Refatorações Arquiteturais: Um estudo sobre o efeito de Mover Classe na Arquitetura de Sistemas

Vagner Clementino[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)

vagnercs@dcc.ufmg.br

Resumo. TODO

1. Introdução

A atividade de refatoração têm por objetivo alterar o código fonte de um software sem modificar o seu comportamento. Em última instância o objetivo de refatorar é melhorar a qualidade interna do sistema[1999, Opdyke 1992]. Sua importância é reconhecida tanto na literatura quanto na indústria, no qual, nesta última, é possível verificar a proposição de processos de desenvolvimento de software ágeis que incorporam a refatoração como atividade rotineira [Beck and Fowler 2000]. No últimos anos, pesquisas foram realizadas com o objetivo de entender com qual frequência os desenvolvedores aplicam diferentes tipos de refatoração [Murphy-Hill et al. 2009], a sua relação entre a correção de *bugs*[Kim et al. 2011] e testes [Kim and Rachatasumrit 2012] e a percepção da mesma pelos desenvolvedores[Kim et al. 2012].

No últimos anos estudos vêm focando em entender as motivações que levam os desenvolvedores a realizem refatoração. Existe um consenso que a motivação original é remover porções de código com baixa qualidade conhecidos como *Bad Smells*[1999]. Todavia, estudos demonstraram que o desenvolvedores refatoram para outros fins, como por exemplo, a refatoração *Extrair Método* que pode ser utilizada para fins de extensão do sistema [Tsantalis et al. 2013]ou ainda para reutilização do código [Silva et al. 2015], dentre outras motivações.

Apesar da existência de estudos relativos à motivação da refatoração, ao bem do nosso conhecimento, não existem trabalhos que relacionem a atividade de refatorar com mudanças na arquitetura do sistema. Ou seja, refatorações que têm por objetivo alterar a arquitetura do software, reorganizar o código existente em uma nova camada lógica ou cujo objetivo é tratar o problema da *Erosão Arquitetural*. O processo de Erosão arquitetural e conhecido como os desvios ocorridos no código de um sistema que causam violação de alguma regra arquitetural previamente estabelecida[Perry and Wolf 1992]. Violações na arquitetura podem ser de dois tipos: *divergência*, quando uma dependência que existe no código fonte viola a arquitetura planejada; *ausência*, caso onde o código fonte não estabelece uma dependência que é prescrita na arquitetura planejada [Passos et al. 2010].

Neste sentido, este trabalho se propõe em analisar a relação entre mudanças na arquitetura de um sistema e a refatoração *Mover Classe*. A fim de investigar tal relação iremos analisar diversas versões de 03 sistemas de código aberto desenvolvidos em Java visando responder as seguintes questões de pesquisa:

RQ1 Com qual frequência a refatoração Mover Classe tem por objetivo alterar a arquitetura do sistema?

RQ2 Com qual frequência o desenvolvedor informa que a refatoração teve por objetivo alterar a arquitetura do sistema?

RQ3 A refatoração Mover Classe prevalece na resolução de desvios arquiteturais do tipo divergência ou ausência?

Ao responder as questões proposta neste trabalho entendemos que iremos contribuir no aumento do entendimento das razões que levam os desenvolvedores a realizar refatorações. Esta informação poderá ser utilizada posteriormente na construção de ferramentas que ajudem os times de desenvolvimento em tarefe relativas à mudança da arquitetura do software. Além disso será possível verificar a relação entre a atividade de refatoração e a mudança de arquitetura de um sistema.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 descreve a metodologia utilizada neste estudo; a Seção 3 apresenta os resultados e responde as questões de pesquisas propostas; na Seção 4 realiza-se uma discussão sobre as ameaças à validade do trabalho; a Seção 5 apresenta os trabalhos relacionados à análise da atividade e detecção de refatoração; a Seção 6 sumariza o artigo e discute suas principais contribuições.

2. Metodologia

3. Resultados

4. Ameaças à Validade

5. Trabalhos Relacionados

6. Conclusão

Referências

(1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Beck, K. and Fowler, M. (2000). *Planning Extreme Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Kim, M., Cai, D., and Kim, S. (2011). An empirical investigation into the role of api-level refactorings during software evolution. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 151–160, New York, NY, USA. ACM.

Kim, M. and Rachatasumrit, N. (2012). An empirical investigation into the impact of refactoring on regression testing. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)*, ICSM '12, pages 357–366, Washington, DC, USA. IEEE Computer Society.

Kim, M., Zimmermann, T., and Nagappan, N. (2012). A field study of refactoring challenges and benefits. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 50:1–50:11, New York, NY, USA. ACM.

Murphy-Hill, E., Parnin, C., and Black, A. P. (2009). How we refactor, and how we know it. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 287–297, Washington, DC, USA. IEEE Computer Society.

Opdyke, W. F. (1992). *Refactoring Object-oriented Frameworks*. PhD thesis, Champaign, IL, USA. UMI Order No. GAX93-05645.

Passos, L., Terra, R., Valente, M. T., Diniz, R., and das Chagas Mendonca, N. (2010). Static architecture-conformance checking: An illustrative overview. *IEEE Software*, 27(5):82–89.

Perry, D. E. and Wolf, A. L. (1992). Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52.

Silva, D., Valente, M. T., and Figueiredo, E. (2015). Um estudo sobre extração de métodos para reutilização de código.

Tsantalis, N., Guana, V., Stroulia, E., and Hindle, A. (2013). A Multidimensional Empirical Study on Refactoring Activity. *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, pages 132–146.