Karine Valença
Murilo Duarte
Tiago Assunção
Wilton Rodrigues

Trabalho Final - Verificação e Validação de Software

Atividade submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção da aprovação em Verificação e Validacção de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Ricardo Ajax

Brasília, DF

2016

Lista de abreviaturas e siglas

FS	Fábrica de *Software*
GQM	*Goal Question Metric*
UnB	Universidade de Brasília
EVeV	Equipe de Verificação e Validação
VeV	Verificação e Validação

Sumário

1 Introdução

O TEM-DF (Tranparência de Escalas Médicas do DF) é um software que visa apresentar as escalas dos médicos da rede pública de saúde do Distrito Federal, facilitando encontrar algum médico da especialidade adequada e na região desejada, além de permitir aos usuários realizarem reclamações ou elogios aos médicos.

O projeto foi desenvolvido por alunos da Faculdade do Gama - UnB, nas disciplinas de GPP e MDS e uma das restrições de qualidade do projeto é cobertura de testes acima de 90%. Porém, por ter sido desenvolvido por programadores iniciantes, boa parte dos testes, mesmo os que passaram, não eram de boa qualidade, ou seja os casos não estão contemplando o particionamento de equivalência e os valores limites. Além de conter apenas testes no nível unitário e funcional. Outro problema, é que durante o desenvolvimento, muito código era enviado ao repositório oficial sem testes e esse código não foi barrado ao ser enviado para as branchs principais do projeto.

Esse software pode ser acessado no seguinte endereço[1]

E sua documentação pode ser acessada neste endereço[2]

1.1 Problema

- Falta de Verificação dos Testes unitários no momento da integração da build.
- Baixa confiabilidade na qualidade dos testes unitários realizados.

1.2 Objetivos Gerais

- Verificar a integridade, analisando se todos os testes de unidade e aceitação passaram, de uma build de software a ser adicionada
- Melhorar a confiabilidade dos testes unitários realizados no software.

1.3 Objetivos Específicos

- Aplicar uma ferramenta de Integração contínua

[1] https://github.com/EscalaSaudeDF/TEM-DF

[2] http://lappis.unb.br/redmine/projects/grupo-1-tem-transparencia-das-escalas-dos-medicos-do-df/wiki

- Analisar os testes unitários no momento da integração

1.4 Questões de Pesquisa

- Como pode ser validado um novo incremento de software?
- Como garantir melhor qualidade dos testes unitários realizados?

1.5 Estratégia de Pesquisa

1.5.1 Revisão Sistemática de Literatura

- Verificação e validação de software
- Integração contínua de software
- Utilização da ferramenta travis para integração contínua de software
- Padrões de qualidade em testes unitários de software

1.5.2 Metodologia

- Definição de palavras-chave
- Gerar de uma string de busca
- Aplicação de string na base scopus
- Filtro de susca(análise do título, análise do abstract, referência)
- Snowball para trás

1.6 Resultados Esperados

1.6.1 Abordagem

- Qualitativas, tendo a confiabilidade de que os incrementos de software estão sendo testados corretamente.
- Quantitativa, o projeto deverá ter pelo menos 90% de cobertura de código de testes.

1.6.2 Métodos

- Aplicação de uma política de avaliação de qualidade estática de código contínua para validação de cada submissão de incremento de software.

1.6.3 Técnicas

- Integração Contínua

1.6.4 Ferramentas

Ferramentas a serem utilizadas e formas de estudos empíricos (Estudos de caso, questionários, entrevistas, estudos estatísticos, dentre vários outros possíveis).

- Travis[3]
- Estudo de Caso

[3] https://docs.travis-ci.com

2 Referencial Teórico

2.1 Verificação e validação

A verificação e validação são chave no desenvolvimento de software. Elas garantem que o produto cumpra o que esta destinado a cumprir, preservando sobretudo aspectos de qualidade e respectivamente sua melhoria contínua. Desta forma, garantem a detecção prematura de falhas no processo de desenvolvimento, bem como reduzem o tempo necessário para remover essas falhas e ajudam a produzir um produto mais robusto.

(ENGELS et al., 2003), define que na Engenharia de Software podemos distinguir dois meios de garantir a qualidade, o construtivo e o analítico. Sendo o construtivo um meio de prevenção de ocorrência de erros no processo de desenvolvimento, através da utilização de linguagens menos propensas a erros e/ou linguagens mais automatizadas. Já o meio analítico é usados para detectar erros em implementações e desenvolvimentos, utilizando modelos de análise que permitam raciocinar sobre propriedades importantes do software, dividindo-se em técnicas de verificação e de validação.

(BELATEGI et al., 2012), em seu estudo, afirma que a verificação refere-se ao processo de examinar cada fase do ciclo de vida do desenvolvimento do software, assegurando que a saída de um produto satisfaça todos os requisitos pertinentes. Com uso de ferramentas é possivel realizar a verificação da aderência do código, documentações e padrões definidos. Por outro lado a validação é a principal causa de testes em sistemas, onde são geradas baterias de testes, que por sua vez testam tanto aspectos funcionais como técnicos e assim são comparados com os requisitos previstos. Para validar as propriedades do que esta sendo desenvolvido a principal abordagem é testar a implementação do sistema. Assim, um caso de teste deve ser gerado a partir do modelo de entrada que a aplicação fornece. A aplicação de um caso de teste para uma implementação de um modelo permite então validar as propriedades da implementação e também para validar o modelo, já que os testes são construídos preservando a semântica entre os testes e os modelos, de acordo com o estudo de, (ENGELS et al., 2003).

Com isso, é possível alcançar uma visão real do planejamento da construção antes mesmo de construir o produto final. Para isso é necessário trazer a responsabilidade da execução das atividades de verificação e validação durantes todos os ciclos do projeto.

2.2 Qualidade de Testes

Quando se trata de manufatura ou produção de componentes físicos, os testes sobre a confiabilidade do que foi produzido é feito a partir de uma porcentagem do lote retirado do todo. Caso a quantidade retirada do lote passe nos testes, este é aprovado, caso não, todas peças são rejeitadas.

Ao se tratar de software, a maneira de executar os testes é bastante diferente. Cada software tem a sua peculiaridade e é desenvolvido de acordo com as capacidades dos desenvolvedores participantes do projeto. (MATHUR, 1991), em seu estudo, afirma que antigamente eram realizados os testes com base nos testes de hardware. E, que esses, por sua vez, não eram realísticos.

Dessa maneira, há a necessidade de que este produto tenha seus testes específicos, partindo desde o menor nível possível, passando pela sua integração e chegando no momento do uso do produto pelo usuário. Os softwares são desenvolvidos e assim, uma bateria de testes é definida no documento de casos de teste. Vários testes são executados de acordo com o sistema, podendo ser entrem eles testes de:

- Testes Unitários
- Testes de Integração
- Testes de Sistema
- Testes de Aceitação
- Testes de Instalação

Mas algumas perguntas que sempre intrigou os programadores responsáveis pelos testes são:

- O quanto eu devo testar de maneira que o software esteja suficientemente confiável?
- Quais são os pontos necessários a ponderar para entender o conceito de confiabilidade dos meus testes, por consequência, do meu software?

(FILHO, 2002) produziu um estudo que informa várias formas de se verificar a atestar a qualidade dos testes de software executados. E todas as suas maneiras de identicar estes pontos é partindo de dois princípios:

- Avaliação do plano de testes, verificando se todos os requisitos foram descritos no caso de testes e se todos os testes e seus múltiplos casos foram contemplados na implementação.

- Associação dos testes cobrindo tudo o que foi implementado como requisito do sistema.

Assim, ele estabeleceu duas métricas que possibilitam aferir a confiabilidade dos testes de um software. Sendo elas:

1. Número de casos de teste implementados / número de casos de teste estimados.
2. Cobertura do código

Dessa maneira, a organização consegue aferir números sobre a confiabilidade da implementação dos testes a partida da aplicação dos resultados obtidos (FILHO, 2002)

2.3 Integração Contínua

Uma prática utilizada para eliminar descontinuidades entre o desenvolvimento e a implantação é a integração contínua (FITZGERALD; STOL, 2015). A integração contínua é bastante utilizada atualmente devido ao avanço das metodologias ágeis, e consiste de uma prática na qual os desenvolvedores de software trabalham em pequenos incrementos, assim, eles contribuem com o código frequentemente e garantem que o projeto compile e passe a suíte de testes a qualquer momento (DESHPANDE; RIEHLE, 2008).

O uso da integração contínua apresenta várias vantagens, ela aumenta a frequência de lançamento de software, a previsibilidade, a produtividade do desenvolvedor, entre outros (STåHL; BOSCH, 2014).

Sendo assim, ao utilizar essa prática, incrementos de software serão lançados com uma frequência alta. Diante disso, uma questão importante é sobre como validar tantos incrementos. Para resolver esse problema existem ferramentas que auxiliam a realização da integração contínua. O que é essencial para a automatização da integração contínua, é o uso de um repositório para o código, assim ferramentas como Git ou Subversion são úteis. Além disso, existem ferramentas analisam as novas mudanças de código enviadas, elas checam o código a fim de verificar se aquela mudança de código é boa e não "quebra"nenhuma outra funcionalidade ou teste (MEYER, 2014), nesse caso, ferramentas como Travis, Jenkins ou GitLab Ci auxiliam esse processo.

3 Desenvolvimento

TO DO

4 Resultados Obtidos

TO DO

5 Conclusão e Trabalhos Futuros

TO DO

Referências

BELATEGI, L. et al. Embedded software product lines: domain and application engineering model-based analysis processes. *Journal of Software: Evolution and Process*, Sep 2012. Citado na página 7.

DESHPANDE, A.; RIEHLE, D. Open source development, communities and quality. In: _____. [S.l.]: Springer US, 2008. cap. Continuous Integration in Open Source Software Development. Citado na página 9.

ENGELS, G. et al. Model-based verication and validation of properties. *FUNIGRA'03, Uniform Approaches to Graphical Process Specification Techniques (Satellite Event for ETAPS 2003)*, Paderborn, Germany, Jun 2003. Citado na página 7.

FILHO, F. dos S. *MÉTRICAS E QUALIDADE DE SOFTWARE*. [S.l.], 2002. Disponível em: <http://www.angelfire.com/nt2/softwarequality/QualitySoftware.pdf>. Citado 2 vezes nas páginas 8 e 9.

FITZGERALD, B.; STOL, K.-J. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 2015. Citado na página 9.

MATHUR, A. P. Performance, effectiveness, and reliability issues in software testing. In: *Computer Software and Applications Conference, 1991. COMPSAC '91., Proceedings of the Fifteenth Annual International*. [S.l.: s.n.], 1991. p. 604–605. Citado na página 8.

MEYER, M. Continuous integration and its tools. *IEEE Software*, May 2014. ISSN 0740-7459. Citado na página 9.

STåHL, D.; BOSCH, J. Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, 2014. Citado na página 9.