Um *framework* para visualização da evolução arquitetural de sistemas de *software* baseado em cenários de casos de uso

Leo Silva[1 2], Uirá Kulesza[1]

[1]Departamento de Informática e Matemática Aplicada (DIMAp) - Universidade Federal do Rio Grande do Norte (UFRN)
Natal – RN – Brasil

[2]Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte (IFRN)
Natal – RN – Brasil

leo.silva@ifrn.edu.br, uira@dimap.ufrn.br

Nível: Mestrado
Ano de ingresso no programa: 2015
Data prevista da aprovação da proposta de dissertação: outubro de 2016
Época prevista de conclusão: abril de 2017
Eventos Relacionados: SBES

Abstract. *The software systems are widely integrated into the society through various devices. Many of them are complex due to their large scale, making their architecture also complex to understand and maintain. The software visualization area uses techniques whose goal is to improve the understanding complex software artifacts in order to improve the productivity of the software process. This paper proposes to use these techniques to assist the analysis of software architecture evolution. A visualization tool is being developed to show the use case scenarios that have degraded their performance quality attribute between software versions. This tool will be evaluated through an empirical study in a real software development environment.*

Keywords: *software architecture, software visualization, software evolution*

Resumo. *Os softwares estão cada vez mais inseridos na sociedade mediante vários dispositivos. Muitos deles são complexos devido a sua larga escala, tornando a sua arquitetura também complexa para entender e manter. A área de visualização de software utiliza técnicas cujo objetivo é melhorar o entendimento e tornar mais produtivo o processo de desenvolvimento do software. Este trabalho propõe usar essas técnicas para auxiliar a análise da evolução arquitetural de softwares. Para isso, está sendo desenvolvida uma ferramenta que indica os cenários de casos de uso que tem seu atributo de qualidade de desempenho degradado entre versões. Ela será avaliada através de um estudo empírico em um ambiente real de desenvolvimento de software.*

Palavras-chave*: arquitetura de software, visualização de software, evolução de software*

1. Introdução

A importância dos *softwares* hoje em dia é cada vez mais percebida, seja através dos celulares e *tablets*, dos computadores, *smart* TVs ou até mesmo dos carros. Caserta e Zendra (2011) dizem que um *software* se torna rapidamente complexo quando o seu tamanho aumenta, trazendo dificuldades para o seu entendimento, manutenção e evolução. Além disso, muitos desses *softwares* podem ter requisitos de qualidade críticos como segurança e desempenho. Todos esses sistemas possuem uma arquitetura que serve de base para o seu desenvolvimento.

A área de arquitetura de *software* (AS) propõe técnicas, métodos e ferramentas para auxiliar no projeto do *software* e análise de seus atributos de qualidade. Embora as realizações da área sejam notáveis, existem sempre desafios inerentes ao desenvolvimento dos sistemas modernos, tais como, a complexidade devido a sua larga escala, os altos custos na mudança e a erosão do projeto [Jansen e Bosch 2005].

Além disso, os *softwares* são virtuais e intangíveis, sendo difícil criar representações mentais totais ou parciais deles [Roy e Graham 2008]. Nesse contexto, a área de visualização de *software* (VS) é definida como o uso de representações visuais para melhorar o entendimento e compreensão de diferentes aspectos de um sistema de *software* [Carpendale e Ghanam 2008].

Um dos principais tópicos na área de visualização da evolução de *software* é a visualização da evolução de arquiteturas de *software* [Caserta e Zendra 2011]. Ter uma visão da evolução de todo o sistema é considerada fundamental, porque ela pode explicar e documentar o estado atual do projeto do *software* [Roy e Graham 2008]. Uma vez que um *software* está em constante mudança para atender a novos requisitos, adaptar-se a novas tecnologias ou reparar erros, a inexistência de atividades para entender a evolução da sua arquitetura pode levar a sua degradação [D'Ambros e Lanza 2009], fazendo com que atributos de qualidade inicialmente definidos deixem de ser atendidos. É especialmente difícil visualizar a evolução da arquitetura pelo fato de ser adicionada a variável tempo, fazendo com que a quantidade de dados envolvidos aumente uma vez que todas as versões do *software* passam a ser levadas em consideração [Caserta e Zendra 2011] [Khan et al. 2012]. Com relação à aspectos da VS, alguns autores sugerem que questões como usabilidade [Caserta e Zendra 2011], efetividade, interação com o usuário, complexidade visual e técnicas de navegabilidade [Carpendale e Ghanam 2008] devem ser considerados. Algumas soluções existentes pecam por mostrar a evolução da arquitetura sob vários aspectos, tornando-as, em alguns casos, pouco eficientes.

Esta proposta de trabalho objetiva aplicar técnicas de VS para acompanhar a evolução arquitetural de um *software*. Pretende-se usar a técnica de cenários de casos de uso [Kazman et al. 1996] para analisar arquiteturas de sistemas *web*. A ferramenta busca apontar quais desses cenários degradaram ou melhoraram o atributo de qualidade de desempenho. Tal atributo foi escolhido por se tratar de uma propriedade crítica para a maioria dos sistemas de *software* atuais. A ferramenta proposta visa prover um melhor entendimento da arquitetura de um *software* por parte dos arquitetos e desenvolvedores, auxiliando-os a: (i) perceber quais cenários de casos de uso degradaram ou melhoraram o seu desempenho; (ii) identificar exatamente qual parte do código-fonte foi o responsável por uma dada degradação; e (iii) identificar quando e qual desenvolvedor realizou mudanças relacionadas a degradação.

2. Fundamentação Teórica

Conceitos de visualização de *software* (VS) e arquitetura de *software* (AS) são importantes para este trabalho. Esta seção oferece breve definições sobre esses temas. A VS é uma parte da visualização da informação. É usada para tornar a estrutura, comportamento e evolução do *software*, como a organização do código-fonte, o estado, e os *bugs*, mais compreensíveis [Diehl 2007]. Diehl (2007) enfatiza que o objetivo da VS é ajudar a compreender sistemas de *software* e aumentar a produtividade do processo de desenvolvimento.

Bass (2007) define AS como sendo "a estrutura ou estruturas do sistema, que compreendem os elementos de *software*, as propriedades externamente visíveis desses elementos e as relações entre eles". Adicionalmente, de acordo com Taylor et al. (2009), a arquitetura de um *software* incorpora todas as decisões de projeto tomadas pelos seus arquitetos, que podem afetar muitos dos seus módulos, incluindo sua estrutura e atributos de qualidade [Taylor et al. 2009]. A arquitetura é essencial para o desenvolvimento de um *software*, visando atender adequadamente os seus requisitos relacionados com os atributos de qualidade [Kazman et al. 2001], tais quais: desempenho, confiabilidade, segurança e portabilidade [Bass 2007]. No *framework* proposto, a arquitetura do *software* será analisada com o objetivo de coletar informações sobre o atributo de qualidade de desempenho.

Para essa análise, será usada uma estratégia comum de avaliação de arquiteturas: a técnica de cenários de casos de uso. Os cenários são definidos pela interação entre os *stakeholders* e o sistema. De acordo com Kazman et al. (2001), existem três tipos de cenários que podem ser usados nesse processo: cenários de casos de uso, cenários de crescimento e cenários exploratórios. No âmbito deste trabalho, será usado o primeiro tipo, que é definido como uma interação completa do usuário com o sistema em execução. Esses cenários de caso de uso serão executados a fim de determinar se a arquitetura satisfaz às restrições impostas pelo atributo de qualidade de desempenho.

3. Contribuições Esperadas

As seguintes contribuições são esperadas ao término do desenvolvimento deste trabalho:

a) Desenvolver um *framework* para visualização da evolução de AS utilizando a abordagem baseada em cenários;

b) Instanciar o *framework* para desenvolvimento de uma ferramenta que indica que cenários degradaram ou melhoraram o atributo de qualidade de desempenho durante a evolução arquitetural de um dado sistema;

c) Avaliar o *framework* no contexto de equipes reais de desenvolvimento de sistemas *web*, através da condução de estudos empíricos que tragam evidências da utilidade do *framework* proposto.

4. Estado Atual do Trabalho

De acordo com a literatura, existem várias ferramentas de VS que tratam da visualização da sua evolução arquitetural, cada uma com suas particularidades. Caserta e Zendra (2011) apontam que essas ferramentas apresentam a evolução arquitetural a partir de: (i) como a arquitetura global é alterada a cada versão, incluindo mudanças no código fonte; (ii) como os relacionamentos entre os componentes evoluem; e (iii) como

as métricas evoluem a cada *release*. Nas pesquisas realizadas até o momento da escrita deste artigo, não foram encontradas soluções de visualização da evolução arquitetural de um sistema a partir da perspectiva de cenários de casos de uso, com foco no atributo de qualidade de desempenho.

Após a revisão literária, foi idealizado um *framework* cujo objetivo é mostrar a evolução arquitetural de um *software* desenvolvido em Java através da realização de análises estáticas e dinâmicas baseadas em cenários de casos de uso. Uma vez feitas as análises nas versões desejadas, serão usadas técnicas de VS para mostrar a evolução com foco no atributo de qualidade de desempenho. A linguagem Java foi escolhida por: (i) ser atualmente uma das mais utilizadas no mundo, de acordo com a empresa TIOBE (http://tiobe.com/tiobe-index/); (ii) estar presente no desenvolvimento de *software standalone*, *web* e móvel, e; (iii) embora nos últimos anos tenham se popularizado linguagens como Python e Ruby, o Java é bem aceito no mercado brasileiro e mundial, em especial para o desenvolvimento *web*, fazendo com que existam vários sistemas desse tipo como alvos potenciais para a avaliação da ferramenta proposta.

Ferramentas de *profiling* para Java possuem análise de desempenho, porém com características diferentes. A VisualVM (http://visualvm.java.net) exibe o tempo de execução dos métodos em tempo real e o usuário pode realizar *snapshots*, entretanto, sem comparação entre eles ou de versões anteriores do *software*. Já a JProfiler (https://www.ej-technologies.com/products/jprofiler/overview.html), ferramenta paga, pode exibir o *call graph* dos métodos em tempo real, com seus respectivos tempos de execuções. *Snapshots* de determinados momentos da execução podem ser registrados e comparados. Entretanto, a comparação não é feita entre as versões anteriores do *software*. A YourKit Java Profiler (https://www.yourkit.com/java/profiler/features/) possui funcionalidades semelhantes a JProfiler. Entretanto, também é uma ferramenta paga e não realiza comparação entre versões do sistema.

Como mencionado, o *framework* proposto visa mostrar a evolução da arquitetura com o foco no atributo de qualidade de desempenho entre as diferentes versões do *software*. A análise da arquitetura requer a execução de funcionalidades do sistema que, por sua vez, executam vários módulos. Essa execução é realizada para cada uma das versões do *software* envolvidas na análise. Dessa forma, quando há degradação ou melhoria no atributo de qualidade em questão para determinada funcionalidade, nem sempre todos os módulos analisados são responsáveis por essa variação. Percebe-se que a análise de desempenho é distinta da análise de propriedades estáticas do *software*, como tamanho, coesão e acoplamento. O *framework* é composto pelos 5 (cinco) módulos descritos a seguir.

No *módulo de engenharia reversa*, a partir do código-fonte do sistema, é realizada uma engenharia reversa com o objetivo de obter as classes, atributos, métodos e relacionamentos. Para auxiliar o desenvolvimento deste módulo, a biblioteca de engenharia reversa PlantUML (http://plantuml.com) foi selecionada, por se tratar de (i) um componente que não está acoplado a nenhuma ferramenta de modelagem UML ou IDE; (ii) ser gratuita; e (iii) possuir uma linguagem própria para definir os diagramas. Os dados obtidos pela engenharia reversa serão usados quando os usuários desejarem extrair mais detalhes sobre os cenários analisados. A implementação deste módulo está finalizada.

Já no m*ódulo de integração com VCS (do inglês version control system)*, o código-fonte analisado será obtido e manipulado. Uma vez identificado um cenário que degradou ou melhorou, o usuário poderá obter informações sobre o *commit* da alteração, bem como qual *issue* está relacionada àquele código, se existir. A implementação deste módulo está finalizada.

O *módulo de integração com analisador*, por sua vez, visa integrar o *framework* a uma ferramenta de análise arquitetural baseada em técnicas de análise estática e dinâmica que foi desenvolvida dentro do grupo de pesquisa por Pinto (2015). Uma análise dinâmica do desempenho de cenários de interesse executados pelo sistema é realizada de forma automatizada. A integração se faz necessária uma vez que serão usados os resultados gerados por essa análise para obter os dados necessários para a visualização. A implementação deste módulo está em andamento.

No m*ódulo core*, são feitas as solicitações para a engenharia reversa, as integrações, a manipulação do banco de dados, o processamento que for necessário e o envio dos dados ao módulo de visualização. A implementação deste módulo está finalizada.

Por fim, é no *módulo de visualização* que serão usadas as técnicas de VS para mostrar a evolução arquitetural. A partir dos resultados coletados do analisador, será exibida em uma página *web* a visualização de (i) pacotes do sistema, destacando, através de cores, quais deles tiveram o atributo de qualidade de desempenho degradados ou melhorados. Os pacotes representam um conjunto de classes e podem ser usados para mostrar a estrutura e organização dos módulos ou componentes de um sistema. O usuário poderá aprofundar a visualização clicando no pacote desejado e, então, serão exibidas as classes daquele pacote ou seus sub-pacotes. Nesse nível de visualização, os métodos que afetaram o atributo de qualidade serão, também, destacados por cores. O usuário poderá visualizar o código-fonte do método ou obter informações do VCS. O *framework* permitirá, ainda, a visualização de (ii) cenários de casos de uso. Nessa visualização, serão mostrados os cenários analisados, destacados com cores para indicar os que afetaram o atributo de qualidade analisado. O usuário poderá escolher um dos cenários para aprofundar a visualização. Feito isso, será exibido o grafo de chamadas dos métodos executados, também diferenciados através de cores os que afetaram o cenário. A partir desse grafo, o usuário poderá escolher o método a ser visualizado e, depois, será exibida a visualização da classe a qual o método pertence.

Vale salientar que, como o *framework* (e os módulos) está em constante evolução, podem surgir novas funcionalidades e novas visualizações que não foram inicialmente previstas ou planejadas, no entanto, sem modificar drasticamente o seu funcionamento global descrito nesta seção.

5. Trabalhos Relacionados

Telea et al. (2008) propõem uma técnica de visualização, chamada *Code Flows*, que mostra a evolução do código-fonte através de uma linha contendo todas as versões. Essa técnica permite rastrear visualmente a evolução de um dado fragmento de código.

Uma metáfora com cidades foi usada por Wettel e Lanza (2007) para mostrar a evolução de arquiteturas de *software*: *CodeCity*. A ferramenta exibe a evolução em uma visualização 3D analisando o histórico do sistema enquanto que a cidade se atualiza

para refletir a versão atual. Para resolver um problema com relação ao desaparecimento de classes ou métodos no *CodeCity*, Wettel e Lanza (2008) criaram uma visualização de *timeline* para mostrar a evolução de uma classe. Essa técnica se baseia na metáfora de um edifício, onde cada prédio representa uma classe e cada tijolo um método.

Pinzger et al. (2005) propõem uma técnica de visualização chamada *RelVis*. Essa técnica pode exibir várias métricas de *software* pertencentes aos módulos e seus relacionamentos, utilizando grafos simples e diagramas *Kiviat* para representar os valores de métricas. As versões do *software* são mostradas em ordem de lançamento, cada um com uma cor. Um dos problemas dessa abordagem é que as faixas coloridas, muitas vezes, se sobrepõem, tornando difícil de visualizar o valor de determinadas métricas [Caserta e Zendra 2011].

Lanza (2001) propôs uma técnica de visualização que exibe a evolução de todo o sistema em uma imagem: *The Evolution Matrix*. Nessa técnica são combinadas a visualização e as métricas do *software* em caixas bidimensionais, onde o número de métodos é representado na largura e o número de atributos na altura. Cada versão do *software* é mostrada em uma coluna e cada linha representa uma classe diferente.

Os trabalhos relacionados aqui mostrados se diferenciam do proposto pelo fato de este propor uma visualização da evolução da arquitetura de um *software* a partir do uso de técnicas de análise estática e dinâmica. Além disso, este trabalho foca especificamente na análise da arquitetura baseada em cenários com foco no atributo de qualidade de desempenho. O resultado dessa análise é mostrado usando técnicas de visualização descritas na seção 4 deste artigo.

6. Avaliação dos Resultados

A avaliação do *framework* proposto nessa dissertação será conduzida da seguinte forma:

- Instanciação do *framework* para sistemas *web* com o intuito de analisar a evolução arquitetural da perspectiva do atributo de qualidade de desempenho;
- Conduzir um estudo empírico de utilização da ferramenta em um ambiente real de desenvolvimento de sistemas *web*. O objetivo será analisar os benefícios da instância do *framework*, assim como identificar eventuais carências e melhorias a serem realizadas. Tal estudo pretende apresentar resultados coletados automaticamente pela instância do *framework* a respeito de cenários reais de evolução da arquitetura, assim como coletar *feedback* de desenvolvedores responsáveis pelo sistema, na forma de uma avaliação qualitativa. Essa metodologia já foi utilizada com sucesso por um outro trabalho de pesquisa conduzida pelo grupo de pesquisa [Pinto et al. 2015].

Referências

Bass, L. (2007). “Software architecture in practice”. Pearson Education India.

Carpendale, S.; & Ghanam, Y. (2008). “A survey paper on software architecture visualization”. Technical report, University of Calgary.

Caserta, P.; & Zendra, O. (2011). “Visualization of the static aspects of software: a survey”. *Visualization and Computer Graphics, IEEE Transactions on*, *17*(7), 913-933.

D'Ambros, M.; & Lanza, M. (2009). “Visual software evolution reconstruction”. *Journal of Software Maintenance and Evolution: Research and Practice*, *21*(3), 217-232.

Diehl, S. (2007). “Software visualization: visualizing the structure, behaviour, and evolution of software”. Springer Science & Business Media.

Jansen, A.; & Bosch, J. (2005). “Software architecture as a set of architectural design decisions”. Em *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on* (pp. 109-120). IEEE.

Kazman, R.; Abowd, G.; Bass, L; Clements, P. (1996). “Scenario-Based Analysis of Software Architecture”. *IEEE Software*, *13*(6), 47-55.

Kazman, R.; Klein, M.; & Clements, P. (2001). “Evaluating Software Architectures-Methods and Case Studies”. Addison-Wesley, Reading, Mass., 2001.

Khan, T.; Barthel, H.; Ebert, A.; & Liggesmeyer, P. (2012). “Visualization and evolution of software architectures”. Em *OASIcs-OpenAccess Series in Informatics* (Vol. 27). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Lanza, M. (2001). “The evolution matrix: Recovering software evolution using software visualization techniques”. Em *Proceedings of the 4th international workshop on principles of software evolution* (pp. 37-42). ACM.

Pinto, F. A. P. (2015). “An Automated Approach for Performance Deviation Analysis of Evolving Software Systems”. 2015. 154f. Tese (Doutorado) – Programa de Pós-Graduação em Sistemas e Computação (PPgSC) - UFRN, Natal/RN. 2015.

Pinto, F.; Kulesza, U.; & Treude, C. (2015). “Automating the performance deviation analysis for multiple system releases: An evolutionary study”. *IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2015*, pp. 201-210. IEEE Computer.

Pinzger, M.; Gall, H.; Fischer, M.; & Lanza, M. (2005). “Visualizing multiple evolution metrics”. Em *Proceedings of the 2005 ACM Symposium on Software Visualization*, pp. 67-75...

Roy, B.; Graham, T. C. N. (2008). “Methods for Evaluating Software Architecture: A Survey”. Technical *report* No. 2008-545, School of Computing, Queen's University at Kingston. Ontario, Canada.

Taylor, R. N.; Medvidovic, N.; Dashofy, E. M. (2009). “Software architecture: foundations, theory, and practice”. Wiley Publishing.

Telea, A.; Auber, D. (2008). “Code flows: Visualizing structural evolution of source code”. Em *Computer Graphics Forum* (Vol. 27, No. 3, pp. 831-838). Blackwell Publishing Ltd.

Wettel, R.; Lanza, M. (2007). “Visualizing software systems as cities”. Em *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on* (pp. 92-99). IEEE.

Wettel, R.; Lanza, M. (2008). “Visual exploration of large-scale system evolution”. Em *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on* (pp. 219-228). IEEE.