Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

Usando Transformações de código para melhorar detecção de conflitos de teste através de ferramentas de geração de testes unitários

Proposta de Trabalho de Graduação

Aluno: **João Pedro Ramos Moisakis**
jprm@cin.ufpe.br
Orientador: **Paulo Henrique Monteiro Borba**
phmb@cin.ufpe.br

Recife

Junho de 2021

Sumário

Introdução

Durante o desenvolvimento colaborativo de software, diferentes desenvolvedores podem realizar mudanças a um mesmo conjunto de arquivos dependentes entre si. Neste sentido, sistemas de controle de versão são amplamente utilizados com o objetivo de aumentar a produtividade do time, considerando que contribuições individuais de desenvolvedores podem ser realizadas em paralelo. Por exemplo, a criação de *branches* de desenvolvimento para uma dada contribuição individual, e posteriormente, sua integração com outras contribuições, processo que é conhecido como cenário de *merge*.

Apesar dos benefícios alcançados pelo uso de ferramentas de controle de versão, durante a tentativa de integrar contribuições de diferentes desenvolvedores, conflitos podem ocorrer impactando negativamente a produtividade do time e a qualidade do software desenvolvido [1], [2]. Estes conflitos podem se manifestar em diferentes estágios durante um cenário de *merge*. Conflitos de *merge* ocorre já durante a tentativa falha de integração das contribuições, enquanto conflitos de *build* e teste durante a tentativa falha de gerar um arquivo executável do programa, que inclui a execução dos testes durante o processo de *build*. Ferramentas de *merge* convencionais apresentam limitações no reporte de conflitos de *merge* com o alto número de falsos positivos. Mesmo usando ferramentas mais inteligentes, nem todos os conflitos podem ser detectados e tratados. Assim, a resolução destes conflitos é realizada via intervenção do integrador, que ainda assim pode resolver estes conflitos e ocasionar outros problemas posteriores.

Neste sentido, conflitos de teste são causados por mudanças de comportamento não esperadas durante a integração, quando a *build* e o *merge* não apresentaram nenhuma falha, mas o resultado do *merge* comprometeu determinada funcionalidade, consequentemente, não conseguem ser identificados por ferramentas de teste comuns por seu impacto ser a nível semântico. Conflitos de teste representam um desafio no que diz respeito sua detecção e resolução, visto que é necessário conhecimento mais profundo sobre o funcionamento do *software*. Como resultado, os impactos causados na qualidade do *software* podem aumentar, considerando que estes conflitos, quando identificados muito tardiamente no ciclo de vida do projeto, podem impactar o usuário final caso encontrados no sistema em produção. Assim, custo e tempo adicionais envolvendo o time são necessários para resolver este problema.

Conflitos de teste poderiam ser detectados já durante a fase execução dos testes durante o processo de *build*; por exemplo, ao observar a resultados diferentes dos testes após a integração das contribuições do cenário de *merge*. Porém, esta abordagem nem sempre pode representar uma opção viável devido à fragilidade e baixa cobertura da suíte de testes presente no projeto. Para solucionar essa limitação, ferramentas como Evosuite[3][4] e Randoop [5] geram testes unitários para uma dada classe, e estes testes poderiam ser usados na detecção de conflitos de teste. Da Silva et al. [6] propõem a ferramenta SMAT, que para um cenário de *merge*, utiliza essas ferramentas para gerar testes unitários, executá-los e posteriormente, detectar mudanças que causam conflitos de teste. Entretanto, essas ferramentas apresentam limitações, considerando que nem sempre os testes unitários gerados podem diretamente exercitar os métodos/atributos de uma classe envolvidos nas mudanças do cenário de *merge*. Através do uso de transformações de código, podemos tornar elementos de uma classe diretamente referenciáveis pelos testes gerados sem alterar a semântica do código, permitindo a exploração das mudanças realizadas em cenários de *merge*.

Este trabalho busca criar e realizar essas transformações em cenários de *merge* com o objetivo de avaliar o código transformado através de um estudo empírico, comparando a testabilidade do código antes e depois das transformações utilizando a ferramenta SMAT proposta anteriormente.

Objetivo

Este estudo tem como objetivo avaliar a testabilidade de um programa com transformações de código usado por ferramentas de geração de testes de unidade (EvoSuite e Randoop) para a criação de suítes de testes, que são usadas para a detecção de conflitos de teste.

Cronograma

Atividade	Junho	Julho	Agosto
Definição do escopo			
Implementação das Transformações			
Criação do *Dataset*			
Execução do estudo e Análise de resultados			
Escrita do TG			
Apresentação			

Possíveis avaliadores

Breno Alexandro Ferreira de Miranda

Marcelo d'Amorim

Referências

[1] A. Sarma, D. F. Redmiles, and A. Van Der Hoek, “Palantir: Early detection of development conflicts arising from parallel code changes,” IEEE Transactions on Software Engineering, vol. 38, no. 4, pp. 889–908, 2012.

[2] S. McKee, N. Nelson, A. Sarma, and D. Dig, “Software practitioner perspectives on merge conflicts and resolutions,” in 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2017, pp. 467–478.

[3] M. M. Almasi, H. Hemmati, G. Fraser, and A. Arcuri, “An industrial evaluation of unit test generation: Finding real faults in a financial application,” in 39th International Conference on Software Engineering, ICSE 2017, Software Engineering in Practice Track. IEEE, 2017, pp. 263–272.

[4] G. Fraser, “A tutorial on using and extending the evosuite search-based test generator,” in Search-Based Software Engineering. Springer, 2018, pp. 106–130. [

[5] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball, “Feedback-directed random test generation,” in 29th International Conference on Software Engineering (ICSE’07), 2007, pp. 75–84.

[6] L. Da Silva, P. Borba, W. Mahmood, T. Berger, and J. Moisakis, “Detecting Semantic Conflicts Via Automated Behavior Change Detection” in 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). (a ser publicado)

Assinaturas

Aluno: **João Pedro Ramos Moisakis**

Orientador: **Paulo Henrique Monteiro Borba**