

[PGWF98] E. Pontelli et al. Natural Language Multiprocessing: a case study. In *National Conf. on Artificial Intelligence. AAAI, 1998.*

## A Model and a Tool for Change Propagation in Software

Václav Rajlich  
Wayne State University  
Detroit, MI 48202  
(rajlich@cs.wayne.edu)

The goal of this research is to make software changes easier, safer, and less expensive. Please note that software changeability is one of the essential properties of software and involves all software technologies. It is the core of software evolution [7].

One of the proposed solutions is to anticipate changes and structure the software in such a way that the changes will be localized inside software components. However, more recent case studies reported that only about 70% of the requirements were predicted in advance and the remaining requirements were discovered during development. Massive changes triggered by company mergers, introduction of Euro, etc., could not be predicted even a few years ago. Therefore it is likely that all software will be exposed to many unanticipated changes during its lifetime, and the support for unanticipated changes is an important research goal.

The process of change is divided into the following phases:

- Change request
- Change design, including program comprehension, feature location, and change impact analysis
- Change implementation, including restructuring for change and change propagation
- Change verification
- Record knowledge gained during the change

Under this NSF program, several phases have been addressed, see the rest of this report and also [5]. Models and tools of change propagation The change starts when the programmer changes a component of the software. After the change, the component may no longer fit with the rest of the software, because it no longer properly interacts with the other components. In order to reintroduce the consistency into the software, the programmer must keep track of inconsistencies and the locations of the secondary changes. The secondary changes, however, may introduce new inconsistencies, etc. The process of change propagation continues until all inconsistencies are removed. A formal model of change propagation was developed and published in [5]. It is based on static analysis of the program that produces evolving interaction graphs (eigs). It deals with scenarios and strategies used for change propagation. Examples are strict or random, and final or nonfinal strategies. A tool "Ripples" that supports change propagation was also implemented.

## Restructuring for change

The purpose of restructuring is to bring together parts of program affected by the change. It is well known that the delocalization of the change, i.e. the number of components that need to be visited during the change, increases the risk and difficulty of the change.

We developed several tools and scenarios that allow the programmer to restructure the code, without changing its functionality. In [1], we deal with misplaced code, i.e. code that appears in wrong classes. In [2], we deal with unnecessary duplication of the code, i.e. code clones. In [3], we deal with encapsulation of imperative code into classes. All three papers report case studies that validate the approach.

The tools that we implemented expulse the code from classes, insert the code into classes, and do several additional restructuring operations. The restructuring scenarios blend tool actions with programmer interventions.

## Recording program comprehension in www

The program comprehension is a prerequisite for program changes and it is a valuable commodity, as more than one half of the software maintenance and evolution work is spent in comprehension. Very often the comprehension is not recorded and resides entirely in the programming team. Since a small project team cannot afford redundancy, each part is comprehended ("owned") by one specific programmer. In that situation, an assignment of personnel to tasks becomes a problem. A resignation of a key programmer can have serious consequences because the comprehension – half of the work he/she has done – leaves also.

In order to address this issue, we developed Partitioned Annotations of Software (PAS) i.e. hypertext annotations based on world wide web. PAS are a universal programmer notebook that is used to record program comprehension and were inspired by theories of program comprehension. They can be browsed by a standard web browser.

We also developed an incremental and opportunistic redocumentation process that records comprehension gained during the changes. Description of PAS technology and an industrial case study is in [4].

## References

- [1] R. Fanta, V. Rajlich, Reengineering an Object Oriented Code, In *Proc. IEEE Int. Conf. On Software Maintenance, 1998*, 238-246.
- [2] R. Fanta, V. Rajlich, Removing Clones from the Code, *Journal of Software Maintenance, 1999*, 223-243.
- [3] R. Fanta, V. Rajlich, Restructuring legacy C code into C++, In *Proc. IEEE Int. Conf. On Software Maintenance, 1999*, 77-85.
- [4] V. Rajlich, Srikant V., Using Web for Software Annotations, *International Journal of Software Engineering and Knowledge Engineering, Vol. 9 (1999)*, 55-72.
- [5] V. Rajlich, Software Change and Evolution, In J. Pavelka, G. Tel, M. Bartosek, editors, *SOFSEM'99, Lecture Notes in Computer Science LNCS 1725, Springer Verlag, 1999*, 186-199.
- [6] V. Rajlich, Modeling Software Evolution by Evolving Interoperation Graphs, to be published in *Annals of Software Engineering, Vol. 9, 2000*, also <http://www.cs.wayne.edu/~vtr/eig.pdf>
- [7] K. Bennett, V. Rajlich, A new perspective on software evolution: The staged model, submitted.