Formal Foundations of Software Evolution: Workshop Report

Tom Mens*

Programming Technology Lab
Department of Computer Science
Vrije Universiteit Brussel, Belgium

tom.mens@vub.ac.be

Michel Wermelinger

Departamento de Informática
Universidade Nova de Lisboa
2825-114 Caparica, Portugal

mw@di.fct.unl.pt

Abstract

This paper summarises the results of the discussions held during the workshop on Formal Foundations of Software Evolution in Lisbon on March 13, 2001. These results can be used as guidelines when dealing with software evolution in general, and when providing formal tool support for it in particular.

Keywords: software evolution, software engineering, formal foundations, software change, software maintenance

Introduction

The workshop on *Formal Foundations of Software Evolution* was co-located with the *5th European Conference on Software Maintenance and Reengineering* (CSMR 2001), which took place at the Centro de Congressos do Instituto Superior Técnico in Lisbon, Portugal, March 14 to 16, 2001. The workshop was organised in the context of the *Scientific Research Network on Foundations of Software Evolution*. This is a research consortium coordinated by the Programming Technology Lab of the Vrije Universiteit Brussel (Belgium), and it involves 9 research institutes from universities in 5 different European countries (Belgium, Germany, Austria, Switzerland, and Portugal). The consortium is financed by the Fund for Scientific Research - Flanders (Belgium).

One full day was allocated for the workshop (March 13, 2001). There were 14 participants, that all contributed with a position paper which was reviewed and revised before the workshop. Next to the submissions of the research consortium partners, there were also participants from research institutes in Spain, United Kingdom, Finland, and Japan. In preparation to the workshop, participants were requested to read all other submissions, and asked to prepare a clear position statement and questions that were likely to stimulate discussion.

The goal of the workshop was to get more insight into how formal techniques can alleviate software evolution problems, and how they can lead to tools for the evolution of large and complex software systems that are more robust and more widely applicable without sacrificing efficiency. Preferably, the evolution-support tools should not be restricted to a particular phase of software evolution [BR00], but should be generally applicable throughout the entire application lifetime. The tools should also provide support for different aspects of software engineering, such as forward engineering, reverse engineering, re-engineering, and team engineering.

In order to stimulate discussions, three general important questions were posed to the participants at the beginning of the workshop:

- Which aspects of software evolution need to be automated by tools?
- Where and how can formalisms help us to achieve tool support?
- How can we build formally-founded tools that are as general and flexible as possible? Note that the generality and flexibility of a tool involves many different aspects:
  - independence of the programming language for which support should be provided;
  - customisability by the user of the tool;
  - applicability in or across different stages of software evolution;
  - interoperability with other tools;
  - scalability to large and complex software systems with multiple developers;
  - usable for static (design-time) as well as dynamic (run-time) evolution;
  - applicable to forward, reverse, and re-engineering;
  - usable before, during, and after evolution;
  - usable for facilitating, supporting, as well as analysing evolution;
  - to deal with the *what and why* as well as the *how* of software evolution

Workshop presentations

The papers and their authors were as follows, with the names of the actual participants in the workshop underlined:

- Wolfram Kahl (Univ. Bundeswehr München, Germany): Software Evolution via Hierarchical Hypergraphs with Flexible Coverage

*Tom Mens is a postdoctoral fellow of the Fund for Scientific Research - Flanders (Belgium).

- Lina García-Cabrera (Univ. Jaén, Spain), Maria José Rodríguez-Fórtiz, José Parets-Llorca (Univ. Granada, Spain): Formal Foundations for the Evolution of Hypermedia Systems

- Claudia Pons, Gabriel Baum (Univ. Nacional de La Plata): Software Development Contracts

- Meir M. Lehman, Juan F. Ramil, Goel Kahen (Imperial College, UK): Thoughts on the Role of Formalisms in Studying Software Evolution

- Timo Aaltonen (Tampere Univ. of Technology, Finland), Tommi Mikkonen (Nokia, Finland): Software Evolution Based on Formalized Abstraction Hierarchy

- Luís Andrade, João Gouveia, Georgios Koutsoukos (Oblog Software, Portugal), José Luiz Fiadeiro (Univ. Lisboa, Portugal): Coordination Contracts, Evolution and Tools

- Reiko Heckel, Gregor Engels (Univ. Paderborn, Germany): Graph Transformation as Meta Language for Dynamic Modeling and Model Evolution

- Jianjun Zhao (Fukuoka Institute of Technology, Japan): Change Impact Analysis for Architectural Evolution

- Michele Lanza, Stéphane Ducasse, Lukas Steiger (Univ. Bern, Switzerland): Understanding Software Evolution Using a Flexible Query Engine

- Michel Wermelinger (Univ. Nova de Lisboa, Portugal), Antónia Lopes, José Luiz Fiadeiro (Univ. Lisboa, Portugal): A Graph Transformation Approach to Architectural Run-Time Reconfiguration

- Tom Mens (Vrije Univ. Brussel, Belgium): Transformational Software Evolution by Assertions

- Jamal Said, Eric Steegmans (K.U. Leuven, Belgium): Transformation of Binary relations into Associations and Nested Classes

The papers were collected in a technical report [MW01], which is available from the workshop's web site at http://prog.vub.ac.be/FFSE.

The actual workshop was organised as follows. In the morning there were three long presentations of 25 minutes, each followed by 15 minutes of discussion. The presented topics were chosen because they offered different or novel perspectives on the workshop topic, and because they had a high potential for generating issues that would stimulate the discussions.

- The first presentation, by Wolfram Kahl, motivated the use of hierarchical hypergraphs as a unifying framework that allows one to design a coherent set of software evolution tools based on a common underlying representation, yet retaining the possibility for each tool to add functionality to deal with specific aspects of software evolution.

- In the second presentation, Maria José Rodríguez-Fórtiz proposed the use of a meta-level to support evolution in hypermedia systems. The model supports different formalisms which allow to specify the evolving actions and the propagation of the changes in order to maintain the integrity of the systems. For example, she proposed a combination of graph theory and temporal logic as basic formalisms. Furthermore, a clear separation between the memorisation system (which contains the actual hypermedia information) and the navigation system was made.

- Juan Ramil focused on the *what and why* instead of the *how* of software evolution. He claimed that both complementary views are important and worthwhile being investigated, and that they can both benefit from the use of formalisms. He also explained that the understanding (i.e., the *why and what*) of a phenomenon can be of great help in seeking to master and improve it (i.e., the *how*). As a formalism to address the why and what he proposed a system dynamics model that makes use of differential equations.

The afternoon was devoted to 8 short presentations (max. 10 minutes), where each participant presented his position statement which was then discussed in group and compared with the opinions of the other participants.

- Timo Aaltonen proposed to formalise the notion of *abstraction hierarchies* in a software system. He advocated the use of abstraction levels exceeding those provided by the implementation language constructs in order to cope with software complexity and to anticipate likely evolutions of the software system. Additionally abstraction hierarchies help in determining whether a requested change will have a minor or a major impact, depending on the level of abstraction where the change occurs.

- Georgios Koutsoukos presented work on *coordination contracts*, which are first-class citizens that provide an extra level of abstraction on top of object-oriented programming constructs in order to separate the coordination behaviour between classes from the actual computation that is implemented inside the classes. In this way, business rules, which are typically very volatile, can be specified and evolved separately from the core domain concepts.

- Reiko Heckel proposed to use a formal metamodelling framework based on graph rewriting to address software evolution problems. Such an approach can be used not only to address static (or design-time) evolution, but also to cope with dynamic (or run-time) evolution.

- Jianjun Zhao proposed to apply change impact analysis techniques, in particular *slicing techniques*, to software architectures rather than the implementation code. The intention is to visualise at an early stage what are the high level effects of change on the system.

- Michele Lanza discussed how a flexible query engine can be used to analyse software evolution after it has occurred. This

can be considered as a lightweight formal approach with a direct practical impact. The approach has been used in the context of reverse engineering, and is applicable to large-size software systems.

- Michel Wermelinger proposed the use of graph transformations and a program design language with explicit state to formalise run-time architectural reconfiguration. Run-time changes are often necessary for safety or economical reasons, since some systems cannot be shut off to be modified. Moreover, he argued that category theory might provide a framework to relate heterogeneous formalisms, which are needed when tackling different aspects of evolution.

- Tom Mens proposed the use of graph rewriting to address the problems of software merging, software upgrading and refactoring in a domain-independent way (i.e., independent from the considered phase in the software life-cycle). He also emphasised that scalability is an important prerequisite for tools to be applicable in large-scale industrial projects, so formal techniques need to address this issue explicitly.

- Jamal Said promoted the use of automatic transformations from analysis to design, because this makes it possible to maintain traceability when the software evolves. He also proposed to select analysis-to-design transformations based on the software quality factors that are deemed important by the developer. It remains an open question, however, how to formally attach quality factors to the transformations.

Workshop discussions and conclusions

Based on the various position statements, a number of claims were made during the discussions. These claims can be followed as guidelines when dealing with software evolution in general, and when trying to provide formal tool support for it in particular.

- *Support for evolution can be eased by raising the level of abstraction.* This claim was explicitly made by Aaltonen and Mikkonen with their formal notion of *abstraction hierarchy*. Koutsokos and Gouveia agreed with this view since *coordination contracts* provide a level of abstraction on top of the normal object-oriented programming constructs. Zhao's approach also raises the level at which the change impact analysis is performed. The goal is not only to get a better conceptual grip on the problem, but also to focus on levels where changes can have greater impact on the overall system.

- *Separation of concerns can help with software evolution.* García-Cabrera et al. achieve this by making a clear separation between a *memorisation* system and a *navigation* system for hypermedia evolution. Aaltonen and Mikkonen make a separation between *abstraction* and *implementation*. The coordination contracts by Andrade et al. provide a separation between *coordination* and *computation*. Finally, Heckel advocated the importance of separating *concrete syntax* and *abstract syntax*. Although these views are very diverse, they all have in common that they separate different concerns in order to facilitate software evolution.

- *Different parts of the software evolve at different rates, so it might be worthwhile to focus on those parts that have the highest change rate.* This claim was made most explicitly by Aaltonen and Mikkonen, since *abstraction hierarchies* allow us to assess whether a requested change has a minor or a major impact, by determining the level of abstraction where the change occurs. The *coordination contracts* approach claims that the coordination aspects, which represent business rules, are much more subject to evolution than the computation aspects, which are represented by ordinary object-oriented language constructs. Finally, García-Cabrera et al. agreed that for hypermedia systems the navigation system is likely to evolve more rapidly than than the memorisation system.

- *There is a need to consider not only "the how" but also "the what and why" aspects of software evolution.* More specifically, Ramil claimed that the understanding of the software evolution process (i.e., the *why and what*) can be of great help in seeking to master and improve the technical aspects of software evolution (i.e., the *how*). This is exemplified in the use of system dynamics simulation models to examine, for example, the performance of an organisation in charge of evolving a software product under different policies. The model presented suggested that complexity control is an important activity to ensure that the evolution of a software product is sustainable.

- *There is a need for more empirical and experimental research in software evolution* (see for example [KS99]). This claim was supported by Ramil, but also by Lanza, who proposed to use metrics for analysing software evolution, and by Said, who addressed non functional factors.

- *When providing formal tool support, one should always keep in mind the scalability, efficiency and usability aspects of the tools.* Note that there is no real solution to this problem. The more sophisticated the formalism becomes, the more powerful it becomes, but the more difficult it becomes to develop efficient tools on top of it.

- *Graph rewriting is a promising formalism for coping with many (but not all) aspects of software evolution.* This conclusion was obvious from the large number of submissions that emphasised the use of graph rewriting: Kahl, Heckel et al., Wermelinger et al., Mens. Based on this common interest, a more specific workshop devoted to the use of transformation approaches to software evolution has been proposed to the First International Conference on Graph Transformation, to take place near Barcelona, Spain, in Fall 2002.

- *Formalisms can help with providing domain-independent support for software evolution.* For example, the same tool could be used for different programming languages, or for different phases in the software life-cycle. In order to achieve this domain independence, a *metamodelling approach* is needed. The use of metamodels was advocated by various

participants: García-Cabrera and Rodríguez-Fórtiz, Heckel, Lanza, and Mens.

- *There is a need to provide formal support for co-evolution.* In a wider sense, co-evolution refers to the need to evolve the software as the various domains (application, development, usage) involved also evolve. One example of these is when the business process supported by the software is subject to rapid evolution. In a restricted sense, co-evolution refers to the situation where different—possibly partial—representations of the software (such as design and implementation) need to be kept consistent [DDMW00]. If one of these representations evolves, the other ones need to co-evolve. It is even possible that different representations of the software evolve in parallel, which makes it even more difficult to maintain consistency. The need for addressing co-evolution at all levels became apparent during many of the presentations.

Acknowledgements

We thank Juan Ramil for his many comments and suggestions on a draft of this report.

References

[BR00] Keith Bennett and Vaclav Rajlich. Software maintenance and evolution: A roadmap. In *The Future of Software Engineering*, pages 75–87. ACM Press, 2000.

[DDMW00] Theo D'Hondt, Kris De Volder, Kim Mens, and Roel Wuyts. Co-evolution of object-oriented design and implementation. In *Proc. Int'l Symp. Software Architectures and Component Technology: The State of the Art in Research and Practice*, Enschede, The Netherlands, January 2000. Kluwer Academic Publishers.

[KS99] C. Kemerer and S. Slaughter. An empirical approach to studying software evolution. *IEEE Trans. Software Engineering*, 25(4):493–509, July/August 1999.

[MW01] Tom Mens and Michel Wermelinger. Proc. of the Workshop on Formal Foundations of Software Evolution. Technical Report UNL-DI-1-2001, Departamento de Informática, Universidade Nova de Lisboa, March 2001.