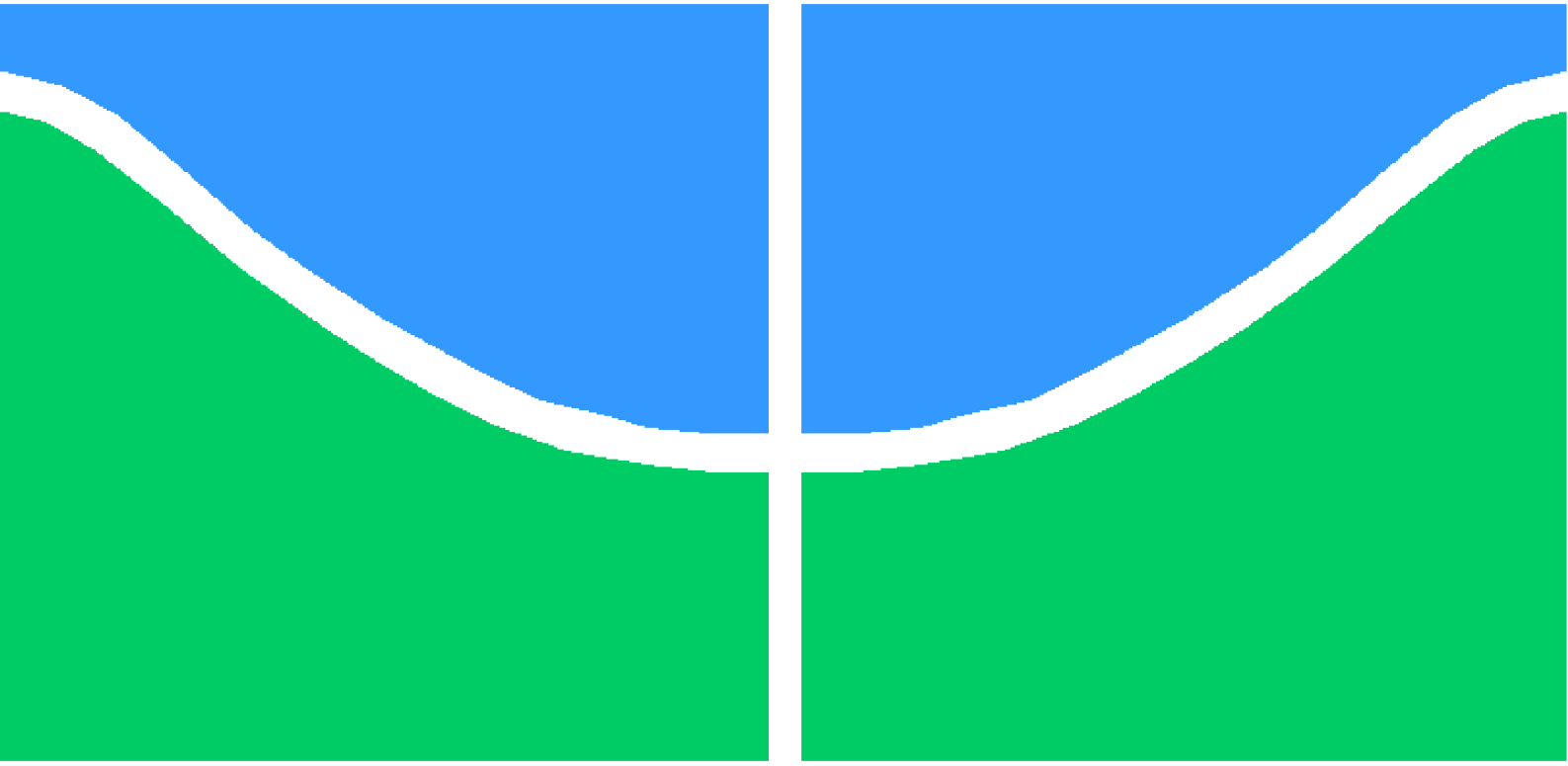

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Técnicas de testes automatizados em processos de desenvolvimento de software empírico: um estudo de caso do projeto Noosfero

Autor: Rodrigo Medeiros Soares da Silva

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2014

Rodrigo Medeiros Soares da Silva

Técnicas de testes automatizados em processos de desenvolvimento de software empírico: um estudo de caso do projeto Noosfero

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2014

Rodrigo Medeiros Soares da Silva

Técnicas de testes automatizados em processos de desenvolvimento de software empírico: um estudo de caso do projeto Noosfero / Rodrigo Medeiros Soares da Silva. – Brasília, DF, 2014-

27 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2014.

1. Testes. 2. Software livre. I. Prof. Dr. Paulo Roberto Miranda Meirelles. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Técnicas de testes automatizados em processos de desenvolvimento de software empírico: um estudo de caso do projeto Noosfero

CDU 02:141:005.6

Rodrigo Medeiros Soares da Silva

Técnicas de testes automatizados em processos de desenvolvimento de software empírico: um estudo de caso do projeto Noosfero

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, Dezembro de 2014:

Prof. Dr. Paulo Roberto Miranda Meirelles
Orientador

Prof..
Convidado 1

Prof.
Convidado 2

Brasília, DF
2014

Resumo

Abstract

Lista de ilustrações

Lista de abreviaturas e siglas

BDD	Behavior Driven Development
CDTC	Centro de Difusão de Tecnologia e Conhecimento
CMS	Content Management System
CPD	Centro de Informática
DEG	Decanato de Ensino de Graduação
DPP	Decanato de Pesquisa e Pós-graduação
FGA	Faculdade UnB Gama
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
ICC	Instituto Central de Ciências
IRC	Internet Relay Chat
JSON	JavaScript Object Notation
LAPPIS	Laboratório de Produção, Pesquisa e Inovação em Software
LDAP	Lightweight Directory Access Protocol
MES	Manutenção e Evolução de Software
MVC	Model-View-Controller
ProIC	Projeto de Iniciação Científica
PuSH	PubSubHubbub
Rails	Ruby on Rails
SMT	Tecnologias de Mídia Social
SSL	Secure Socket Layer
TCC	Trabalho de Conclusão de Curso

TLS	Transport Layer Security
UnB	Universidade de Brasília
USP	Universidade de São Paulo
W3C	World Wide Web Consortium

Sumário

1 Introdução

1.1 Objetivos

1.1.1 Objetivos Gerais

1.1.2 Específicos

1.2 Organização do Trabalho

2 2. Testes

Testar é uma pratica intrínseca ao desenvolvimento e é antiga a necessidade de cirar programas para testar cenários específicos (EVERETT et al., 2007). A automação de testes é uma prática ágil, eficaz e de baixo custo para melhorar a qualidade dos sistemas de software.

No entanto utilizar testes automatizados como uma premissa básica do desenvolvimento é um fenômeno relativamente recente,] com início em meados da década de 1990 (POTEL; COTTER, 1995). Além do fato de ser uma técnica bastante utilizada pelas metodologias ágeis de desenvolvimento.

2.1 Testes Automatizados

Testes automatizados é a prática de tornar os testes de software independentes da intervenção humana, criando scripts ou programas simples de computador que exercitam o sistema em teste, capturam os efeitos colaterais e fazem verificações, tudo automaticamente e dinamicamente (MESZAROS; WESLEY, 2007).

Os testes automatizados afetam diretamente a qualidade dos sistemas de software, portanto agregam valor ao produto final, mesmo que os artefatos adicionais produzidos não sejam visíveis para os usuários finais do sistemas. Estes testes podem ser divididos em diversos tipo, o que facilita a manutenção dos mesmos, coleta de métricas.

1. **Testes de unidade:** teste de correção responsável por testar os menores trechos de código de um sistema que possui um comportamento definido e nomeado. Normalmente, ele é associado a funções para linguagens procedimentais e métodos em linguagens orientadas a objetos.

2. **Testes funcionais:**

3. **Testes de integração:**denominação ampla que representa a busca de erros de relacionamento entre quaisquer módulo de um software, incluindo desde a integração de pequenas unidades até a integração de bibliotecas das quais um sistema depende, servidores e gerenciadores de banco de dados.

4. **Testes de interface de usuário** testes que verificam a correção por meio da simulação de eventos de usuário, a partir destes eventos, são feitas verificações na interface e em outras camadas.

5. **Testes de leiaute:** testes que buscam avaliar a beleza da interface e verificar a presença de erros após a renderização, dificeis de indentificar com testes comuns de interface

6. **Testes de aceitação:** são testes de correção e validação, idealmente especificados por clientes ou usuários finais do sistema para verificar se um modulo funciona como foi especificado (MARTIN, 2005). Testes de aceitação devem utilizar linguagem proxima da natural para evitar problemas de interpretação e de ambiguidades (MUGRIDGE; DCUNNINGHAM, 2005).

7. **Testes de desempenho:** testes que executam trechos do sistema e armazenam os tempos de duração obtidos, que ajudam a identificar gargalos que precisam de otimização para diminuir o tempo de resposta para o usuario (LIU, 2009).

8. **Testes de carga:** teste que exercita o sistema sobre condições de uso intenso para avaliar se a infraestrutura é adequada para a expectativa de uso do sistema. (AVRITZE; WEYUKER, 1994)

9. **Testes de estresse:** teste que visa descobrir os limites do uso da infraestrutura, isto é , qual a quantidade máxima de usuários e requisições que o sistema consegue antender corretamente e em um tempo aceitável.

10. **Testes de longevidade:** teste que tem por objetivo encontrar erros somente visiveis com um

11. longo tempo de execuçao do sistema, erros que podem ser de cache, replicação, execução de serviços agendados, vazamento de memória.

12. **Testes de segurança:** os testes de segurança servem para verificar se os dados ou funcionalidades confidenciais de um sistema estão protegidos de fraude ou de usuários não autorizados. A segurança de um sofware pode envolver aspectos de confidenciabilidade, integridade, autenticação, autorização, privacidade (WHITTAKER, 2006).

2.2 Técnicas de Desenvolvimento de testes automatizados

Automação de testes é uma técnica voltada principalmente para a melhoria de qualidade dos sistemas de software. No processo de desenvolvimento de software é fundamental controlar o custo do processo de testes, para isso baterias de testes automatizados devem ser bem definidas e implementadas. Assim é importante conhecer boas práticas e técnicas de de desenvolvimento de testes automatizados. Existem várias técnicas de desenvolvimento de software com testes que influenciam diretamente na qualidade do sistema.

Estas técnicas geralmente possuem um processo de atividades pequeno e simples, como TDD e BDD.

2.2.1 TDD - Test Driven Development

Desenvolvimento dirigido por testes, também conhecido como TDD (Test-Driven Develepment) é uma técnica de desenvolvimento de software que se dá pela repetição disciplinada de um ciclo curto de passos de implementação de testes e do sistema (KOSKELA, 2007). O ciclo de TDD é definido pelos seguintes passos:

1. Implementar um caso de teste;
2. Implementar um trecho do código suficiente para o novo caso de teste ter sucesso de tal modo que não quebre os testes previamente escritos;
3. Se necessário, refatorar o código produzido para que ele fique mais organizado;

2.2.1.1 Benefícios do TDD

Uma boa prática do TDD é a bateria de testes, que ajuda o desenvolvedor a evitar erros de regreção, quando o desenvolvimento de uma nova funcionalidade quebra uma já existente. TDD também tende a contribuir com uma alta cobertura de código, uma fez que o desenvolvedor precisa escrever o testes antes da funcionalidade, possibilitando a criação de um código mais preciso, coeso e menos acoplado. Para Massol em JUnit in Action, "o objetivo de TDD é 'código claro que funciona' (MASSOL; HUSTED, 2003). TDD propõe o desenvolvimento sempre em pequenos passo, deve-se escrever testes sempre para uma menor funcionalidade possível, escrever o código mais simples que faça o teste passar e fazer sempre apenas uma refatoração por vez (BECK, 2002). Assim o desenvolvedor se detém a criar soluções simples, sempre acompanhado de um constante feedback dos testes. O ciclo curto de passos definidos por TDD cria uma dependência forte entre codificação e testes, o que favorece e facilita a criação de sistemas com alta testabilidade (BERNARDO, 2011). Índices altos de cobertura de código e testabilidade não garantem necessariamente qualidade do sistema, mas são métricas bem vistas para sistemas bem desenvolvidos.

2.2.2 BDD - Behavior Driven Development

3 Métodos de Desenvolvimento Empírico

3.1 Software Livre

3.2 Métodos ágeis

4 Estudo de Caso: Noosfero

4.1 Desenvolvimento no processo de colaboração ao Noosfero

4.2 Testes no processo de colaboração ao Noosfero

4.3 Funcionalidades desenvolvidas

5 Considerações finais

5.1 Resultados

5.2 Propostas futuras

Referências

AVRITZE, A.; WEYUKER, E. J. Generating test suites for software load testing in international symposium on software testing and analysis (issta). 1994. Disponível em: <http://dl.acm.org/citation.cfm?id=186258.186507>. Citado na página 18.

BECK, K. *Test-Driven Development by Example*. [S.l.]: Addison-Wesley Prefessional, 2002. Citado na página 19.

BERNARDO, P. C. *Padrões de testes automatizados*. Dissertação (Mestrado) — Instituto de Matemática e Estatística – Universidade de São Paulo, 2011. Disponível em: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-02042012-120707/>. Citado na página 19.

EVERETT, G. D. et al. *Software Testing*. [S.l.: s.n.], 2007. Citado na página 17.

KOSKELA, L. *Test Driven: Pratical TDD and Acceptance TDD for Java Developers*. [S.l.]: Manning Publications, 2007. Citado na página 19.

LIU, H. H. *Software Performance and Scalability: A Quantitative Approach (Quantitative Software Engineering Series)*. [S.l.: s.n.], 2009. Citado na página 18.

MARTIN, R. C. The test bus imperative: Architectures that support automated acceptance testing. 2005. Disponível em: <http://www.martinfowler.com/ieeeSoftware/testBus.pdf>. Citado na página 18.

MASSOL, V.; HUSTED, T. *JUnit in Action*. [S.l.]: Manning Publications, 2003. Citado na página 19.

MESZAROS, G.; WESLEY, A. *XUnit Test Patterns: Refactoring Test Code*. [S.l.: s.n.], 2007. Citado na página 17.

MUGRIDGE, R.; DCUNNINGHAM, W. *Fit for Developing Software: Framework for Integrated Tests*. [S.l.: s.n.], 2005. Citado na página 18.

POTEL, M.; COTTER, S. *Inside Taligent Technology*. [S.l.]: Taligent Press, 1995. Citado na página 17.

WHITTAKER, M. A. J. A. *How to break Web software: functional and security testing of Web applications and Web services*. [S.l.: s.n.], 2006. Citado na página 18.