RAID: Tool Support for Refactoring-Aware Code Reviews

Rodrigo Brito[1], **Marco Tulio Valente**[1]

[1]Departamento de Ciência da Computação
Universidade Federal do Minas Gerais (UFMG)
Belo Horizonte, MG – Brasil

{britorodrigo,mtov}@dcc.ufmg.br

Abstract. *Code review is a key development practice that contributes to improve software quality and to foster knowledge sharing among developers. However, code review usually takes time and demands detailed and time-consuming analysis of textual diffs. Particularly, detecting refactorings during code reviews is not a trivial task, since they are not explicitly represented in diffs. For example, a Move Function refactoring is represented by deleted (-) and added lines (+) of code which can be located in different and distant source code files. To tackle this problem, we introduce RAID, a refactoring-aware and intelligent diff tool. Besides proposing an architecture for RAID, we implemented a Chrome browser plug-in that supports our solution. Then, we conducted a field experiment with eight professional developers who used RAID for three months. We concluded that RAID can reduce the cognitive effort required for detecting and reviewing refactorings in textual diff. Besides documenting refactorings in diffs, RAID reduces the number of lines required for reviewing such operations. For example, the median number of lines to be reviewed decreases from 14.5 to 2 lines in the case of move refactorings and from 113 to 55 lines in the case of extractions.*

Resumo. *Revisão de código é uma importante prática no desenvolvimento de software que contribui para a melhoria de qualidade do código e transferência de conhecimento. No entanto, revisão de código leva tempo e exige uma análise detalhada e demorada de diffs textuais. Particularmente, detectar refatorações durante as revisões não é uma tarefa trivial, uma vez que as refatorações não são representadas em diffs. Por exemplo, ao mover uma função, o diff é representado por linhas adicionadas (+) e linhas removidas (-) em partes que podem estar localizados em diferentes e distantes arquivos do código fonte. Para solucionar este desafio, nós apresentamos RAID, uma ferramenta de diff inteligente que identifica atividades de refatoração e instrumenta os diffs textuais. Além de propor uma arquitetura para o RAID, implementamos um plug-in para o navegador Chrome que suporta nossa solução. Finalmente, avaliamos RAID em um experimento de campo por três meses, quando oito desenvolvedores profissionais usaram nossa ferramenta em quatro projetos Go. Concluímos que RAID reduz o esforço cognitivo necessário para detectar e revisar atividades de refatoração em diffs textuais. Particularmente, RAID também reduz o número de linhas necessárias para revisar tais operações. Por exemplo, o número médio de linhas a serem revisadas diminuiu de 14,5 para 2 linhas no caso de refatorações envolvendo movimentação e de 113 para 55 linhas no caso de extrações.*

1. Introduction

Code review is a widely used software engineering practice. Over the years, lightweight code review practices have emerged and gained popularity, in order to make the process more agile. Besides that, modern version control platforms—such as GitHub and GitLab—are contributing to popularize code reviews by means of pull/merge requests. However, code review takes time and may introduce delays in the release of new versions. The reason is that it is a manual process that requires expertise in the codebase and careful inspection of textual diffs.

In this dissertation, we introduce RAID, a refactoring-aware and intelligent diff tool. RAID is a tool for instrumenting textual diffs—particularly, the ones provided by GitHub—with information about refactorings. Typically, refactorings are represented in textual diffs as a sequence of removed lines in the left (-) and a sequence of added lines in the right. Therefore, code reviewers must infer by themselves whether this "difference" represents a refactoring operation, which requires an amount of cognitive effort.

Therefore, our key goal in this master thesis is to alleviate the cognitive effort associated with code reviews, by automatically detecting refactoring operations included in pull requests. Besides supporting refactoring detection, the proposed tool seamlessly instruments current diff tools with information about refactorings. As a result, reviewers can easily inspect the changes performed in the refactoring code after the operation. Particularly, RAID relies on RefDiff [Silva et al. 2021], which is a tool that detects refactorings for Java, C, JavaScript, and Go programming languages. RAID operates with a low runtime overhead, and it is fully integrated with state-of-the-practice continuous integration pipelines (GitHub Actions) and browsers (Google Chrome).

2. Methodology

To evaluate RAID, we relied on a field experiment. More specifically, we obtained permission to include the tool in the development workflow of a medium-sized technology company that develops software for musical products. In this way, two development teams (with 5 and 3 developers) used the tool during three months.

3. Conclusion

In this work, we presented RAID, a refactoring-aware tool that instruments GitHub diff with refactoring information. We concluded that RAID can reduce the cognitive effort required for reviewing refactorings when using textual diffs. In addition, our field experiment showed a reduction in the number of lines required for reviewing such operations. In the case of move refactorings, the number of lines decreases from 14.5 to 2 lines (median values); and from 113 to 55 lines in the case of extractions.

RAID is publicly available at GitHub.[1]

References

Silva, D., da Silva, J. P., Santos, G., Terra, R., and Valente, M. T. (2021). RefDiff 2.0: A multi-language refactoring detection tool. *IEEE Transactions on Software Engineering*, 1(1):1–17.

[1] `https://github.com/rodrigo-brito/refactoring-aware-diff`