Testes de *software*

Mauro Henrique Lima de Boni - mauro@ifto.edu.br

2018-08-20

Antes de começar é importante ler essa parte...

Qual o propósito deste material

Este material de apoio é destinado inicialmente aos estudantes do curso Superior de Tecnologia em Sistemas para Internet (CSTSI) do Instituto Federal de Educação, Ciência e Tecnologia do Tocantins, mas também pode ser usado por discentes de outros cursos tais como Ciência da Computação, Sistemas de Informação, Engenharia da Computação e Engenharia de Software. Ele tem o objetivo de apresentar os principais conceitos relacionados a testes de *sofware*, seus níveis, seus objetivos, como os testes são projetados e executados. De nenhuma forma esse material tem pretensão de cobrir todos os assuntos relacionados à área de testes, que é muito extensa. Ele também não substitui livros de Engenharia de Software ou outros livros técnicos relacionados. A proposta é que ele sirva como um guia inicial para aqueles que estão dando os primeiros passos nessa área tão importante para o desenvolvimento de *software*.

Procure por outras fontes para complementar esse material. Um bom lugar para começar é o site INFOQ que traz palestras e artigos atualizados, relacionados a diversos aspectos da engenharia de *software*.

Como o material está organizado

É importante lembrar que o presente material foi construído com o intuito prioritário de atender ao CSTSI. Assim, as buscas por conteúdo foram feitas para atender à ementa da disciplina que está em vigor, como mostrada a seguir:

- Conceitos básicos relacionados a testes de *software*;
- Processos de testes;
- Ferramentas para planejamento, elaboração e automatização de testes de *software*;
- Manipular ferramentas para execução de planos de testes de *software*;
- Classes de automação: QAI x ISTQB;
- Gerenciamento do planejamento de testes;
- Projeto de Casos de Teste;
- Tipos de Testes;
- Ferramentas;
- Gerenciamento dos defeitos;
- Elaboração de laudo e parecer técnico.

Com base nisso, o conteúdo disponível nesse material está assim dividido:

- O capítulo "As atividades do processo de *software* e os testes" tem o objetivo de posicionar as atividades de teste de *sofware* dentre as demais atividades básicas do ciclo de desenvolvimento

de *sofware*. As interações entre o teste e análise, projeto e implementação são apresentadas com o objetivo de auxiliar o estudante em um primeiro contato e entender em que momento elas ocorrem. Trata-se de capitulo informativo que não está relacionado diretamente com a ementa do curso.

- O capítulo Uma visão geral sobre o que é teste de *software* apresenta o conceito sobre teste e reforça sua importância na obtenção de um *sofware* que atenda melhor aos requisitos do cliente e que seja construído da melhor forma possível. Aqui é onde o estudo relativo aos testes começa de fato.

Público alvo

O público alvo desse livro, conforme mencionado anteriormente, são os estudantes do Curso Superior de Técnologia em Sistemas para Internet do Instituto Federal de Eucação, Ciência e Tecnologia do Tocantins, na modalidade presencial. Ele pode ser usado por outros cursos em que os estudantes tenham tido contato com disciplinas de Análise e Projeto de Sistemas, Introdução a Programação ou Programação para Web.

Como você deve estudar cada capítulo

- Leia a visão geral do capítulo
- Estude os conteúdos das seções
- Realize as atividades no final do capítulo
- Verifique se você atingiu os objetivos do capítulo

Na sala de aula do curso

- Tire dúvidas e discuta sobre as atividades do livro com outros integrantes do curso
- Leia materiais complementares eventualmente disponibilizados
- Realize as atividades propostas pelo professor da disciplina

1. As atividades do processo de *software* e os testes

1.1. Introdução

Esse capítulo tem como objetivos:

- fazer uma breve revisão sobre as atividades do processo de *software*;
- fornecer ao estudante uma ideia geral de onde os testes de *software* estão inseridos dentro do processo de desenvolvimento;
- realizar algumas conexões entre essa disciplina e outras do curso, sobretudo Análise de Sistemas, Projeto de Sistemas, Programação para Web II.

1.1.1. Como esse capítulo está estruturado

O presente capítulo possui três seções principais. Cada uma delas trata de um assunto específico, conforme a lista a seguir:

- A seção Atividades do processo de *software* é um breve resumo sobre as atividades que são executadas durante o processo de *software*, de maneira que possamos encontrar onde os testes estão inseridos.
- A seção Quando a atividade de testes inicia? faz uma revisão sobre dois modelos de desenvolvimento bastante conhecidos: o cascata e o incremental. Ainda discute brevemente sobre quando os testes devem começar.
- A última seção apresenta Relação entre essa disciplina e outras disciplinas no curso.

1.2. Atividades do processo de *software*

Antes de falarmos sobre os testes é necessário localizá-los dentre os processos da Engenharia de *software*. Existem uma grande quantidade de processos de desenvolvimento de *software* diferentes mas todos possuem as seguintes atividades:

- especificação – definição do quê o sistema deve fazer;
- projeto e implementação – definição da organização e implementação do sistema;
- verificação e validação – checagem se o sistema faz o que o cliente deseja e se satisfaz os requisitos não funcionais;
- evolução – evolução em resposta à mudanças nas necessidades do cliente.

Na primeira atividade, especificação, é onde conseguimos obter um entendimento geral sobre o problema a ser resolvido. Não temos todos os detalhes ainda, mas é possível obter o mínimo necessário para passar para as próximas etapas. Podemos dizer ainda que essa etapa estabelece quais serviços são necessários e quais são as restrições na operação e desenvolvimento do sistema. A disciplina Análise de Sistemas é a responsável por tratar dos assuntos relacionados a essa etapa. Projeto e Implementação é uma atividade que pode ser entendida contendo duas partes. Elas

dificilmente são tratadas em separado dentro do mundo da computação. Essas partes têm como objetivo a conversão da especificação do sistema em um sistema executável. Em projeto criamos uma estrutura de *software* que possa materializar a especificação; e por implementação devemos entender que é o ato de transformar o projeto em um programa executável. No curso de sistemas para internet a disciplina Projeto de *software* é responsável por apresentar as técnicas e ferramentas utilizadas para a elaboração de estrutura do *software*. Já implementação está fracionada em várias disciplinas tais como Programação Orientada ao Objeto, Programação para Web, dentre outras.

A terceira atividade, Verificação e Validação (V&V) tem o objetivo de mostrar que o produto que estamos construindo, um sistema por exemplo, está em conformidade com sua especificação e que ainda está de acordo com os requisitos do cliente. Essa etapa é o objeto de estudo desta disciplina.

Por fim, temos a atividade chamada de evolução de *software*, que ocorre quando é necessário alterar um sistema que já existe, tornando-o adequado à novas necessidades. Devemos lembrar que qualquer *software* precisa evoluir, pois caso isso não aconteça, ele deixará de atender às necessidades de seus usuários, o que pode torná-lo inútil em pouco tempo. Não há disciplina específica no Curso de Sistema para Internet do IFTO e provavelmente em nenhum outro curso que lide diretamente com essa questão. No entanto, existe literatura específica que trata dos chamados *softwares* legados.

1.3. Quando a atividade de testes inicia?

A resposta para a pergunta depende da maneira como as atividades do processo de *software* estão organizadas. Resumidamente, tudo depende do paradigma que está sendo usado. Assim, se estamos trabalhando em um paradigma orientado a planos, como o modelo cascata, elas são organizadas em sequência. Isso significa que os testes só começarão após as atividade de implementação terem sido concluídas. Já se estivermos falando sobre um ambiente que usa um modelo incremental, tal qual os modelos Ágeis, as atividades de (i) projeto e implementação e (ii) verificação e validação são intercaladas. Para clarificar um pouco mais, a seguir os modelos em cascata e incremental são apresentados e as figuras Exemplo do modelo em cascata e Exemplo do modelo incremental mostram como as atividades são organizadas.

1.3.1. Testes no modelo cascata

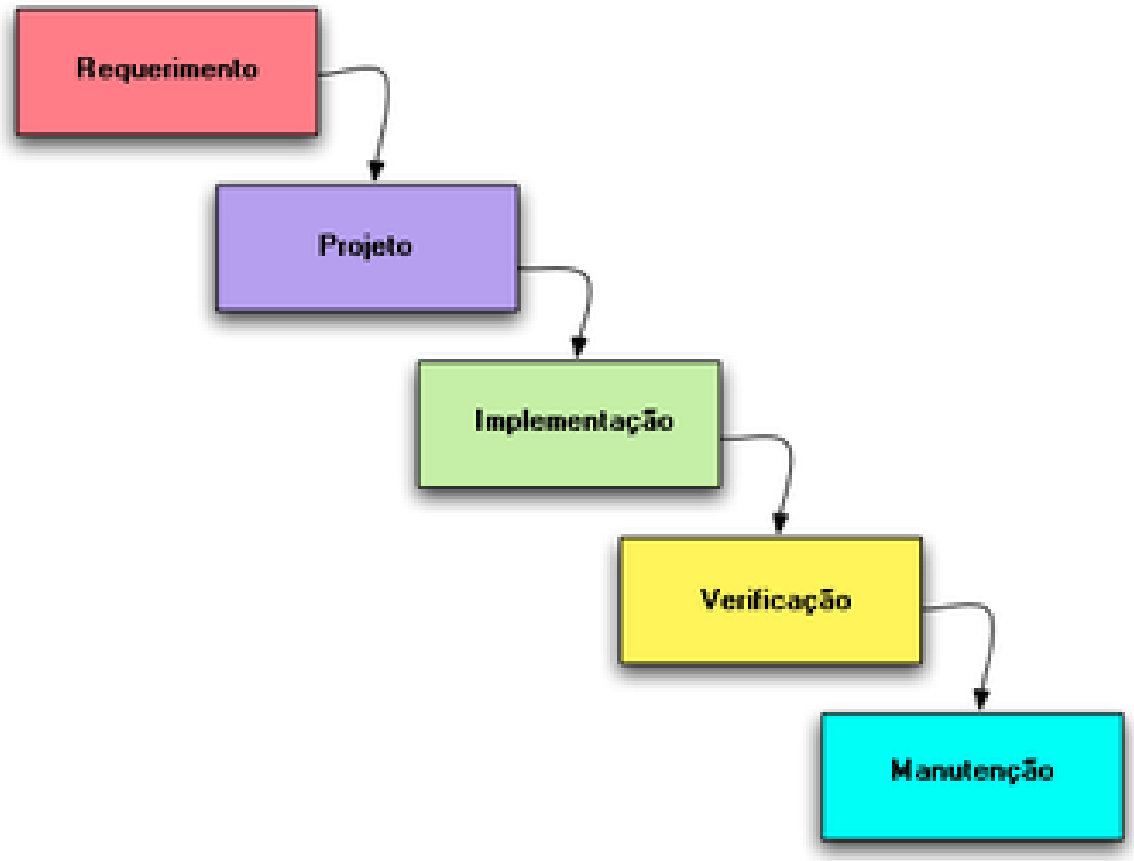

Figura 1. Exemplo do modelo em cascata

Antes de mais nada é importante lembrar que o esse modelo, que pode ser chamado de ciclo de vida clássico, que segue uma abordagem sistemática e sequencial. Ele pode ser usado em cenários em os requisitos são bem entendidos ou que mudam pouco. Esse é o paradigma mais antigo da Engenharia de *software*, tendo provavelmente sido introduzido por engenheiros de outras áreas.

No modelo cascata as atividades precisam ser completadas antes de se mover para a próxima. É possível a realização de testes durante a etapa de implementação, onde podem ser executados vários testes de desenvolvimento. Depois que esses testes tenham sido concluídos com êxito é que podemos passar para a verificação e realizar outros tipos de testes, como os testes de sistema e testes de usuário. Mas para atingirmos essas etapas devemos ter concluída as outras duas.

1.3.2. Testes no modelo incremental

Segundo Pressman[PRESSMAN], há muitas situações em que os requisitos iniciais são razoavelmente definidos e pode haver a necessidade de fornecer rapidamente um conjunto limitado de funcionalidades do *software* aos usuários. Tais requisitos serão então aumentados em versões posteriores. Em situações como essa, um modelo incremental pode ser escolhido.

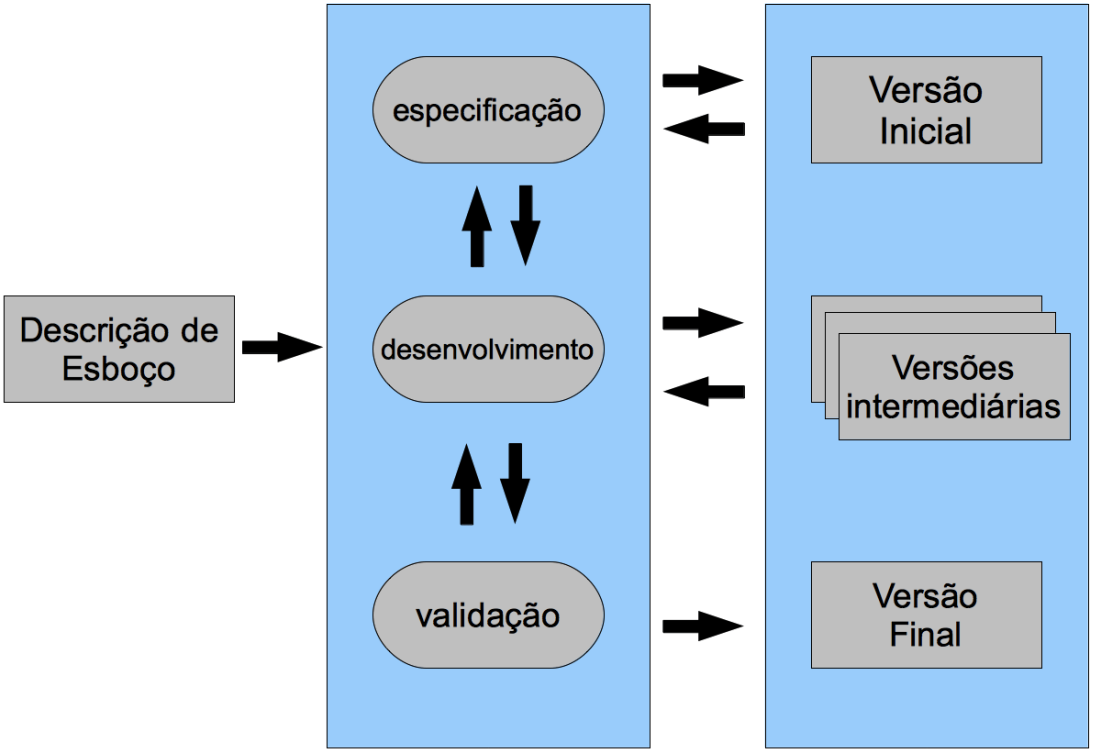

Figura 2. Exemplo do modelo incremental

O modelo incremental é mais flexível que o modelo cascata. Tudo começa com um esboço do sistema. Uma vez que esse esboço seja feito é possível passar para a próxima atividade que intercala especificação, desenvolvimento e validação. Isso significa que a partir do esboço é possível especificar os requisitos, aplicar técnicas de projeto, fazer a implementação e realizar a validação. Essas validações permitem que sejam entregues versões intermediárias que são na verdade um produto operacional.

Cumpre ainda dizer que nos últimos anos, a visão sobre os testes vem mudando. Dessa maneira, o teste já não é mais visto como uma atividade que começa somente após a conclusão da fase de implementação, com o objetivo limitado de detectar falhas. Testes de *software* é, ou deveria ser, executado durante todo o ciclo de vida de desenvolvimento e manutenção. De fato, o planejamento para testes de *software* deve começar com os estágios iniciais do processo de requisitos de *software*. Os planos e procedimentos de testes devem ser sistematicamente e continuamente desenvolvidos - e possivelmente refinados - à medida que o desenvolvimento de *software* prossiga. Essas atividades de planejamento de testes e projeto de testes fornecem informações úteis para os projetistas de *software* e ajudam a destacar possíveis fraquezas, como omissões / contradições de design ou omissões / ambiguidades na documentação.

1.4. Relação entre essa disciplina e outras disciplinas no curso

1.4.1. Relação existente entre Análise de Sistema e Testes

Do ponto de vista do cliente, os maiores erros são aqueles que deixam de satisfazer aos seus requisitos. Assim, é importante que uma relação contendo os requisitos funcionais do produto que

está sendo desenvolvido, e que posteriormente será testado, tenha sido construída. Em geral durante a fase de análise é que essa relação é feita. Conforme dito anteriormente, o planejamento de testes pode ter início durante o processo de especificação de requisitos, onde os chamados critérios de aceitação podem ser criados.

1.4.2. Relação existente entre disciplinas de Programação e Testes

Os testes de integração e de unidade estão fortemente relacionados com ações de implementação quando queremos garantir um nível maior de qualidade ao produto que está sendo produzido. Pode-se testar desde métodos ou funções isoladamente, até sistemas inteiros. :cap: Capitulo2 :imagesdir: imagens/Capitulo1/

2. Uma visão geral sobre o que é teste de *software*

2.1. Introdução

Esse capítulo tem o objetivo de passar uma ideia geral sobre os conceitos relacionados ao teste de software.

2.1.1. Como esse capítulo está estruturado

O presente capítulo possui quatro seções. Cada uma delas trata de um assunto específico, conforme a lista a seguir:

- A seção Conceitos iniciais traz os conceitos iniciais sobre testes. Esses conceitos são informações fundamentais se você está tendo seu primeiro contato com testes.
- A seção Quais são os objetivos do teste de *software*? é a responsável por apresentar qual o propósito do teste de software.
- A seção Por que executar testes de *software*? tem como objetivo justificar porque os testes devem ser feitos.
- A seção [atividades] tem alguns exercícios que o ajudaram a fixar melhor o conteúdo que foi apresentado.

2.2. Conceitos iniciais

O teste de *software* é uma atividade que tem o objetivo de verificar se os resultados obtidos, através do uso do produto, correspondem aos resultados esperados. Além disso, ela é responsável também por tentar fazer com que o sistema que está sendo testado esteja livre de Defeitos.

Roger Pressmann, diz que o **teste de *software* faz parte de um processo amplo ao qual podemos chamar de Verificação e Validação (V&V)**. Por Verificação estamos nos referindo a um conjunto de atividade que estão relacionadas com a garantia de que o *software* "implementa corretamente uma função específica". Já a Validação tem como preocupação o atendimento aos requisitos do cliente.

Segundo Ian Sommerville [SOMMERVILLE], o teste de *software* é uma etapa dentro do ciclo de vida que **visa descobrir os defeitos que um programa possui, antes que ele entre em produção**. Isto ocorre antes de o *software* ser entregue para que os usuários comecem a utilizá-lo.

Edsger Dijkstra, argumenta que **os testes apenas são capazes de mostrar a presença de erros e não a sua ausência**, pois é possível de usar uma entrada de dados de testes que não seja boa ou até mesmo esquecermos de testar alguma parte do sistema. Assim, os testes não conseguem demonstrar que não há defeitos ou que ele sempre se comportará da maneira prevista em qualquer situação.

Podemos entender que o teste é um processo independente, pois nem sempre quem desenvolve

testará o *software*. Além disso, cumpre lembrar que o número de tipos diferentes de testes varia tanto quanto as técnicas de desenvolvimento.

1. Envolve processos de (i) inspeção e revisão e (ii) testes do sistema.
2. Testes do sistema envolvem executar o sistema com casos de teste. Tais casos são provenientes de especificações dos dados reais que deverão ser processados pelo sistema.
3. O teste é a atividade de V&V mais usada.

2.2.1. Defeito x Erro x Falha

É importante, antes de definirmos melhor o que é teste, diferenciar três termos: defeito, erro e falha. Há autores como [PRESSMAN] que afirmam que dentro da comunidade de Engenharia de *Software*, esses termos são tratados como sinônimos. Ele, no entanto, faz uma diferenciação entre erro e defeito usando o critério temporal. Assim, temos:

- Erro: é qualquer problema encontrado antes que o software tenha sido entregue aos usuários finais.
- Falha: é um problema que é encontrado somente após a entrega para os usuários finais.

Por outro lado, a *International Software Testing Qualification Board* ISTQB, entidade que certifica os profissionais da área de teste, faz uma distinção maior, adicionando o termo defeito. Ela altera um pouco o significado de erro e falha, criando uma espécie de graduação:

- Defeito: é uma deficie ncia meca nica ou algori tmica que, se ativada, pode levar a uma falha. É uma inconsistência no *software*, algo que foi implementado de maneira incorreta. Ele ocorre em uma linha de código, como uma instrução errada ou um comando incorreto. O defeito é a causa de um erro, porém, se uma linha de código que contém o defeito nunca executar, o defeito não vai provocar um erro.
- Erro: Erro humano produzindo resultado incorreto. O erro evidencia o defeito, ou seja, quando há diferença entre o valor obtido e o valor esperado, isto constitui um erro.
- Falha: evento nota vel em que o sistema viola suas especificac o es. Ele é o resultado ou manifestação de um defeito ou erro.

Neste material sempre usaremos erro e falha como sinônimos, mas o autor considera importe que você saiba que existem outras maneiras de interpretar esses termos.

2.3. Quais são os objetivos do teste de *software*?

Em seu livro de Engenharia de *Software*, Roger Pressman elenca três objetivos principais, a saber:

- Teste é um processo de execução de um programa com a finalidade de encontrar um erro.
- Um teste bem-sucedido é aquele que descobre um erro ainda não descoberto.
- Um bom caso de teste é aquele que tem alta probabilidade de encontrar um erro ainda não descoberto.

Isso posto, nos faz pensar que **estamos indo contra a visão inicial que alguns de nós poderíamos ter de que um teste bem-sucedido é aquele no qual não são encontrados erros.**

Os profissionais que se dedicam a testar *softwares* são responsáveis por projetar testes que descubram diferentes tipos de erros. Isso deve ser feito de uma forma sistemática dentro de um ambiente em que há sempre uma pressão muito grande em relação ao produto que está sendo testado. Os testes devem ser os mais completos possíveis, mas eles contam com uma quantidade mínima de tempo e devem consumir o mínimo de esforço possível.

2.4. Por que executar testes de *software*?

O teste é importante porque os erros de *software* podem ser caros ou até tornarem-se perigosos. Erros de *software* podem causar perdas monetárias e humanas. Nossa história recente está cheia de exemplos.

- O Voo 501 da Agência Espacial Européia Ariane 5 foi destruído 40 segundos após a decolagem (4 de junho de 1996).
- O protótipo de foguete de US$ 1 bilhão foi destruído devido a um bug no *software* de orientação a bordo. Clique no link abaixo para ver como foi o lançamento e a destruição do foguete.

  ```
  http://youtu.be/gp_D8r-2hwk
  ```

Podemos então dizer a grosso modo, que o teste de *software* é necessário para detectar os bugs no *software* e para testar se o *software* atende aos requisitos do cliente. Isso ajuda a equipe de desenvolvimento a corrigir os erros e entregar um produto de boa qualidade.

Há vários pontos no processo de desenvolvimento de *software* em que o erro humano pode levar a um *software* que não atende aos requisitos dos clientes. Alguns deles estão listados abaixo.

- O cliente / pessoa que fornecer os requisitos, em nome da organização do cliente, pode não saber exatamente o que é necessário ou pode esquecer de fornecer alguns detalhes. Isto eventualmente resultará em falta de recursos no *software*.
- A pessoa que está coletando os requisitos pode interpretar erroneamente ou perder completamente um requisito quando documentá-los.
- Se houver problemas durante a fase de projeto, isso pode levar a erros no futuro.
- *Bugs* podem ser introduzidos durante a fase de desenvolvimento por erro humano, falta de experiência, etc.
- Os testadores podem perder erros durante a fase de testes devido a erro humano, falta de tempo, experiência insuficiente, etc.
- Os clientes podem não ter a largura de banda para testar todos os recursos do produto e podem liberar o produto para seus usuários finais, o que pode levar estes a encontrarem erros no *software*.

O negócio e reputação de organizações depende da qualidade de seus produtos e, em alguns casos, até a receita pode depender das vendas de produtos de *software*. Os usuários podem preferir

comprar um produto concorrente se um produto da empresa tiver qualidade ruim. Isso pode resultar em perda de receita para a organização. No mundo de hoje, a qualidade é uma das principais prioridades de qualquer organização.

2.5. Atividade

- Organizar grupos com até 5 estudantes.
- Cada um dos grupos deverá responder às seguintes perguntas:
  1. **Explique por que um programa não precisa, necessariamente, ser completamente livre de defeitos antes de ser entregue ao cliente.**
  2. **Explique melhor a afirmação que diz que os testes podem detectar apenas a presença de erros e não a sua ausência.**
  3. **Acesse os links a seguir e escolha qual desses erros foi, em sua opinião, o mais preocupante.**: (i) What is Software Testing? (ii) 5 Most Embarrassing Software Bugs in History.

Os grupos terão 30 minutos para a discussão e elaboração das repostas. No fim, todos deverão compartilhar suas respostas. Essa atividade vale 0,25 pontos na nota final.

3. Níveis de teste de acordo com o SWEBOK

3.1. Introdução

Esse capítulo tem o objetivo de apresentar os níveis de teste de acordo com o *Guide to the Software Engineering Body of Knowledge* (SWEBOK) [SWEBOK] (Guia para a Base de Conhecimento de Engenharia de *Software*), fornecendo assim uma ideia geral sobre os conceitos relacionados ao teste de *software*.

O SWEBOK é um guia criado pelo *Institute of Electrical and Electronics Engineers* - IEEE (Instituto de Engenheiros Elétricos e Eletrônicos) que trata de vários aspectos sobre a Engenharia de *Software*. O capítulo 4 de sua terceira e última edição é dedicado a testes de *software*. Cumpre ressaltar que essa obra informa que o teste de *software* geralmente é realizado em diferentes níveis ao longo dos processos de desenvolvimento e manutenção. Os níveis podem ser distinguidos tomando como base o objeto que será testado. Este varia conforme (i) o escopo, que é chamado de alvo, ou (ii) finalidade, que é chamado de objetivo.

3.1.1. Como esse capítulo está estruturado

O presente capítulo possui duas seções, a saber:

- Em Alvo do teste são apresentados os conceitos sobre os níveis de testes, em que é verificada uma parte do *software* que está sendo desenvolvido. O nível é diferente conforme o escopo.
- Objetivos do teste é responsável por falar sobre os níveis de testes usados quando temos um objetivo em vista, como usabilidade ou performance.
- Técnicas para o Teste de *Software*
- Medidas de Teste de *Software*
- Avaliação dos Testes Realizados (?)

3.2. Alvo do teste

O alvo do teste está relacionado ao escopo, podendo ser um:

- único módulo;
- grupo desses módulos (relacionados por finalidade, uso, comportamento ou estrutura);
- ou um sistema inteiro.

Podemos criar três estágios ou níveis de teste. Esses três estágios de teste não implicam em nenhum modelo de processo e nenhum deles é considerado mais importante que os outros.

3.2.1. Testes de unidade / unitários

Os testes de unidade, comumente chamados de testes unitários, são responsáveis por verificar o funcionamento de elementos de *software* de maneira isolada. Ou seja, eles são testados individualmente. O teste de unidade focaliza o esforço de verificação na menor unidade do projeto

de *software* — o componente ou módulo de *software*. Se estivermos escrevendo um código usando uma linguagem como o Java por exemplo, serão testados os métodos das classes, uma vez que eles normalmente são as menores unidades do projeto. O teste de unidade, na maioria das vezes, ocorre com acesso ao código fonte que está sendo testado e com o suporte de ferramentas de depuração. Os programadores que escreveram o código frequentemente, mas nem sempre, realizam testes unitários. O *Test Driven Development* (TDD) ou Desenvolvimento Guiado por Testes, um exemplo de teste unitário, é apresentado na seção a seguir.

Desenvolvimento Guiado por Testes (TDD, do inglês *Test Driven Development*)

O TDD é uma abordagem para o desenvolvimento de programas em que se intercalam testes e a escrita de código-fonte. O código evolui de maneira incremental, juntamente com um teste para esse incremento. Importante: **Nada é feito, ou seja, nenhum novo código é escrito, sem que aquilo que está sendo testado passe no teste.** A figura O ciclo do TDD, mostrada a seguir, que foi copiada do livro de Engenharia de *Software* de Ian Sommerville[SOMMERVILLE], fornece uma visão geral sobre o processo do TDD.

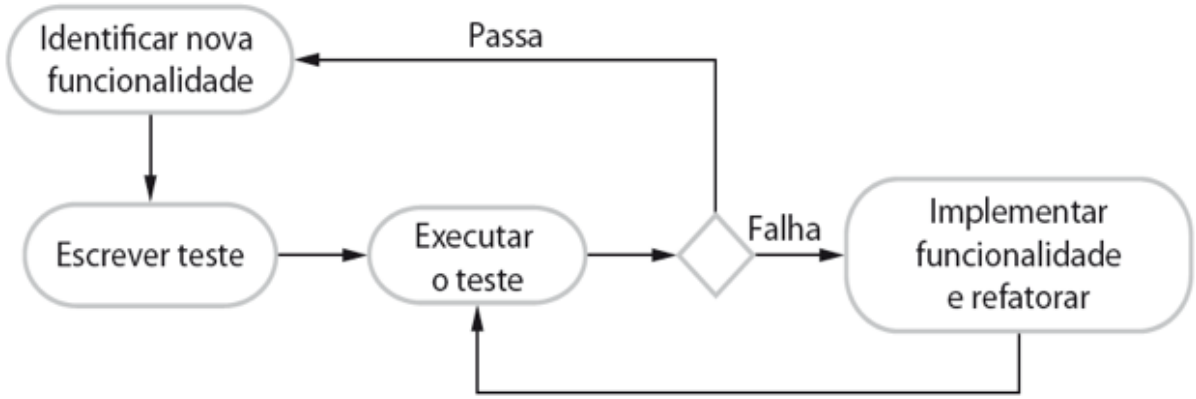

Figura 3. O ciclo do TDD

O ciclo de desenvolvimento usando o TDD envolve, antes de tudo, uma mudança no paradigma ao qual o programador está acostumado. Ao invés de começar a escrever a funcionalidade a ser incrementada, ele escreve um teste. A sequência de passos do TDD pode ser explicada assim:

1. Identificar o incremento necessário. Ele deve ser pequeno e implementável em poucas linhas de código. Ele não será escrito ainda;
2. Escrever um teste para esse incremento. Esse teste será feito por uma ferramenta própria de testes, como o *JUnit*. Essa ferramenta é que fará o teste e relatará se houve sucesso ou não.
3. Executar o teste. A ferramenta poderá também refazer todos os testes que já foram feitos anteriormente. Nesse caso, o resultado obtido é uma falha do teste. Mas por que? Lembre-se que você ainda não implementou a nova funcionalidade. Essa falha o ajudará a não esquecer de fazer a codificação.
4. Implementar a nova funcionalidade e executar o teste novamente. O código pode ser refatorado para melhorá-lo.
5. Depois que os testes forem executados com sucesso, voltamos ao passo 1.

Benefícios do TDD

Cobertura de código

Cada segmento de código que você escreve deve ter pelo menos um teste associado.

Testes de regressão

Um conjunto de testes de regressão é desenvolvido de forma incremental enquanto um programa é desenvolvido.

Depuração simplificada

Quando um teste falhar, deve ser óbvio onde está o problema. O código recém-escrito tem de ser verificado e modificado.

Documentação de sistema

Os próprios testes são uma forma de documentação que descreve o que o código deve estar fazendo.

3.2.2. Testes de integração

O teste de integração, também conhecido como teste de componente, é o processo de verificar as interações entre os componentes de *software*. Ele fará com que duas ou mais classes, por exemplo, sejam postas em funcionamento juntas. Devemos pensar que se individualmente elas funcionaram, quando colocadas juntas, elas devem continuar funcionando. Estratégias clássicas de teste de integração, como *top-down* 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"*.

3.2.3. Teste de sistema

O teste do sistema está preocupado em testar o comportamento de um sistema inteiro, definido pelo escopo de um projeto ou programa de desenvolvimento. De acordo com o ISTQB, no teste de sistema, o ambiente de teste deve corresponder o máximo possível ao objetivo final ou ao ambiente de produção. Isto visa minimizar os riscos de falhas específicas de ambiente não serem encontradas durante o teste. Ele pode ser baseado em descrições de alto nível do comportamento do sistema, tais como: (i) especificac a o de riscos e/ou de requisitos, (ii) processos de negócios ou (iii) casos de uso.

O teste do sistema é geralmente considerado apropriado para avaliar os requisitos não funcionais do sistema como segurança, velocidade, precisão e confiabilidade. Interfaces externas para outros aplicativos, utilitários, dispositivos de *hardware* ou os ambientes operacionais também são geralmente avaliados nesse nível. Uma equipe de teste independente é frequentemente responsável pelos testes de sistema.

3.3. Objetivos do teste

Quando tratamos de objetivos do teste, estamos nos referindo a uma característica específica. Ela pode sofrer alterações conforme o *software* é testado. Segundo o SWEBOK, declarar os objetivos do teste em termos precisos e quantitativos permite que os resultados obtidos possam ser medidos, além de dar mais controle ao 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. Isto é referido na literatura como testes de conformidade, testes de correção ou testes funcionais. No entanto, várias outras propriedades emergentes também podem ser testadas, incluindo desempenho, confiabilidade, entre muitas outras.

Outros objetivos importantes para o teste incluem, mas não se limitam a, identificação de vulnerabilidades de segurança, avaliação de usabilidade e aceitação de *software*. Para estes objetivos, diferentes abordagens podem ser 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.

O SWEBOK enumera 13 tipos de testes na categoria de objetivos do teste. Os tipos elencados a seguir são os mais usados na literatura. Observe que alguns são mais apropriados, por exemplo, para:

- pacotes de *software* personalizados, como os testes de instalação;
- e outros para produtos de consumo, como os testes beta.

3.3.1. Teste de Aceitação / Qualificação

O teste de aceitação / 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 representante de um cliente, portanto, especifica ou realiza atividades diretamente para verificar se seus requisitos foram atendidos. No caso de um produto de consumo,ele verifica se a organização atendeu aos requisitos estabelecidos para o mercado-alvo.

O objetivo desse teste é estabelecer a confianc a no sistema, parte do sistema ou uma caracteri stica não específica do sistema. Procurar defeitos não é o principal foco. Ele pode avaliar a disponibilidade do sistema para entrar em produção. Este não é necessariamente o último nível de teste, uma vez que, por exemplo, um teste de integração em larga escala pode ser feito posteriormente.

Esse teste é realizado pelo cliente ou por usua rios do sistema; os interessados (*stakeholders*) também podem ser envolvidos.

3.3.2. Testes de Instalação

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

3.3.3. Teste Alfa e Beta

Antes do lançamento, o *software* as vezes é fornecido a (i) um grupo pequeno e selecionado de usuários em potencial para uso experimental (teste alfa) e/ou (ii) 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.

3.3.4. Teste de Conquista e Avaliação de Confiabilidade

Este tipo de teste melhora a confiabilidade do sistema, identificando e corrigindo falhas. Além disso, medidas estatísticas de confiabilidade podem ser obtidas gerando aleatoriamente casos de teste de acordo com o perfil operacional do *software* (consulte Perfil Operacional na seção 3.5, Técnicas Baseadas em Uso). Esta última abordagem é chamada de teste operacional. Usando modelos de crescimento de confiabilidade, ambos os objetivos podem ser perseguidos juntos (ver Teste de Vida, Avaliação de Confiabilidade na seção 4.1, Avaliação do Programa em Teste).

3.3.5. Teste de Regressão

Um teste de regressão mostra que o *software* ainda passa nos testes feitos anteriormente (na verdade, às vezes também é chamado de teste de não-regressão). 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 (i) a garantia dada pelo teste de regressão, toda vez que uma alteração é feita, e (ii) os recursos necessários para executar os testes de regressão. Esta compensação pode ser bastante demorada 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. 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.

3.3.6. Teste de Performance

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

3.3.7. Testes de Segurança

O teste de segurança é focado em verificar se 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).

Exemplos de testes de segurança são os chamados Testes de Penetração (*Penetration Test, PenTest*).

3.3.8. Teste de Estresse

O teste de estresse executa o *software* na carga máxima do projeto, com o objetivo de determinar os limites comportamentais e testar os mecanismos de defesa em sistemas críticos.

3.3.9. Teste back-to-back [7]

O padrão IEEE/ISO/IEC 24765 define o teste *back-to-back* como “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”.

3.3.10. Teste de Recuperação

O teste de recuperação visa verificar os recursos de reinicialização do *software* após uma falha do sistema ou outro "desastre". Isto visa medir a capacidade de resiliência do software, ou seja, de recuperação após uma falha. Normalmente neste tipo de teste, falhas são intencionalmente injetadas no ambiente em que o sistema está sendo testado, para avaliar o comportamento do *software* nestes casos.

3.3.11. Teste de Interface

Os defeitos da 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.

3.3.12. Teste de Configuração

Nos casos em que o *software* é criado para atender a diferentes usuários, o teste de configuração verifica o *software* em diferentes configurações especificadas.

3.3.13. Teste de Usabilidade e Interação Homem-Computador

A principal tarefa dos testes de usabilidade e interação entre o homem e computador é avaliar como é fácil para os usuários finais aprenderem a usar o *software*. Em geral, pode envolver o teste das funções do *software* que suportam as tarefas do usuário, a documentação para auxílio aos usuários e a capacidade do sistema de se recuperar de erros humanos (consulte Design da interface do usuário no *Software Design KA*).

=x=x=x=x=x=x=x

3.4. Testes de Desenvolvimento

Os testes de desenvolvimento representam todos os testes que são realizados pelos desenvolvedores de um sistema. Nesse caso, o testador é o próprio desenvolvedor ou um membro da equipe. Um exemplo são os testes unitários.

3.5. Teste de Aceitação

O teste de aceitação ou de aceite frequentemente é realizado pelo cliente ou por um usua rio do sistema; os interessados (*stakeholders*) também podem ser envolvidos. O objetivo desse teste é estabelecer a confianc a no sistema, parte do sistema ou em uma caracteri stica não específica do sistema. Procurar defeitos não é o principal foco. Ele pode avaliar a disponibilidade do sistema para entrar em produção. Este não é necessariamente o último nível de teste, uma vez que, por exemplo, um teste de integração em larga escala pode ser feito posteriormente. As formas de teste de aceite incluem tipicamente os seguintes:

- Teste de aceitação pelo usuário
- Teste operacional de aceite
- Teste de aceite de contrato e regulamento
- Teste Alfa e Beta (ou teste no campo)

Teste Alfa

Realizados pelos usuários - testes manuais. São testes realizados em um ambiente controlado pelo desenvolvedor que registra os problemas de uso e os erros que aconteceram.

Teste Beta

Realizados pelos usuários mais usuários - testes manuais. Os testes são feitos no ambiente do usuário. São mais difíceis para o desenvolvedor acompanhar, uma vez que pode haver uma quantidade muito grande de usuários.

3.6. Estratégias de teste

Segundo Roger Pressman [PRESSMAN], há várias estratégias de testes existentes e elas fornecem as seguintes características genéricas:

1. As revisões formais são feitas no início.

2. O teste começa no nível de componente e prossegue "para fora", em direção à integração de todo o sistema.
3. Diferentes técnicas de teste são adequadas em diferentes momentos.

3.7. Técnicas para o Teste de *Software*

3.8. Medidas de Teste de *Software*

3.9. Avaliação dos Testes Realizados (?)

4. Técnicas de Teste

4.1. Introdução

Esse capitulo vai ...

4.2. O que são técnicas de teste

Um dos objetivos do teste é detectar tantas falhas quanto possível. As técnicas de teste fornecem as diretrizes sistemáticas para que sejam projetados testes eficientes. Testes eficientes são aqueles que conseguem:

- exercitar a lógica interna e as interfaces de cada componente de *software*
- verificam as entradas e saídas dos programas usando dados um conjunto de dados de teste, de maneira que possam ser encontrados erros em funções ou métodos, comportamento indesejado ou desempenho inadequado.

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 pelo SWEBOK 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 do software. Essas técnicas também pode ser referenciadas como testes estruturais e testes funcionais respectivamente.

As técnicas de teste fornecem as diretrizes sistemáticas para que sejam projetados testes eficientes. Testes eficientes são aqueles que conseguem:

- exercitar a lógica interna e as interfaces de cada componente de *software*
- verificam as entradas e saídas dos programas usando dados um conjunto de dados de teste, de maneira que possam ser encontrados erros em funções ou métodos, comportamento indesejado ou desempenho inadequado.

A lista a seguir inclui as técnicas de teste que são comumente usadas, mas alguns profissionais dependem de algumas técnicas mais do que outras.

- Caixa Branca - Este teste também é chamado de teste de caixa vidro. Segundo [PRESSMAN], ele é uma filosofia de projeto de casos de teste que usa a estrutura de controle descrita como parte do projeto ao nível de componentes para derivar casos de teste. Usando métodos de teste caixa-branca, o engenheiro de software pode derivar casos de testes que

    - garantem que todos os caminhos independentes de um módulo tenham sido exercitados pelo menos uma vez
    - exercitem todas as decisões lógicas em seus lados verdadeiro e falso
    - executam todos os ciclos nos seus limites e dentro de seus intervalos operacionais
    - exercitem as estruturas de dados internas para garantir a sua validade.

(caixa aberta - visualiza o código fonte - Teste de unidade, testes estáticos (análise do código, sem executar-lo)) Dentro dessa categoria de teste de software, o testador tem acesso ao código fonte da aplicação e pode construir códigos para efetuar linker(?) de componentes. Essa técnica adota critérios para a geração dos casos de teste com a finalidade de encontrar defeitos nas estruturas internas no software, através de situação que exercitem adequadamente todas as estruturas utilizadas na codificação.

Esse tipo de teste é desenvolvido analisando-se o código-fonte e elaborando os casos de teste que possam cobrir todas as possibilidades do programa.

DEssa maneira, todas as variações originadas por estruturas de condições são testadas. Um exemplo prático desse teste e o JUnit.

O analista tem acesso ao código fonte, conhece a estrutura interna do produto sendo analisado e possibilita que sejam escolhidas partes específicas de um componente para serem avaliadas. Esse tipo de teste, também conhecido como teste estrutural, é projetado em função da estrutura do componente e permite uma averiguação mais precisa do comportamento dessa estrutura. Perceba que o acesso ao código facilita o isolamento de uma função ou ação, o que ajuda na análise comportamental das mesmas.

- Teste de métodos e Classes, Testes de comando de repetição, teste de condições
    - Teste de cobertura
    - Teste de Caminhos
    - Teste de comandos
    - Teste de condições
        - Caixa Cinza - uma definição deste tipo de teste seria um ponto de equilibro virtual entre o teste de caixa-branca e o caia-preta.
        - Caixa-preta - Neste tipo de teste de software, o desenvolvedor dos testes não possui acesso ao código fonte do programa. Técnica de teste que adota critérios para a geração dos casos de teste com a finalidade de garantir que os requisitos funcionais do software que foi construído sejam plenamente atendidos. Verifica se o resultado gerado por esses requisitos estão de acordo com o esperado.

Para essa categoria podem ser levados em consideração todos os eventos que podem ser disparados pelo usuário, como por exemplo, cada clique de mouse a ser realizado em uma interface.

O analista não tem acesso ao código fonte e desconhece a estrutura interna do sistema. É também conhecido como teste funcional, pois é baseado nos requisitos funcionais do software. O foco, nesse caso, é nos requisitos da aplicação, ou seja, nas ações que ela deve desempenhar.

Para mostrar quais problemas que esse tipo de teste rastreia, podemos citar alguns exemplos:

1. Data de nascimento preenchida com data futura;
2. Campos de preenchimento obrigatório que não são validados;
3. Utilizar números negativos em campos tipo valor a pagar;
4. Botões que não executam as ações devidas;
5. Enfim, todo tipo de falha funcional, ou seja, falhas que contrariam os requisitos da aplicação.

Exemplos: - Teste de valor limite - Teste de classe de equivalência

Há que se destacar, contudo, que existe um elemento comum aos dois tipos de teste. Tanto no teste de caixa branca quanto no teste de caixa preta, o analista não sabe qual será o comportamento da aplicação ou do alvo de teste em uma determinada situação. A imprevisibilidade de resultados é comum aos dois casos.

- Caixa Preta (não vejo o interior - é baseado em entradas e saídas - Teste de integração, Teste de sistema, Teste de Aceitação, Teste Alfa, Teste Beta) Teste baseado em entradas e saídas de Cenários Macro
    - Teste baseado em cenários
    - Teste baseado em Casos de uso
    - Análise de Valores limites

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: os testes são obtidos 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. visa explorar o sistema sem nenhum planejamento definido, sem um resultado esperado, neste caso os bugs são descobertos por acaso Imagine você tendo que entender um sistema de grande porte para criar e realizar os testes em um curto espaço de tempo. Nesse caso, seriam inviáveis os planejamentos e documentação. Então, nada melhor do que utilizar testes ad-hoc, a não ser que o testador não tenha experiência suficiente para isso. Não é porque o mesmo não possui planejamento que qualquer um poderá fazê-lo com eficiência. O teste ad-hoc exige muito mais experiência e criatividade do testador, porém é mais passível a insucessos.

3.1.2. Teste exploratório O teste exploratório é definido como aprendizado simultâneo, projeto de teste e execução de testes [6, parte 1]; 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 software o conhecimento do engenheiro, 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 falhas e falhas, o risco associado a um produto específico etc. Todo o planejamento é pré definido assim como o tempo para a realização do mesmo, neste caso, o entendimento do sistema fica muito mais evidente e os erros encontrados são bem entendidos Leia mais aqui:

Testes A/B :cap: Capitulo5 :imagesdir: imagens/Capitulo4/

5. Quais são os tipos de teste

Tipos de Teste: o alvo do teste

Um grupo de atividades de teste pode ser direcionado para verificar o sistema (ou uma parte do sistema) com base em um motivo ou alvo específico. Cada tipo de teste tem foco em um objetivo particular, que pode ser o teste de uma funcionalidade, a ser realizada pelo software; uma característica da qualidade não funcional, como por exemplo a confiabilidade ou a usabilidade, a estrutura ou arquitetura do software ou sistema; ou ainda mudanças relacionadas, ex.: confirmar que os defeitos foram solucionados (teste de confirmação) e procurar por mudanças inesperadas (teste de regressão). Modelos do software podem ser elaborados e/ou usados no teste estrutural ou funcional. Por exemplo, para o teste funcional, um diagrama de fluxo de processo, um diagrama de transição de estados ou uma especificação do programa, e para teste estrutural um diagrama de controle de fluxo ou modelo de estrutura do menu.

- Teste de funcionalidade
- Teste de interface
- Teste de desempenho
- Teste de usabilidade
- Teste de segurança
- Teste usuário

5.1. Teste de regressão (reteste)

Cada vez que um novo módulo é adicionado como parte do teste de integração, o *software* se modifica. Novos caminhos de fluxo de dados são estabelecidos, nova E/S pode ocorrer e nova lógica de controle é acionada. Assim sendo, tudo o que havia sido previamente testado corre o risco de apresentarem problemas. Esse tipo de teste ajuda a garantir que modificações não introduzam algum comportamento indesejável ou erros adicionais.

5.2. Testes de usabilidade

Os testes devem ser realizados independentemente do tamanho do projeto e da estrutura disponível, pois sempre revelam aspectos que influenciarão com mais ou menos profundidade o produto final. Os principais objetivos dos testes são:

- Permitir que cada usuário realize a tarefa a que se propõe ao usar a interface, em um tempo razoável. Se a utilização é fácil, precisa, auto-explicativa, relativamente rápida, mas não atende a uma necessidades clara, não tem muita utilidade. Ou valor.
- Tornar o uso da interface o mais intuitivo possível. Quanto menos tempo o usuário leva para realizar seu objetivo no website, maior e seu grau de satisfação com a interface.
- Verificar a atitude positiva ou negativa do usuário durante a experiência de uso. Neste caso, “atitude” se refere a percepções, sentimentos, opiniões do usuário, que podem ser verificadas por meio de entrevistas orais ou escritas.

As pessoas tendem a realizar melhor as suas tarefas e objetivos ao usar uma interface quando esta os agrada de maneira geral e lhes é familiar. Estabelecer consenso na equipe de projeto ou manutenção evolutiva sobre os resultados esperados. Os testes podem diminuir as dúvidas e discordâncias sobre as soluções e decisões adotadas. Os objetivos de testes como os citados acima podem ser baseados em aspectos quantitativos, mas não se resumir a estatísticas sobre o uso e a satisfação do uso. É importante também considerar aspectos qualitativos, mais subjetivos, que compõem quadros mais completos do contexto de uso.

Os testes de usabilidade fazem com que o desenvolvedor/testador fique junto ao usuário. O objetivo é aprender como ele realmente usa seu produto. O desenvolvedor escolhe algumas tarefas que ele precisa realizar e assiste e registra ele os locais em surgiram algum tipo de dificuldades. Este teste ajuda a criar hipóteses de melhoria do produto.

5.3. Testes de usuário

São testes onde os usuários ou clientes usam o *software* a fim de fornecer um *feedback*. Assim, eles experimentam o *software* para ver se gostam desse produto e verificam também se ele está em conformidade com a suas necessidades.

De modo geral, os testes de usuário ajudam a verificar se a interface permite o uso fácil e intuitivo, se provê funcionalidades que os usuários valorizam e se proporciona, de modo geral, uma experiência de uso satisfatória.

O teste de usuário é essencial, mesmo em sistemas abrangentes ou quando testes de release tenham sido feitos. O motivo é que a influencia realizada pelo ambiente de trabalho do usuário interfere muito sobre a confiabilidade, o desempenho, a usabilidade e a robustez de um sistema, tendo em vista que para o desenvolvedor é praticamente impossível replicar o ambiente de trabalho em que todos os possíveis usuários estarão.

Sobre o teste de aceitação é importante lembrar que os clientes querem usar o *software* assim que possível por causa dos benefícios que podem ser obtidos. Os testes de aceitação podem não terem obtido um bom resultado mas devido a uma séria de outros fatores, a adoção do produto pode começar mesmo assim.

Podemos separar os testes de usuário em três categorias:

- Teste Alfa
- Teste Beta
- Teste de aceitação

Os testes de aceitação possuem seis estágios a saber:

1. Definir critérios de aceitação
2. Planejar os testes de aceitação
3. Derivar testes
4. Executar testes
5. Negociar resultados dos testes

6. Rejeitar / aceitar o sistema

[testeAB] | *testesAB.png*

Figura 4. A mountain sunset

As pesquisas atitudinais são focadas no que as pessoas falam que acreditam (por exemplo, ao responderem um formulário online ou em uma conversa dentro de um grupo focal (*focus group*)), enquanto as pesquisas comportamentais analisam o que as pessoas fazem (por exemplo, em um teste de usabilidade, ou em testes A/B).

https://brasil.uxdesign.cc/muito-além-do-teste-de-usabilidade-os-vários-tipos-de-pesquisas-com-usuários-em-ux-b91a6e15bc61

Referencias

- [PRESSMAN] PRESSMAN, Roger S. **Engenharia de Software-8ª Edição (2006)**. Ed. Mc Graw Hill, 2006.
- [SOMMERVILLE] SOMMERVILLE, Ian. **Engenharia de Software-9ª Edição (2011)**. Ed Person Education.
- [SWEBOK] BOURQUE, Pierre et al. ***Guide to the software engineering body of knowledge (SWEBOK ®): Version 3.0***. IEEE Computer Society Press, 2014.