

Uma Abordagem Visual para Evolução de *Test Smells* em Sistemas de Software Java

Adriana P. S. Cruz¹ (mestrando), Heitor Costa (orientador)¹

¹Mestrado Acadêmico em Ciência da Computação
Programa de Pós-Graduação em Ciência da Computação (PPGCC)
Departamento de Ciência da Computação – Universidade Federal de Lavras (UFLA)
Câmpus Universitário - Aqueça Sol – CEP 37200-000 Lavras – MG – Brazil
Ingresso: 08/2019 – Qualificação: 08/2020 – Previsão Defesa: 07/2021

adriana.cruz@estudante.ufla.br, heitor@ufla.br

Resumo. *O teste de software é parte do desenvolvimento de software. Durante o teste, podem surgir más escolhas de implementação do código de teste, levando a inserção de Test Smells. A presença de Test Smells em código de teste pode o tornar ineficaz em encontrar bugs, gerando retrabalho e custos adicionais. A manutenção do código de teste ocorre por causa da evolução do código de produção. Para visualizar a evolução de sistemas de software, têm sido utilizadas técnicas de visualização de software. Apesar dessas técnicas contribuírem para visualizar a evolução de sistemas de software, elas não abrangem código de teste para visualização de Test Smells. Dessa forma, no trabalho de mestrado, o objetivo é proporcionar melhor entendimento das ocorrências de Test Smells e sua evolução em código de teste aos Gerentes de Testes/Testadores, para agilizar a identificação de problemas nos casos de testes, melhorando sua qualidade. Para isso, será elaborada uma abordagem para visualizar essa evolução.*

Palavras-Chave. *Evolução de Software, Visualização de Software, Test Smells.*

Evento CBSOft: SBES

1. Caracterização do Problema

Teste de software é o processo de executar o software de maneira controlada para verificar se o software se comporta conforme o especificado, sendo fundamental para a sua avaliação durante seu desenvolvimento [Crespo *et al.*, 2004]. O teste não garante um software livre de erros, porém é necessário ter testes durante o projeto de sistemas de software, para minimizar os erros e diminuir custos [Silva *et al.*, 2016]. Por causa da quantidade de código de teste que as empresas lidam, há a preocupação com a qualidade desses códigos, surgindo a Engenharia de Código de Teste de Software (ECTS) (*Software Test-Code Engineering*), que refere-se aos métodos, às técnicas e às ferramentas utilizadas para verificar e manter código de teste [Yusifoglu *et al.*, 2015]. Na prática, nem sempre a ECTS é seguida, resultando na inserção de anti-padrões em códigos de testes (*Test Smells*) [Palomba *et al.*, 2018]. *Test Smells (TS)* são más escolhas de *design*/implementação do código de testes e sua presença pode afetar negativamente a compreensão e a manutenção desse código. Eles podem ocasionar testes menos eficazes em encontrar *bugs* no código de produção [Spadini *et al.*, 2018]. Diante disso, algumas ferramentas foram propostas para automatizar a detecção de *TS*. Dentre essas ferramentas, destacam-se JNose Test [Vériginio *et al.*, 2019], que detecta 21 tipos de *TS* e a quantidade de vezes em que eles ocorrem e a ferramenta VITrUM [Pecorelli *et al.*, 2020], que analisa classes de teste e utiliza cores diferentes conforme a presença de *TS*, sendo possível filtrar os resultados por período e por tipo de *TS*.

Assim como o código de produção passa por evoluções, o código de teste evolui para atender novos casos de teste. Não se sabe ao certo se a evolução do código de teste é responsável por introduzir *TS* [Greiler *et al.*, 2013], mas, mesmo que eles sejam removidos, outros podem ser inseridos [Kim, 2019]. Técnicas de visualização de software têm sido utilizadas para mostrar a evolução de sistemas de software para facilitar a sua compreensão [Bastos; Costa, 2016]. Apesar disso, elas não abrangem a evolução de códigos de teste para visualização de *TS*. Desse modo, não foram encontradas na literatura ferramentas que utilizam técnicas de visualização de software para mostrar a presença de *TS* em códigos de teste durante sua evolução [Garousi *et al.*, 2018]. Dessa forma, nesse trabalho de mestrado, o objetivo é proporcionar melhor entendimento da evolução de *TS* em código de teste aos Gerentes de Testes/Testadores. Para isso, será elaborada uma abordagem para visualizar essa evolução para facilitar o processo de identificação de problemas no código de teste e melhorar sua qualidade. Esse trabalho está no escopo de um *framework* conceitual em desenvolvimento por um grupo de pesquisadores interinstitucional, que consiste na integração de componentes para prevenção, identificação e correção de *TS* para melhorar a qualidade de código de teste.

Este trabalho está organizado da seguinte forma. Fundamentação teórica é apresentada na Seção 2. O método de pesquisa para a realização do trabalho é detalhado na Seção 3. Avaliação dos resultados é discutida na Seção 4. Contribuições esperadas são apresentadas na Seção 5. Trabalhos relacionados são apresentados na Seção 6.

2. Fundamentação Teórica

2.1. Test Smells

Para minimizar erros no código de produção, esforços têm sido voltados para o desenvolvimento de código de testes com mais qualidade [Tufano *et al.*, 2016]. Contudo, a presença de *TS* tem impacto negativo sobre manutenção do código de teste por gerarem testes complexos e difíceis de entender e modificar, prejudicando a independência e a estabilidade dos testes [Van Rompaey, 2006]. Como indicadores de possíveis *TS*, tem-se a maneira como os casos de teste são documentados ou organizados e como os casos de teste interagem com outros testes, com o código de produção e com recursos externos [Tufano *et al.*, 2016]. Muitas vezes, *TS* não são percebidos pelos testadores como problemas reais de *design*. Além disso, o responsável pela inserção do *TS* pode não identificar a sua presença em seu próprio código [Spadini *et al.*, 2018]. Contudo, estudos destacaram empiricamente o efeito negativo de *TS* quanto à compreensibilidade e à manutenibilidade do código, destacando a importância de investir esforços no desenvolvimento de ferramentas para *TS* [Tufano *et al.*, 2016].

Na literatura, foi introduzido um conjunto com 30 tipos de *TS* [Van Deursen *et al.*, 2001; Meszaros, 2006; Peruma, 2019]. Desse conjunto, foram selecionados 21 *TS* para serem investigados durante a evolução do código de teste apresentada neste trabalho. Esses *TS* foram escolhidos por terem sido formalmente validados em uma pesquisa com desenvolvedores [Peruma, 2019]. Além disso, 21 *TS* são detectados pela ferramenta utilizada neste trabalho (JNose Test [Virgínio *et al.*, 2019]).

2.2. Visualização de Software

Visualização é o processo de transformar informações em uma forma visual, permitindo aos usuários observar a informação. Visualização de software tem sido utilizada na Engenharia de Software para facilitar a tarefa de compreensão, de manutenção e de evolução de sistemas de software [Diehl, 2007; Conceição *et al.*, 2012]. Ao utilizar recursos gráficos, a compreensão torna-se mais intuitiva do que representações textuais. Ao representar um sistema de software por meio de recursos visuais, a sua representação mental pode ser clara, pois os seres humanos possuem mais facilidade em entender informações representadas de forma gráfica [Caserta; Zendra, 2011]. Em um estudo [Cruz *et al.*, 2016], 20 técnicas de visualização de software foram identificadas.

Um sistema de software deve ser constantemente mantido. Para isso, evoluções no código de produção são necessárias para atender novos requisitos, corrigir *bugs/design* ou melhorar o desempenho [Sulaiman; 2004], o que acarreta na evolução do código de teste. A compreensão dessas evoluções pode ser usada para encontrar a origem de problemas atuais ou obter informações que possibilitam prever características futuras desses sistemas. No entanto, o aumento da quantidade de informações, funções e quantidade de código pode deixar o sistema mais complexo, dificultando sua compreensão. Desse modo, técnicas de visualização de software têm sido usadas para representar essas evoluções [Bastos; Costa, 2016].

Para a realização do trabalho de mestrado será utilizada as técnicas **Graphs** (**Grafos**) e **Timeline**. *Graphs* é utilizada para a representação estrutural dos dados, sendo capaz de mostrar a ligação entre os artefatos de software [Herman *et al.*, 2000].. *Timeline* é utilizada como estratégia de análise da evolução de software, retratando a

evolução do projeto de software considerando várias versões disponíveis para análise. Dadas 2 versões de um projeto de software (v_1 e v_2), essa análise permite a comparação dessas versões sendo possível analisar as mudanças entre diferentes versões sobre o ciclo de evolução do software. Além disso, será utilizada a perspectiva de autoria para identificar quem possivelmente inclui (assumiu a autoria de) *TS*, ela exibe as atividades dos autores durante a evolução do software (e.g., quantos *commits* executaram ou em quais arquivos eles trabalharam [Novais *et al.*, 2013]).

3. Método de Pesquisa

O método de pesquisa (Figura 1) a ser utilizado para o desenvolvimento da pesquisa de Mestrado possui quatro etapas:

- **Etapa 1 - Revisão da Literatura (Estado: Concluído).** Nessa etapa, foram buscados na literatura artigos sobre *TS*, técnicas de visualização de software para *TS*, evolução de *TS* e identificação de autores de *TS*. Essa busca foi realizada nas bases de bibliotecas científicas virtuais, por serem amplamente utilizadas no meio acadêmico: i) IEEE Xplore (<http://ieeexplore.ieee.org>); ii) ACM Digital Library (<http://dl.acm.org>); iii) Scopus (<http://www.scopus.com>); iv) Ei Compendex (<http://www.engineeringvillage.com/>); e v) Google Scholar (<https://scholar.google.com.br/>);

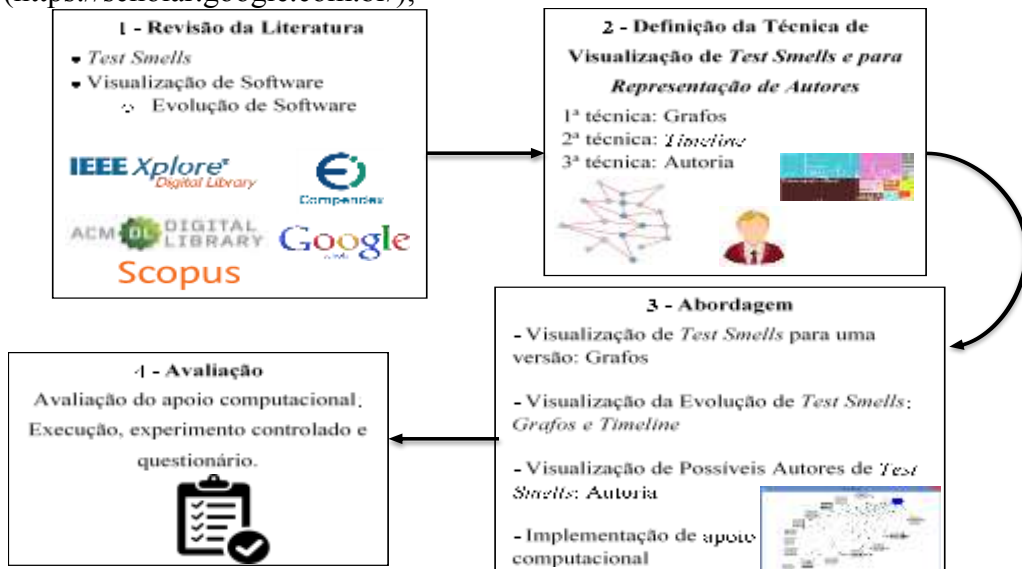


Figura 1. Método da Pesquisa

- **Etapa 2 - Definição de Técnicas para a Visualização de *TS* (Estado: Concluído).** Nessa fase, foi definida perante as técnicas identificadas, as técnicas mais adequadas ao propósito de exibir visualmente os *TS*. Primeiramente, para a visualização das ocorrências de *TS* foi escolhida a técnica de *Graph* por ser capaz de mostrar a ligação entre os artefatos de software. Assim, pudemos representar os *TS* e suas relações com o projeto, as classes de teste e com autores. Para a visualização da evolução de *TS* (duas versões) foi utilizada também a técnica de *Graph* e foi definida mais uma a técnica denominada *Timeline* que é mais intuitiva para exibir a evolução, pois fornece visualmente um comparativo. Os possíveis autores são representados nos grafos;

evolução do código de teste e com a aplicação de questionário para coletar pontos positivos, negativos e melhorias para a ferramenta proposta.

5. Contribuições Esperadas

A abordagem desenvolvida será avaliada com a utilização quanto a sua eficácia em atender ao propósito para identificar se realmente facilita visualizar/acompanhar ocorrências de *TS* nos códigos de testes. O objetivo é verificar se a abordagem proposta pode apoiar os Gerentes de Testes/Testadores no processo de tomada de decisão e se facilita encontrar problemas no código de teste. Com isso, pretende-se responder as seguintes questões: i) “Qual é tempo para visualizar *TS* em várias versões de um sistema de software?” ii) “A identificação dos possíveis autores facilitou o processo de tomada de decisão ao Gerentes de Testes/Testadores?”, iii) “Visualização de ocorrências de *TS* no código de teste foi intuitiva para Gerentes de Testes/Testadores?” e iv) “Durante a evolução do código de teste os *TS* diminuíram ou aumentaram?”.

6. Trabalhos Relacionados

A ferramenta TestQ [Breugelmans; Van Rompaey, 2008] suporta várias maneiras de visualizar dados para explorar a estrutura do código de teste e quantificar a presença de 12 tipos de *TS*. Essa ferramenta apresenta a visualização *Test Suite Topology*, representando os dados da estrutura da suíte de testes em uma árvore expansível. A ferramenta também apresenta a visualização *Smell Flower* que permite ao usuário explorar pontos de *TS* (*hotspots*) por meio de sua representação em gráficos e quantificar instâncias individuais usando um gráfico de pizza.

A ferramenta VITrUM [Pecorelli *et al.*, 2020] inclui o cálculo de medidas de código de teste relacionadas à coesão, ao acoplamento e à complexidade, detecta 7 tipos de *TS* e calcula a cobertura das classes de teste. Como resultado, ela apresenta uma lista de classes de teste em três cores (vermelho: classes de teste afetadas por *TS* com valores de medidas de código de teste acima dos limites definidos, amarelo: classes de teste afetadas por *TS* com valores de medidas de código de teste nos limites definidos e cinza: classes de teste sem *TS*). Além disso, o usuário pode filtrar os resultados por tipo de *TS* para exibir sua evolução em um gráfico ao longo do desenvolvimento do projeto.

Diferentemente das ferramentas apresentadas, a ferramenta proposta se concentra em mostrar a ocorrência de 21 tipos de *TS* por meio de duas técnicas de visualização (Grafos e *Timeline*), apresentando suas relações de projeto, classes, e possíveis autores. As ferramentas encontradas não abrangem quantidade maior de *TS* associados às técnicas de visualização de software nem apresentou a comparação durante a evolução das versões e possíveis autores (autoria).

Referências

- Bastos, C.; Costa, H. (2016) “Uma Abordagem para Visualização da Evolução de Código Morto em Sistemas de Software Orientados a Objetos”. In: Workshop de Teses e Dissertações em Qualidade de Software, p.33.
- Breugelmans, M.; Van Rompaey, B. (2008) “Testq: Exploring Structural and Maintenance Characteristics of Unit Test Suites” In: International Workshop on Advanced Software Development Tools and Techniques.
- Caserta, P.; Zendra, O. (2011) “Visualization of the Static Aspects of Software: A Survey”. In: Visualization and Computer Graphics, v. 17, n. 7, p. 913-933.

- Conceição, C. F. R.; Carneiro, G.; David, J. M. N. (2012) “Usando Recursos de Visualização Enriquecidos com Elementos de Percepção para a Compreensão de Software em um Ambiente de Desenvolvimento Distribuído”. In: WBVS. p. 41-48.
- Crespo, A. N., Silva, O. J., Borges, C. A., Salviano, C. F., Argollo, M., Jino, M. (2004) “Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo”. In: Simpósio Brasileiro de Qualidade de Software, p. 271-285.
- Cruz, A., Bastos, C., Afonso, P., Costa, H. (2016) “Software Visualization Tools and Techniques: A Systematic Review of the Literature”. In: SCCC. p. 1-12.
- Diehl, S. (2007) “Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software”, Berlim: Springer. 186p.
- Garousi, V., Kucuk, B., Felderer, M. (2018) “What We Know About Smells in Software Test Code”. In: IEEE Software. p. 61-73.
- Greiler, M.; Van Deursen, A.; Storey, M. A. (2013) “Automated Detection of Test Fixture Strategies and Smells”. In: ICST. p. 322-331.
- Herman, I., G. Melançon, and M. S. Marshall (2000). "Graph visualization and navigation in information visualization: a survey". In: IEEE Transactions on Visualization and Computer Graphics. p. 24-43.
- Kim, D. J. (2019) “An Empirical Study on the Evolution of Test Smell”, <https://djaekim.github.io/djae.io/img/EvolutionOfTestSmell.pdf>. Acesso: 02/04/2020.
- Novais, L. R.; Nunes, C.; Garcia, A.; Mendonça, M. Sourceminer Evolution: A Tool for Supporting Feature Evolution Comprehension. In: ICSM. p. 508-511, 2013.
- Palomba, F.; Zaidman, A.; De Lucia, A. (2018) “Automatic Test Smell Detection Using Information Retrieval Techniques”. In: ICSME. p. 311-322.
- Pecorelli, F.; do Lillo, G.; Palomba, F.; de Lucia, A. (2020) “VITRuM-A Plug-in for the Visualization of Test-Related Metrics.” International Conference on Advanced Visual Interfaces, 1-3.
- Peruma, A.; Almalki, K.; Newman, C. D.; Mkaouer, M. W.; Ouni, A.; Palomba, F. (2019) “On the Distribution of Test Smells in Open Source Android Applications: An Exploratory Study”. In: International Conference on Computer Science and Software Engineering. p. 193-202.
- Silva, R. O.; Machado, G. B. G.; Viana, G. B.; dos Santos Silva, J. S. (2016) “O Processo de Teste de Software”. In: Tecnologias em Projeção. v. 7.
- Spadini, D.; Palomba, F.; Zaidman, A.; Bruntink, M.; Bacchelli, A. (2018) “On the Relation of Test Smells to Software Code Quality”. In: ICSME. p. 1-12.
- Sulaiman, S. (2004) “Viewing Software Artifacts for Different Software Maintenance Categories Using Graph Representations”. In: Journal of Science. v. 17, p. 55-67.
- Tufano, M.; Palomba, F.; Bavota, G.; Di Penta, M.; Oliveto, R.; De Lucia, A.; Poshyanyk, D. (2016) “An Empirical Investigation into the Nature of Test Smells”. In: International Conference on Automated Software Engineering, p. 4-15.
- Van Deursen, A.; Moonen, L.; Van Den Bergh, A.; Kok, G. (2001) “Refactoring Test Code”. In Conference on extreme programming and flexible processes in software engineering. p. 92-95.
- Van Rompaey, B., Du Bois, B., Demeyer, S. (2006) “Characterizing the Relative Significance of a Test Smell”. In: ICSM p. 391-400.
- Virgínio, T.; Santana, R.; Martins, L. A.; Soares, L. R.; Costa, H.; Machado, I. (2019) “On the Influence of Test Smells on Test Coverage”. In: SBES. p. 467-471.
- Yusifoglu, V. G.; Amannejad, Y.; Can, A. B. (2015) “Software Test-Code Engineering: A Systematic Mapping”. In: Information and Software Technology. p. 123-147.