambiente de teste, e formando a programação da execução de teste para viabilizar o início da execução dos casos de teste. Isso também inclui a verificação mediante critérios explícitos e implícitos de entrada para o nível de teste em questão.

Os procedimentos de teste devem ser priorizados para garantir que os objetivos identificados em uma estratégia tenham sido alcançados de uma forma eficiente, por exemplo, executando os procedimentos de teste mais importantes primeiro pode ser uma abordagem.

O nível de detalhe e a complexidade associada para o trabalho realizado durante a implementação de teste devem ser influenciados pelo detalhe dos produtos de trabalho do teste (casos e condições de teste).

Os dados de teste são necessários para o teste e, em alguns casos, esses conjuntos de dados podem ser realmente grandes. Durante a implementação, os testadores criam dados de entrada e de ambiente para carregar os bancos de dados e outros repositórios. Testadores também criam scripts e outros geradores de dados que irão criar dados que são enviados para o sistema como carga de entrada durante a execução do teste.

Durante a implementação de teste, os testadores devem finalizar e confirmar o pedido no qual os testes manuais e automatizados devem ocorrer. Quando a automatização é contratada, a implementação de teste inclui a criação do ambiente preparado de teste e scripts de teste.

Testadores devem verificar cuidadosamente as restrições que podem requerer que os testes sejam realizados em uma determinada ordem. Dependências no ambiente e nos dados de teste devem ser conhecidas e verificadas.

A implementação de teste também se preocupa com o ambiente de teste. Durante esse estágio ele deve ser completamente iniciado e verificado antes da execução do teste. Um ajuste do ambiente de teste é essencial: o ambiente de teste deve ser capaz de permitir a exposição dos defeitos presentes nas condições de teste, operando normalmente quando falhas não ocorrem, e replicados adequadamente se requisitado, por exemplo, na produção ou no ambiente do usuário final para níveis mais altos de teste.

Durante a implementação de teste, os testadores devem assegurar que os responsáveis pela criação e manutenção do ambiente de teste são conhecidos e disponíveis e que todo o testware, ferramentas de suporte ao teste e processos associados estão prontos para o uso. Isso inclui o gerenciamento de configuração, gerenciamento de incidentes, registro e gerenciamento do teste.

Juntamente a isso, os testadores devem verificar os procedimentos que coletam dados para a verificação do critério de saída e o relatório dos resultados de teste.

É bom usar uma abordagem balanceada para a implementação de teste. Por exemplo, estratégias de teste analítica baseada em risco são frequentemente aliadas a estratégias dinâmicas de teste. Nesse caso, alguma porcentagem do esforço da execução de teste é alocada para o teste que não segue scripts predeterminados.

O teste sem script não deve ser ad hoc ou a esmo já que este pode ser imprevisível em duração a menos que esteja com tempo limitado. Os testadores têm desenvolvido uma grande variedade de técnicas baseadas em experiência, tais como ataques, suposição de erro e teste exploratório. Análise, modelagem e implementação de teste ainda ocorrem, mas elas acontecem primariamente durante a execução do teste. Seguindo tais estratégias de teste dinâmico, os resultados de cada teste influenciam a análise, modelagem e implementação dos testes subsequentes. Ao passo que essas estratégias são leves e diversas vezes efetivas no encontro de bugs, elas necessitam de testadores experientes, podem ser imprevisíveis em duração, várias vezes não fornecem uma boa cobertura de informação, e podem ser difíceis de repetir sem auxílio de ferramentas específicas para teste de regressão.

## Execução de Teste

A execução de teste começa quando o objeto de teste é entregue e o critério de entrada para a execução do teste foi satisfeita. Os testes devem ser executados de acordo com os procedimentos de teste, apesar de que algum grau de liberdade deve ser dado aos testadores para garantir a cobertura de cenários adicionais de teste de interesse e de comportamentos que são observados durante o teste (qualquer falha detectada durante tal desvio deve descrever as variações dos procedimentos de teste que são necessários para reproduzir a falha). Testes automatizados seguirão suas instruções definidas sem desvios.

No âmago da atividade de execução de teste está a comparação dos resultados reais com os esperados. Os testadores devem prestar a maior atenção e manter o foco nessas tarefas, senão todo o trabalho de modelagem e implementação de teste pode ser desperdiçado quando falhas estão faltando (falsos positivos) ou comportamentos corretos são mal classificados como incorretos (falsos negativos). Se os resultados atuais e os esperados não se correspondem, um incidente aconteceu.

Os incidentes devem ser cuidadosamente esmiuçados para estabelecer a causa (que deve ou não ser um defeito no objeto de teste) e para coletar dados para auxiliar a resolução do incidente.

Quando um defeito é identificado, a especificação de teste deve ser cuidadosamente analisada para assegurar que é correto. Uma especificação de teste pode ser incorreta por diversas razões, incluindo problemas nos dados de teste, defeitos na documentação de teste, ou um engano na forma que foi executado. Se ele não é correto, ele pode ser corrigido e re-executado. A partir do momento que mudanças na base e no objeto de teste podem resultar em uma especificação incorreta mesmo depois de várias execuções de sucesso, os testadores devem manter-se atentos à possibilidade de que os resultados observados podem ser devidos a testes incorretos.

Durante a execução de teste, os resultados de teste devem ser registrados de forma apropriada. Os testes que foram executados e que não foram registrados devem ser repetidos para identificar o resultado correto, trazendo ineficiência e atrasos. (Note que um registro adequado pode tratar de preocupações sobre cobertura e repetibilidade associadas com estratégias de teste dinâmicas).

Desde que os objetos de teste, *testware* e ambiente de teste podem estar todos evoluindo, o registro poderá identificar as versões específicas testadas.

O registro de teste fornece uma gravação cronológica de detalhes relevantes sobre a execução dos testes.

O registro de resultados é aplicado tanto para testes individuais como para eventos. Cada teste deve ser unicamente identificado e seu status registrado com a execução do teste. Quaisquer eventos que afetem a execução do teste devem ser registrados. Informações suficientes devem ser registradas para medir a cobertura dos testes e documentar os motivos dos atrasos e interrupções no teste. Além disso, informações devem ser registradas para apoiar o controle do teste, o relato de progresso do teste, as medições de critério de saída e a melhoria do processo de teste.

O registro varia de acordo com o nível do teste e a estratégia. Por exemplo, se um teste de componente automatizado estiver acontecendo, os testes automatizados recolhem várias das informações registradas. Se um teste de aceite manual estiver sendo feito, o gerente de teste pode compilar ou organizar o registro. Em alguns casos, assim como com a implementação dos testes, o registro é influenciado por regulamentos ou requisitos de auditoria.

A norma IEEE 829 inclui uma descrição da informação que poderia ser capturada em um registro de teste:

* Identificador do registro de teste;
* Descrição;
* Entradas de atividades e eventos.

Em alguns casos, os usuários ou clientes podem participar da execução de teste. Isso pode servir como uma forma para construir confiança no sistema, apesar de que assume que os testes não terão grande sucesso em encontrar defeitos. Tais suposições são frequentemente inválidas em níveis de teste mais iniciais, mas podem ser válidos durante testes de aceite.

Métricas para monitorar a implementação e execução dos testes podem incluir:

* Porcentagem de ambientes de teste configurados;
* Porcentagem de registros de dados de teste carregados;
* Porcentagem de condições e casos de teste executados;
* Porcentagem de casos de teste automatizados.

## Avaliação do Critério de Saída e Relatório

Do ponto de vista do processo de teste, o progresso da monitoração do teste acarreta na garantia da coleta de informações apropriadas para apoio de requisitos a serem relatados. Isso inclui medição do progresso em direção ao encerramento.

As métricas para monitorar o progresso e finalização do teste incluirão um mapeamento do critério de saída combinado (acertado durante o planejamento de teste), o qual pode incluir um ou todos dos seguintes itens:

* Número de condições, casos e especificações de teste, planejados e aqueles executados e discriminados por quando eles passaram ou falharam;
* Defeitos totais encontrados, discriminados pela severidade e prioridade para aqueles corrigidos ou pendentes;
* Número de alterações (mudanças solicitadas) promovidas, aceitas (construídas) e testadas;
* Custos planejados versus custos reais;
* Tempo gasto planejado versus tempo gasto real;
* Riscos identificados discriminados por sua mitigação pela atividade de teste e qualquer um mantido pendente;
* Porcentagem de tempo de teste perdido por eventos bloqueadores;
* Itens retestados;
* Tempo total de tempo planejado mediante o tempo de teste efetivamente gasto.

Para um relatório de teste a IEEE especifica um Relatório de Resumo de Teste, consistindo das seguintes seções:

* Identificador de relatório de resumo de teste;
* Resumo;
* Variações;
* Contribuição compreensiva;
* Resumo dos resultados;
* Avaliação;
* Resumo das atividades;
* Aprovações.

O relato de teste pode ocorrer após cada nível de teste que esteja completo, assim como no final de todo o esforço de teste.

# REFERÊNCIAS

* PS-MCTIC – Processo de Software do Ministério da Ciência, Tecnologia, Inovações e Comunicações
* Guide to the 2006 CSTE CBOK - Versão 6.2
* Foundation Level Syllabus - Versão 2011br
* Advanced Level Syllabus - Versão 2007br
* Advanced Software Testing Vol. 1: Guide to the ISTQB Advanced Certification as an Advanced Test Analyst, Rex Black
* Advanced Software Testing Vol. 2: Guide to the ISTQB Advanced Certification as an Advanced Test Manager, Rex Black
* Foundations of Software Testing - ISTQB CERTIFICATION - Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black