Model-Based Reuse for Crosscutting Frameworks: Assessing Reuse and Maintainability Effort

Rafael S. Durelli*, Vinicius S. Durelli†, Valter Vieira de Camargo‡ Nicolas Anquetil*
*Departamento de Computação, Universidade Federal de São Carlos,
Caixa Postal 676 – 13.565-905, São Carlos – SP – Brazil
Email: {thiago_gottardi,valter}@dc.ufscar.br
†Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo,
Av. Trabalhador São Carlense, 400, São Carlos – SP – Brazil
Email: rsdurelli@icmc.usp.br
‡Universidad Politecnica de Valencia, Camino de Vera s/n, Valencia, Spain
Email: opastor@dsic.upv.es

***Abstract*—Today, almost every company runs systems that have been implemented a long time ago. These systems usually are still under adaptation and maintenance to address current needs. Very often, adapting legacy software systems to new requirements needs to make use of new technological advances. Furthermore, legacy software systems mainly consist of two kinds of artifacts: source code and databases. Usually, the maintenance of those artifacts is carried out through restructuring processes in isolated manners. Nevertheless, for a more effective maintenance of the whole system both should be analyzed and evolved jointly. Therefore, the lifespan of the legacy software systems are expected to improve. This paper proposes an approach to assist the modernization of both source code and database legacy systems. For this purpose, our technique analyses SQL queries embedded in the legacy source code in order to restructure and re-organize the legacy system by using design patterns. In order to validate our approach we have carried out an experiment throughout a real-life case study. The results were promising regarding the effort employed to modernize a legacy software system.**

I. Introduction

II. Software Restructuring

Perhaps the most common of all software engineering activities is the modifications of software. Unfortunately, software modification, i.e., software maintenance, often leaves behind software that is difficult to understand for those other than its author. In this context, software restructuring is a field that seeks to reverse these effects on software.

More specifically, software restructuring is the modification of software to make the software easier to understand and to change, or less susceptible to error when future changes are made (ref). In other words, it is the process of re-organizing the logical structure of existing software system to improve specific attributes [**?**]. Some examples of software restructuring are improving coding style, editing documentation, transforming program components (moving class, creating class, etc). The central idea of restructuring is the action of transformation. According to [**?**] a transformation can be defined formally as a function that receives a program, P, as input and produces a new program, P'. Thus, P' is said to be functionally equivalent to P $\Leftrightarrow$ P' exhibits identical behavior to P for all defined inputs of P. Finally, T is called a meaning preserving transformation if P' $\equiv$ P.

According to Griswold's experiments programmers tends to not only commit syntactic errors and behave inconsistently, they usually ignore the global impact of the changes they make

Manually restructuring software may have undesirable, and often unforeseen results that can affect the be- haviour of a system. Griswoldâs experiments found that programmers not only commit syntactic errors and behave inconsistently, but they also ignore the global impact of the changes they make [Griswold 1991]. Fur- thermore, manual techniques demand the maintainer to guarantee the preservation of the systemâs behaviour.

III. Model-Driven Restructuring

According to Griswold's experiments programmers tends to not only commit syntactic errors an behave inconsistently, they usually ignore the global impact of the changes they make during the restructuring process. Moreover, he also argues that manual restructuring is an error-prone and expensive activity [**?**]. Therefore, software engineers have applied Model-Driven Development (MDD) technologies to software restructuring to deal with those limitations and to automatize the software restructuring. MDD consists of the combination of generative programming, domain-specific languages and model transformations. It also aims to reduce the semantic gap between the program domain and the implementation, using high-level models that shield software developers from complexities of the underlying implementation platform [**?**].

IV. Related Work

V. Conclusions

Acknowledgments

References

[1] B.-K. Kang and J. M. Bieman, "A quantitative framework for software restructuring," *Journal of Software Maintenance: Research and Practice*, vol. 11, no. 4, pp. 245–284, 1999.

[2] J. Eloff, "Software restructuring: implementing a code abstraction transformation," in *Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, 2002, pp. 83–92.
[3] W. G. Griswold, "Program restructuring as an aid to software maintenance," Ph.D. dissertation, Ph.D. Dissertation, 1991.
[4] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*, ser. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37–54. [Online]. Available: http://dx.doi.org/10.1109/FOSE.2007.14