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 software, seus níveis, seus objetivos, como os testes são projetados e executados. De forma esse material 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 que é 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 a :ementa da disciplina que está em vigor, que é mostrada a seguir:

- Conceitos básicos relacionados a teste de software;
- Processos de testes;
- Ferramentas para planejamento, elaboração e automatização 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 capitulo As atividades do processo de *software* e os testes tem o objetivo de posicionar as atividades de teste de software dentre as demais atividades básicas do ciclo de desenvolvimento

de software. 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 capitulo Uma visão geral sobre o que é teste de *software* apresenta o conceito sobre teste e reforça sua importância na obtenção de um software 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 capitulo 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 e realizar algumas conexões entre essa disciplina e outras do curso, sobretudo Analise de Sistemas, Projeto de Sistemas, Programação para WEB II.

1.1.1. Como esse capitulo está estruturado

O presente capitulo 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 e discute brevemente sobre quando os testes devem começar.
- A seção 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-lo 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 do sistema e implementação do sistema;
- verificação e validação – checagem de que o sistema faz o que o cliente deseja;
- evolução – evolução em resposta a 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 que dificilmente são tratadas em separado dentro do mundo da computação e que têm como objetivo a conversão da especificação do sistema em um sistema executável. Em projeto criamos um *Design* de uma estrutura de *software* que possa materializar a especificação; e por implementação

devemos entender que é o ato de transformar o *Design* 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 ainda que ele 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* ocorre quando é necessário alterar um sistema que já existe, tornando-o adequado a 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 torna-lo útil em pouco tempo. Não há disciplina que específica no Curso de Sistema para Internet do IFTO e provavelmente em nenhum outro curso que lide diretamente com essa questão.

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 projeto e implementação e de 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

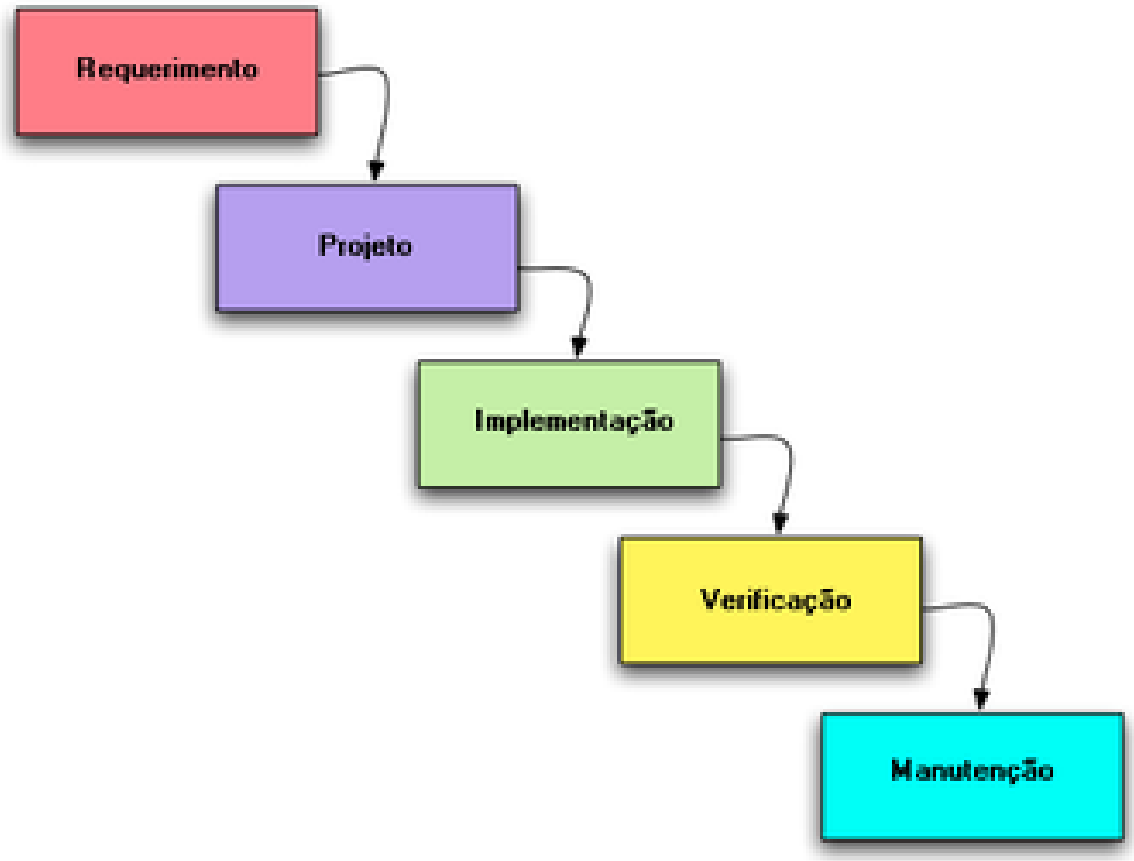

Figura 1. Exemplo do modelo em cascata

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 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 é possível que sejam executados de 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 que será aumentado em versões posteriores. Em situações como essa, um modelo incremental pode ser escolhido.

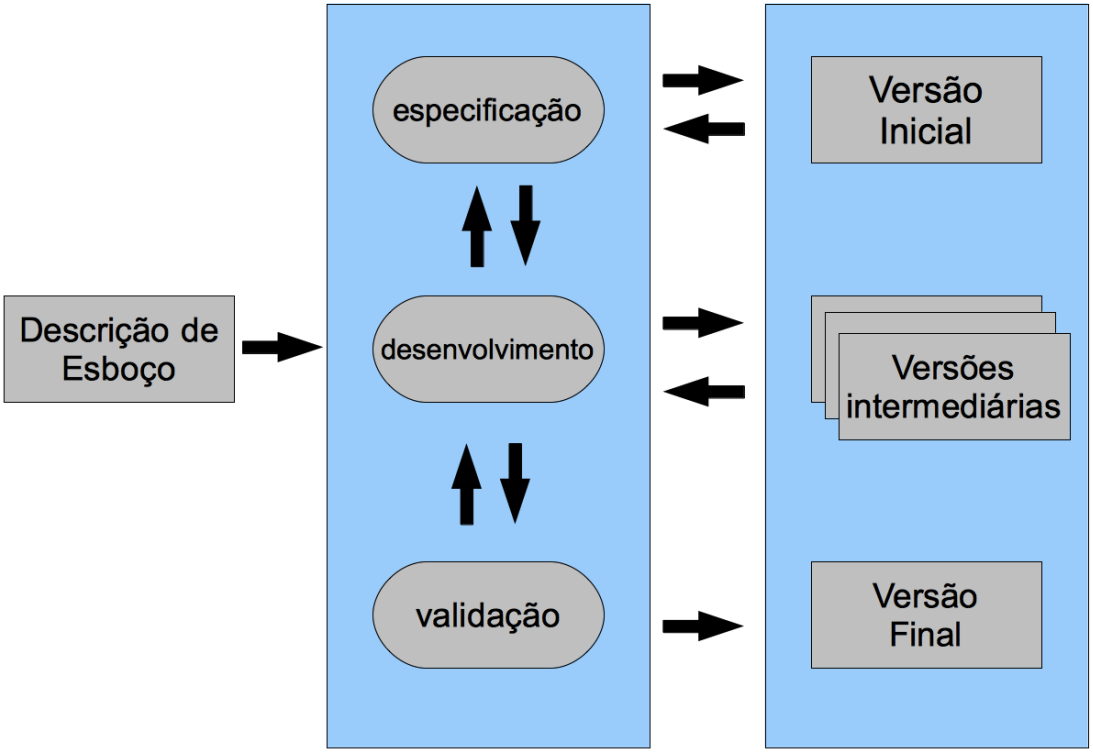

Figura 2. Exemplo do modelo incremental

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 parir do esboço é possível especificar os requisitos, aplicar técnicas de projeto e 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. Teste 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, e os planos e procedimentos de teste devem ser sistematicamente e continuamente desenvolvidos - e possivelmente refinados - à medida que o desenvolvimento de software prossiga. Essas atividades de planejamento de teste e projeto de teste fornecem informações úteis para os projetistas de software e ajudam a destacar possíveis fraquezas, como omissões / contradições de design ou omissões / ambiguidades na documentação.

1.4. Relação entre essa disciplina e outras disciplinas no curso

1.4.1. Relação existente entre análise de sistema e os 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 é durante a fase de análise é que essa relação é feita. Conforme dito anteriormente, o planejamento testes pode ter inicio 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 os testes

In particular, unit and integration testing are intimately related to software construction, if not part of it.

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

2.1. Introdução

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

2.1.1. Como esse capitulo está estruturado

O presente capitulo 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 vocês está tendo seu primeiro contato com testes.
- A seção Quais são os objetivos do teste de *software*? é a responsável por apresentar qual é 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 como que 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)**, sendo que 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 os requisitos do cliente.

Segundo Ian Sommerville [SOMMERVILLE], o teste de *software* é uma etapa dentro do ciclo de vida que **tem como objetivo descobrir os defeitos que um programa possui antes que ele entre em produção**, isto é antes de 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 o teste é um processo independente, pois nem sempre quem desenvolve testará

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

1. Envolve processos de inspeção e revisão, e testes do sistema.
2. Testes do sistema envolvem executar o sistema com casos de teste. 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 . Há autores como [PRESSMAN] que afirmam que dentro da comunidade de Engenharia de Software, esse 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, alterando um pouco o significado de erro e falha e criando uma espécie de graduação:

http://josebarbosa.com.br/?p=463 https://glossary.istqb.org/search/defect

- 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 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 completos possíveis, mas eles contam com uma quantidade mínima de tempo e deve 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 e a 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.

▶ https://www.youtube.com/watch?v=gp_D8r-2hwk *(YouTube video)*

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, o que pode levar à falta de recursos.
- A pessoa que está coletando os requisitos pode interpretar erroneamente ou perder completamente um requisito quando documentá-los.
- Durante a fase de design, se houver problemas no design, isso pode levar a erros no futuro
- *Bugs* podem ser introduzidos durante a fase de desenvolvimento durante o 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 os usuários finais a encontrar erros no aplicativo

Um 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 em um produto que tenha qualidade ruim e 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, necessáriamente, ser completamente livre de livre de defeitos antes de ser entregue aos seus 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** https://www.guru99.com/software-testing-introduction-importance.html https://www.scientificamerican.com/article/pogue-5-most-embarrassing-software-bugs-in-history/
- Os grupos terão 30 minutos para a discussão e elaboração das repostas.
- No fim, todos os grupos deverão compartilhar suas respostas. Essa atividade vale 0,25 pontos na nota final.

3. Quais os estágios de teste?

O *Guide to the Software Engineering Body of Knowledge* (SWEBOK) [SWEBOK] é um guia criado pelo IEEE e que trata de vários aspectos sobre a Engenharia de Software. Ele possui um capitulo dedicado ao testes que nos informa que o teste de software é geralmente 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, que é chamado de alvo, ou na finalidade, que é chamado de objetivo.

O alvo do teste pode variar: um único módulo, um 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 dois.

Segundo Ian Sommerville[SOMMERVILLE], todo software deve passar por três estágios de teste.

1. Testes de desenvolvimento
2. Testes de integração
3. Testes de usuário

Apesar dessa divisão ser simples, podemos criar outras categorias de estágio. Para cada ni vel de teste, os seguintes aspectos podem ser identificados: seus objetivos genéricos, os produtos de trabalho utilizados como referência para derivar os casos de testes (ex.: base do teste), o objeto do teste (o que esta sendo testado), defeitos e falhas típicas a se encontrar, testes (“harness”) e ferramentas de suporte e abordagens e responsabilidades específicas. A seguir cada um dos estágios é tratado com mais detalhes e as suas subdivisões são apresentadas.

3.1. 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.

3.1.1. Testes de unidade / unitário

Os testes de unidade, muitas vezes chamados de testes unitários, são responsáveis por verificar o funcionamento de elementos de software de maneira isolada, ou seja eles são testados de forma individual separadamente. 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 TDD (do inglês *Test Driven Development*) ou Desenvolvimento Guiado por Testes, que é um exemplo de teste unitário, é apresentado na seção a seguir.

Esse material pode ajudar. Dar uma olhada !! https://edisciplinas.usp.br/pluginfile.php/384739/mod_resource/content/1/Aula%205_2014_Tipos-de-teste-software-v2.pdf

http://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/testes-software_v01.pdf

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.

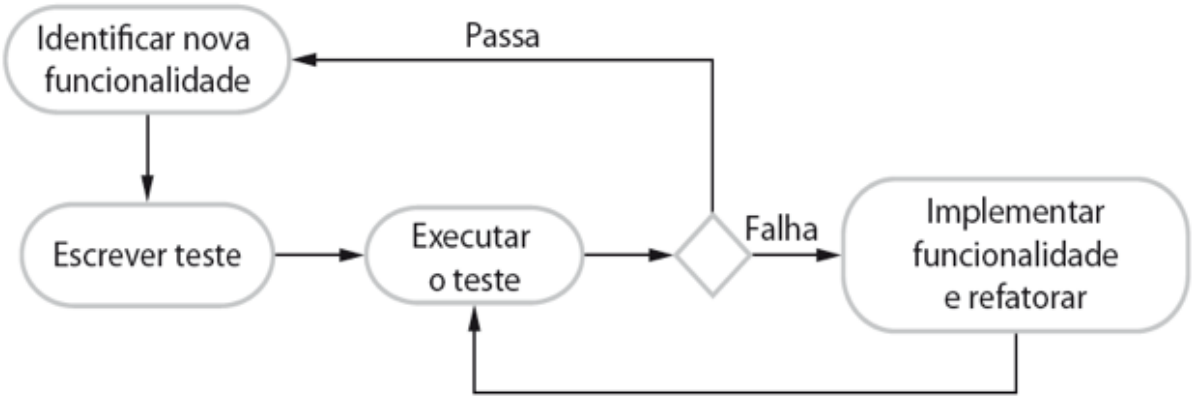

Figura 3. O ciclo do TDD

O ciclo de desenvolvimento usando o TDD envolve antes de tudo uma mudança no paradigma ao qual o programador está acostumando. Ao invés de começar a escrever a funcionalidade a ser incrementada, ele escreve um teste. A sequencia 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*. Esse ferramenta é quem 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 e melhorado completamente.
5. Depois que os testes forem executados com sucesso, voltamos ao passo 1.

Beneficios do TDD

Cobertura de código

Cada segmento de código que você escreve deve ter pelo menos um teste associado, para todo o código escrito tem pelo menos um teste.

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.

3.1.2. 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"*.

3.1.3. 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.

3.2. 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 o ambiente de produção, para minimizar que 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 especificac a o de riscos e/ou de requisitos, processos de negócios ou casos de uso. O teste do sistema é geralmente considerado apropriado para avaliar os requisitos não funcionais do sistema, ou seja a segurança, a velocidade, precisão e também a 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 pelo teste de sistema.

3.3. Teste de aceitação

O teste de aceitação ou de aceite frequentemente é realizado pelo cliente ou por 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 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, apesar de não ser necessariamente o último nível de teste, uma vez que, por exemplo, um teste de integração em larga escala pode ser feito após. 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
- Alfa e Beta Teste (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. Mais mais difícil para o desenvolvedor acompanhar uma vez que podem haver uma quantidade muito grande de usuários.

4. Técnicas de Teste

4.1. Introdução

Esse capitulo vai ...

4.2. O que são técnicas de teste

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.
    - Caixa Branca (caixa aberta - visualiza o código fonte - Teste de unidade, testes estáticos (análise do código, sem executar-lo))
        - 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 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

Testes A/B

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 especi fico. Cada tipo de teste tem foco em um objetivo particular, que pode ser o teste de uma funcionalidade, a ser realizada pelo software; uma caracteri stica da qualidade na o funcional, tal como a confiabilidade ou usabilidade, a estrutura ou arquitetura do software ou sistema; ou mudanc as relacionadas, ex.: confirmar que os defeitos foram solucionados (teste de confirmac a o) e procurar por mudanc as inesperadas (teste de regressa o). 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 transic a o de estados ou uma especificac a o do programa, e para teste estrutural um diagrama de controle de fluxo ou modelo de estrutura do menu.

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 especi fico. Cada tipo de teste tem foco em um objetivo particular, que pode ser o teste de uma funcionalidade, a ser realizada pelo software; uma caracteri stica da qualidade na o funcional, tal como a confiabilidade ou usabilidade, a estrutura ou arquitetura do software ou sistema; ou mudanc as relacionadas, ex.: confirmar que os defeitos foram solucionados (teste de confirmac a o) e procurar por mudanc as inesperadas (teste de regressa o). 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 transic a o de estados ou uma especificac a o 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. 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.

5.2. 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

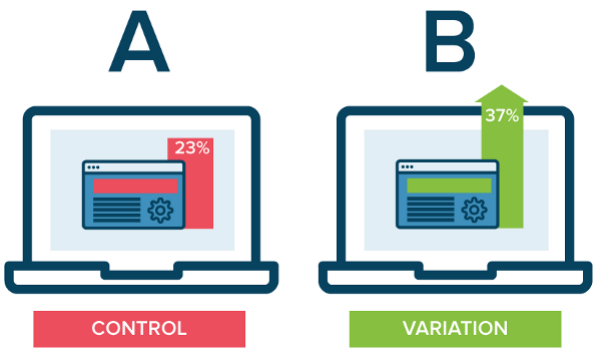

Figura 4. A mountain sunset

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.