CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS V – DIVINÓPOLIS - TÉCNICO EM INFORMÁTICA – 3º B
DISCIPLINA DE PSI

Testes de Software

Arthur Gomes Batista de Souza
Bianca dos Reis Santos
Ítalo Augusto Silva
Jade Moreira

Professor: Michel Silva

Divinópolis, 17 de setembro, 2015.

SUMÁRIO

1. INTRODUÇÃO

Em consequência do crescimento acelerado da utilização de sistemas baseados em computação estabeleceu-se uma crescente demanda por qualidade e produtividade. Surge então a necessidade de técnicas e ferramentas que seriam responsáveis por testar a qualidade, eficiência e segurança desses sistemas. Neste trabalho abordaremos alguns destes testes de software, juntamente com a justificativa da utilização de um deles no trabalho de conclusão de curso (TCC) e implementação do mesmo.

2. IMPORTÂNCIA DOS TESTES DE SOFTWARE

Os testes de software são recursos que muitas vezes não são levados em consideração por desenvolvedores, que acreditam se tratar de um investimento caro, ao qual não vale a pena para o projeto desenvolvido. Porém, um investimento caro voltado para o planejamento e execução de testes assegura o desenvolvedor de um investimento futuro, mais caro, na manutenção de erros do sistema, pois são os testes de software que são responsáveis por averiguar se todo o sistema está funcionando em devido acordo como o planejado. Assim como em um sistema podem existir vários erros e falhas diferentes, existem testes de software diferentes que são responsáveis por analisar uma determinada dimensão do sistema, por exemplo, existem teste que buscam apenas falhas no código, na segurança e na performance do sistema como um todo.

3. TESTE DE UNIDADE

Testes de unidade, também conhecido como testes unitários, são aqueles que testam uma única unidade do sistema. Eles são responsáveis por testar de maneira isolada, geralmente simulando as prováveis dependências que aquela unidade tem. Em sistemas orientados a objetos, é comum que a unidade seja uma classe. Ou seja, quando queremos escrever testes de unidade para uma classe *Pessoa* por exemplo, essa bateria de testes testará o funcionamento da classe *Pessoa*, isolada, sem interações com outras classes. Para a execução deste tipo de teste utiliza-se *frameworks* de testes, como por exemplo o *JUnit*.

O *JUnit* e o *PHPUnit* facilitam a criação de código para a automação de testes unitários com apresentação dos resultados. Com ele, pode ser verificado se cada método de uma classe funciona da forma esperada, exibindo possíveis erros ou falhas e podendo ser utilizado tanto para a execução de baterias de testes como para extensão.

4. TESTE DE INTEGRAÇÃO

Teste de integração é a fase do teste de software em que módulos são combinados e testados em grupo. Ela sucede o teste de unidade, em que os módulos são testados individualmente, e antecede o teste de sistema, em que o sistema completo (integrado) é testado num ambiente que simula o ambiente de produção.

O teste de integração é alimentado pelos módulos previamente testados individualmente pelo teste de unidade, agrupando-os assim em componentes, como estipulado no plano de teste, e resulta num sistema integrado e preparado para o teste de sistema. O propósito do teste de integração é verificar os requisitos funcionais, de desempenho e de confiabilidade na modelagem do sistema. Com ele é possível descobrir erros de interface entre os componentes do sistema. Ou seja, nos testes unitários, cada módulo do sistema será testado individualmente. Sem influência de outras classes do sistema e se possuir alguma dependência, são criados *Mocks* que simulem o funcionamento dessa dependência.

Nos testes de integração, nós queremos que os módulos sejam testados de forma que interajam com o restante do sistema, oferecendo dependências reais para garantir que elas funcionem em conjunto. Imagine a situação de um cadastro. Com o Teste de Unidade, você cria um teste para as regras de negócios, outro para as classes de persistência com o banco e assim por diante. Nos Testes de Integração, você testa somente o método Inserir, ou Pesquisar ou Atualizar e deixa que os módulos si relacionem de verdade, sem *Mock Objects*, sem nenhum artifício qualquer.

Abordagem

A integração pode seguir abordagem incremental ou não. Enquanto na abordagem não-incremental o sistema é agrupado por completo, na abordagem incremental o sistema é agrupado em etapas, facilitando assim o isolamento do erro. Abordagens incrementais diferem entre si na forma em que se constrói as etapas de agrupamento de módulos. Por exemplo, na abordagem descendente os módulos de alto nível são testados e integrados primeiro, permitindo encontrar primeiro os erros de lógica e fluxo de dado de alto nível. Por outro lado, a abordagem ascendente requer o teste e integração dos módulos de baixo nível primeiro.

5. TESTE CAIXA-BRANCA

Essa técnica trabalha diretamente sobre o código fonte do componente de software para avaliar vários aspectos, como: teste de condição, teste de fluxo de dados, teste de ciclos e teste de caminhos lógicos (PRESSMAN, 2005). A forma que a técnica de teste de caixa branca é aplicada torna se totalmente diferente da abordagem da técnica de caixa preta. A técnica de teste de caixa branca é conhecida por vários nomes tais como teste estrutural e teste de caixa de vidro. O engenheiro de sistema realiza o teste direto no código fonte do *software*. São determinados os dados de entrada para analisar a lógica do *software* (MYERS, 2004).

A desvantagem da técnica de caixa de caixa branca é que não analisa se a especificação está certa, concentra apenas no código fonte e não verifica a lógica da especificação (LEWIS e VEERAPILLAI, 2005).

O teste da caixa branca usa a estrutura de controle do projeto procedimental para derivar casos de teste. O engenheiro de software pode derivar os casos de teste que:

- Garantam que todos os caminhos independentes dentro de um módulo tenham sido exercitados pelo menos uma vez;
- Exercitem todas as decisões lógicas para valores falsos ou verdadeiros;
- Executem todos os laços em suas fronteiras e dentro de seus limites operacionais;
- E exercitem as estruturas de dados internas para garantir a sua validade.

Para usar a técnica de teste de caixa branca o código fonte deve estar pronto, neste código fonte terminada, extrai se o grafo de fluxo que representa a lógica do código fonte (GAO, TSAO e WU, 2003). O grafo de fluxo é um gráfico que demonstra a lógica do código fonte através de fios e ramos (MCCABE, 2010).

De acordo Pressman (2006) na construção do grafo de fluxo existem representações simbólicas correspondentes do grafo de fluxo. Usando o grafo de fluxo consegue-se visualizar os controles lógicos do *software*. Para cada círculo (ramos) demonstra uma ou várias linhas do código fonte e para cada seta (arestas) mostra o caminho ou caminhos que o código fonte pode fazer. Quando a existência de condições composta torna se mais difícil à construção do grafo de fluxo, encontra se quando ocorrem operações booleanas (ou, e, não-e, não-ou lógicos). Todos os grafos de fluxo são construídos através uma base principal de grafos tais como: sequência, *if*, *while*, *do while* (BURNSTEIN, 2003).

Um exemplo bem prático desta técnica de teste é o uso da ferramenta livre *JUnit* para desenvolvimento de casos de teste para avaliar classes ou métodos desenvolvidos na linguagem Java. A técnica de teste de estrutural é recomendada para os níveis de Teste da Unidade e Teste da Integração, cuja responsabilidade principal fica a cargo dos desenvolvedores do software, que são profissionais que conhecem bem o código-fonte desenvolvido e dessa forma conseguem planejar os casos de teste com maior facilidade. Dessa forma, podemos auxiliar na redução dos problemas existentes nas pequenas funções ou unidades que compõem um software.

6. TESTE CAIXA-PRETA

A técnica de teste caixa-preta, não apresenta nenhum conhecimento de como foi desenvolvido o software, sabe-se apenas qual foi a entrada do sistema e sua saída, dessa forma poderá saber se o sistema está correto ou não. Dessa forma, pode se afirmar que esse teste tem o intuito de encontrar erros em funções incorretas ou ausentes, erros na interface, erros nas estruturas de dados ou no acesso a banco de dados externos, erros de desempenho e erros de inicialização e término.

Esse teste é utilizado ao final do sistema em busca de saber o que é necessário lapidar no projeto para estar pronto para ser utilizado. Além disso, a caixa-preta tenta responder

algumas questões referentes ao sistema, como: o sistema é particularmente sensível a certos valores de entrada? Quais índices de dados e volumes de dados o sistema pode tolerar? Após esses testes será possível saber até onde o software consegue chegar e o que é necessário alterar para o trabalho ficar excepcional para utilização.

Um exemplo de método para realizar esse teste é:

- **Particionamento de Equivalência:** que divide o domínio de entrada em uma série de diferentes classes, para descobrir uma classe que possua erros. Um exemplo desses testes são: número positivos ou strings em branco.

7. TESTE DE INTEGRIDADE

Testes destinados a avaliar a robustez do objetivo do teste (resistência a falhas) e a compatibilidade técnica em relação a linguagem, sintaxe e utilização de recursos. Esse teste é implementado e executado em vários objetivos do teste, como unidades e unidades integradas. Esta técnica de teste tem o objetivo também de encontrar erros/falhas provenientes da integração entre componentes de um sistema. Normalmente estes erros são de transmissão e validação de dados.

8. TESTE DE SEGURANÇA

O Teste de Segurança tem como meta garantir que o funcionamento da aplicação esteja exatamente como especificado. Verifica também se o software se comporta adequadamente mediante as mais diversas tentativas ilegais de acesso, visando possíveis vulnerabilidades. Para isso, testa se todos os mecanismos de proteção embutidos na aplicação de fato a protegerão de acessos indevidos.

É muito comum que as aplicações se tornem alvo de sujeitos que buscam provocar ações que possam prejudicar ou, até mesmo, beneficiar pessoas. Em função de situações como estas, o Teste de Segurança propõe demonstrar se a aplicação faz exatamente o que deve fazer ou se a aplicação não faz o que não deve ser feito.

A execução do Teste de Segurança possibilita que dúvidas sobre prováveis vulnerabilidades do software sejam sanadas. Pode auxiliar também na definição de um

plano de contingência, visando determinar qual precaução será tomada contra os possíveis ataques.

A validação de segurança pode ser realizada em duas fases, a fase estática e a fase dinâmica:

- A primeira tenta localizar falhas inseridas durante o desenvolvimento do projeto, como um estado não-alcançável ou possíveis erros humanos introduzidos no código. Nesse caso são utilizados métodos de análise estática (ex.: inspeção de código, analisadores de vulnerabilidade estáticos), ou prova de teorema, os quais não necessitam executar o sistema.
- A segunda se foca na verificação da implementação durante sua execução, i.e, verificar o sistema exercitando seu código, onde entradas reais são fornecidas para verificar os mecanismos de segurança.

9. TESTE PARA O TCC

O teste de software escolhido para o trabalho de conclusão de curso (TCC) foi o teste de unidade, ou teste unitário, por diversos motivos, como os objetivos de serviços que se encaixam com o projeto e o tipo de teste que o grupo planejou em realizar e facilidade ao acesso de tutoriais e informações na internet sobre a implementação do teste. E como o projeto se trata de uma rede social voltada para à área da saúde o teste realizado em cada unidade do código exerce muita influência na análise dos erros que se relacionam diretamente com o código fonte do projeto.

Outro fator que foi responsável por influenciar o grupo na escolha do teste foi a forma como este é executado no código do projeto, através do *PHPUnit*, um *framework* semelhante ao *JUnit* porém voltado para a linguagem PHP, bastante utilizado e recomendado pela comunidade computacional online de alguns fóruns da internet justamente pela sua eficiência e facilidade na execução dos testes de software.

10. CONCLUSÃO

Portanto, conclui-se que os teste de software são recursos importantíssimos para manter a qualidade, integridade e segurança do software. Mesmo se tornando um custo inicial um pouco alto, são recursos preventivos, ao qual isenta o desenvolvedor de futuros custos que envolva a manutenção e correção de erros do sistema. Sendo de responsabilidade do desenvolvedor escolher os tipos de teste que mais se adequam aos objetivos e funcionalidades que o projeto que está sendo desenvolvido possui.

REFERÊNCIAS

UFCG. Teste de unidade. Disponível em: < http://www.dsc.ufcg.edu.br/~jacques/cursos/apoo/html/impl/impl3.htm >. Acessado em 13 de setembro, 2015.

DEVMEDIA. Artigo engenharia de software – Introdução a teste de software. Disponível em: < http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035 >. Acessado em 13 de setembro, 2015.

CAELUM. Unidade, Integração ou Sistema? Qual teste fazer?. Disponível em: < http://blog.caelum.com.br/unidade-integracao-ou-sistema-qual-teste-fazer/ >. Acessado em 13 de setembro, 2015.

INFOBLOGS. Criando testes com JUnit. Disponível em: < http://javafree.uol.com.br/dependencias/tutoriais/testes_junit.pdf >. Acessado em 13 de setembro, 2015.

DEVMEDIA. A importância dos testes de software para a qualidade do software. Disponível em: < http://www.devmedia.com.br/a-importancia-dos-testes-para-a-qualidade-do-software/28439 >. Acessado em 13 de setembro, 2015.

YURI ADAMS. Teste de integração validando ainda mais sua aplicação. Disponível em: < https://yuriadamsmaia.wordpress.com/2011/08/09/testes-de-integracao-validando-ainda-mais-sua-aplicacao-pa/ >. Acessado em 13 de setembro, 2015.

QUALIDADE DE SOFTWARE. Teste de integração. Disponível em: < http://qualidade-de-software.blogspot.com.br/2010/01/teste-de-integracao.html >. Acessado em 13 de setembro, 2015.

UFRJ. Técnicas de Testes de Software. Disponível em: < http://www.dcc.ufrj.br/~schneide/es/2000/1/a1/al13_20.htm >. Acessado em 13 de setembro, 2015.

UFPR. Técnicas de Testes de Software. Disponível em: < http://www.inf.ufpr.br/lmperes/ci221/aula_tecnicas_teste_Luis_Renato.pdf >. Acessado em 13 de setembro, 2015