Issue Overview

Separation of concerns for software evolution

This is a preprint of an article published in
JOURNAL OF SOFTWARE
MAINTENANCE AND EVOLUTION:
RESEARCH AND PRACTICE 2002,
Vol.14, pp. 311–315
URL: http://www.interscience.wiley.com/

Tom Mens[1,*,†] and Michel Wermelinger [2,‡]

[1] *Dept. Computer Science, Vrije Universiteit Brussel, 1050 Brussels, Belgium*
[2] *Departamento de Informática, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal*

SUMMARY

This special issue reports on approaches that apply the idea of separation of concerns to software evolution. In this context, separation of concerns allows us to separate parts of the software that exhibit different rates of change or different types of change. This makes it possible to provide better evolution support for those parts that have a higher change rate, or to provide different evolution techniques for different views on the software. Another common way to achieve separation of concerns is by raising the level of abstraction to the level of software architectures, business rules and metamodels. This makes software evolution more manageable. The above ideas emerged as important conclusions of the workshop on *Formal Foundations of Software Evolution*, which was co-located with the Conference on Software Maintenance and Re-engineering in Lisbon in March 2001. Of the 12 original position papers, 5 have been selected for revision and inclusion in this special issue of the *Journal of Software Maintenance and Evolution*.

KEY WORDS: software engineering, formal foundations, abstraction, rates of change

SELECTED PAPERS

This special issue contains five substantially revised and extended papers that have been selected from the position papers of the workshop on *Formal Foundations of Software Evolution*

*Correspondence to: Dr. Tom Mens, Programming Technology Lab, Dept. Computer Science, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel, Belgium.
†E-mail: tom.mens@vub.ac.be
‡E-mail: mw@di.fct.unl.pt
Contract/grant sponsor: Tom Mens is a postdoctoral fellow of the FWO (Fund for Scientific Research - Flanders, Belgium). Michel Wermelinger was supported by ATX Software SA and by Fundação para a Ciência e Tecnologia and FEDER through project POSI/32717/00 (*Formal Approach to Software Architecture*). Both guest editors participate in an FWO research network on *Foundations of Software Evolution* [4], and in *RELEASE*, an European Science Foundation research network on software evolution [3].

[9]. This workshop was co-located with the *5th European Conference on Software Maintenance and Re-engineering*, and took place at the Centro de Congressos do Instituto Superior Técnico in Lisbon, Portugal, on March 13th, 2001. Being an official activity of the research network on *Foundations of Software Evolution* [4], 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. In total, 12 position papers were accepted for the workshop, which have been collected in a technical report [8]. All but one position papers were presented during the workshop, subdivided into 3 long and 8 short presentations.

The selected paper "Evolving Hypermedia Systems: a Layered Software Architecture" by García-Cabrera, Rodríguez-Fórtiz, and Parets-Llorca presents an architecture to facilitate evolution of hypermedia systems. The architecture is composed of three subsystems (conceptual, presentation and navigation) and two abstraction levels (system level and meta level).

The paper "Relating Functional Requirements and Architecture: Separation and Consistency of Concerns" by Heckel and Engels proposes to use a formal metamodelling framework based on graph rewriting to address evolution and consistency problems among functional and architectural sub-models of the same system.

The paper "Separating Computation, Coordination and Configuration" by Andrade, Fiadeiro, Gouveia, and Koutsoukos, presents a 3-layered architectural approach to run-time software evolution based on the strict separation between computation, coordination and configuration. They define a modelling primitive, called *coordination contract*, to encapsulate the interaction between components in a way that is transparent to the components. In this way, business rules, which are typically very volatile, can be specified and evolved separately from the core domain concepts.

The paper "Change Impact Analysis to Support Architectural Evolution" by Zhao, Yang, Xiang, and Xu, applies change impact analysis techniques, in particular *slicing and chopping 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. The authors present an architectural slicing and chopping technique and apply it to systems described with the WRIGHT architectural description language.

The paper "Behavioural Modelling of Long lived Evolution Processes - Some Issues and an Example" by Lehman, Kahen and Ramil, explains how 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 suggests that complexity control is an important activity to ensure that the evolution of a software product is sustainable.

SEPARATION OF CONCERNS

All workshop conclusions are summarised in the official workshop report [9]. The main conclusion of the workshop was that *separation of concerns is needed to manage the complexity of software evolution.* This result emerged from nearly all position papers in one way or another. Although the views taken in the various papers were very diverse, they all had in common that they separate different concerns in order to facilitate software evolution.

Within the scope of software development, the term *separation of concerns* is typically used to cope with the complexity of a program by separating its fundamental computational algorithm (i.e., the basic functionality) from specific computing requirements (such as concurrency, distribution, real-time constraints, persistence, error handling) [5]. These so-called *concerns* typically cross-cut the basic algorithm, so separating them makes the software more manageable. This gave rise to an entirely new research area and software development paradigm, which is commonly referred to as *aspect-oriented software development* [1].

In the context of this special issue, however, the notion of separation of concerns should be seen from a much broader perspective. When investigating the evolution of software, it is often observed that *different views on the software* (e.g., architecture versus implementation) *may imply different types of evolution* and, consequently, require different techniques to handle them. It is also observed that *different parts of the software evolve at different rates.* Hence it is important to focus on those parts that have the highest change rate since providing automated support for the evolution of these parts will yield the highest benefit. A similar conclusion was made during the *4th ECOOP Workshop on Object-Oriented Architectural Evolution* [7], co-located with the *European Conference on Object-Oriented Programming.* To guarantee that a software architecture should be robust towards evolution and change as little as possible, the following definition was proposed that emphasises this requirement: *A software architecture is a collection of categories of elements that share the same likelihood of change. Each category contains software elements that exhibit shared stability characteristics.* Additionally, *a software architecture always contains a core layer that represents the hardest layer of change. It identifies those features that cannot be changed without rebuilding the entire software system.*

To summarise, in the context of software evolution, a *concern* may be *any criterion that allows us to separate parts of the software that exhibit different rates of change or that have a different impact on evolution.* In the contributions to this special issue, this separation of concerns can take on a variety of forms.

- With their *coordination contracts* approach, Andrade et al. claim that the *coordination* part, which represents business rules, evolves at a higher rate than the *computation* part. This is because the business process supported by software is typically subject to rapid evolution to cope with highly volatile business requirements.
- García-Cabrera et al. investigate hypermedia evolution and propose a clear separation between a *conceptual* subsystem, a *presentation* subsystem and a *navigation* subsystem. They agree that for hypermedia systems the conceptual subsystem is the most stable and the navigation system is likely to evolve the most frequent.
- Zhao et al. realise that the techniques and algorithms proposed to support change impact analysis at source code level need to be revised significantly in order to be applicable at

an architectural level as well, and suggest a novel slicing and chopping algorithm for this purpose.
- In order to master and improve the technical aspects (i.e., the *how*) of software evolution, one needs to investigate and understand the software evolution process (i.e., the *why and what*). This separation of concerns between the *how* and the *why and what* of software evolution is suggested by Ramil and Lehman. They show that the techniques required to understand the *why and what* may be quite different from techniques to deal with the *how* of software evolution.

RAISING THE LEVEL OF ABSTRACTION

Another common denominator in all five papers of this special issue is that they do not restrict themselves to mere evolution at implementation level (whether it be development-time evolution of source code, or run-time evolution of executable code). Instead, all papers propose to raise the level of abstraction at which to perform or understand software evolution. In agreement with what is stated in [2], this can be seen as another important way to achieve separation of concerns: *"An abstraction facilitates separation of concerns: The implementor of an abstraction can ignore the exact uses or instances of the abstraction, and the user of the abstraction can forget the details of the implementation of the abstraction, so long as the implementation fulfills its intention or specification."*

Below we give an overview of how the various approaches to software evolution proposed in this special issue raise the level of abstraction:

- Andrade et al. provide *coordination contracts* as an extra level of abstraction on top of the normal object-oriented programming constructs. By decoupling computation, configuration and coordination at this level, business rules can be expressed in a more natural and evolvable way.
- Zhao et al. raise the level at which the change impact analysis is performed (from implementation level to architectural level). 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.
- Heckel et al. and García-Cabrera et al. rely on *meta models* as an extra level of abstraction. This has the additional benefit that it 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.

ACKNOWLEDGEMENTS

We thank all workshop participants for their contributions, and the authors of selected papers for their efforts in making substantial revisions. We also express our gratitude to the many reviewers that took the time to provide detailed comments on the papers submitted to this special issue. Last, but

certainly not least, we thank Ned Chapin for inviting us to prepare this issue and supporting us during the process.

AUTHORS' BIOGRAPHIES

Tom Mens is a postdoctoral fellow of the Fund for Scientific Research - Flanders (Belgium) since October 2000. He is the main coordinator of two European research networks on software evolution [3, 4]. He obtained his PhD on "A Formal Foundation for Object-Oriented Software Evolution" at the Programming Technology Lab of the Vrije Universiteit Brussel. His main research interest lies in the use of formal techniques for improving support for software evolution, and he published several papers on this research topic. For the European Masters in Object-Oriented Software Engineering, jointly organised by the Vrije Universiteit Brussel (Belgium) and the Ecole des Mines de Nantes (France), he gives an advanced course on object-oriented software evolution.

Michel Wermelinger is an assistant professor at the Department of Computer Science of the New University of Lisbon, and a researcher at the Laboratory for Global Computing at the University of Lisbon. His main research interests are formal foundations of software architecture, software evolution, and coordination mechanisms. For his PhD on "Specification of Software Architecture Reconfiguration" he used rewriting, categorical, and graph techniques. He is the principal investigator of the Portuguese Science and Technology Foundation project "Formal Approach to Software Architecture". He was a member of the program committee of the International Conference on Software Engineering in 2002. He is a member of the Board of the European Association for Software Science and Technology and of the steering committees of the "Architectures for Mobility" project, funded by the European Commission, and of the RELEASE network [3].

REFERENCES

1. Elrad T, Filman R, Bader A (guest editors). Special section on Aspect-Oriented Programming. *Communications of the ACM* 2001; **44**(10):28-97.
2. Balzer R, Belz F, Dewar R, Fisher D, Gabriel RP, Guttag J, Hudak P, Wand M. Draft report on requirements for a common prototyping system. *ACM SIGPLAN Notices* 1989; **24**(3):93-165.
3. European Science Foundation scientific research network "REsearch Links to Explore and Advance Software Evolution (RELEASE)" webpage. http://labmol.di.fc.ul.pt/projects/release [19 July 2002]
4. FWO scientific research network "Foundations of software evolution" webpage. http://progwww.vub.ac.be/FFSE/network.html [19 July 2002]
5. Hursch WL, Lopes CV. *Separation of concerns.* Technical Report, College of Computer Science, Northeastern University, Boston, 1995.
6. Kemerer C, Slaughter S. An empirical approach to studying software evolution. *IEEE Trans. Software Engineering* 1999; **25**(4):493–509.
7. Mens T, Galal GH. 4th Workshop on object-oriented architectural evolution. In Frohner A (ed.), *ECOOP 2001 Workshop Reader, Lecture Notes in Computer Science*, **2323**. Springer-Verlag, 2002; 150–164.
8. Mens T, Wermelinger M. *Proc. of the workshop on formal foundations of software evolution.* Technical Report UNL-DI-1-2001, Departamento de Informática, Universidade Nova de Lisboa, Lisbon, 2001.
9. Mens T, Wermelinger M. Formal foundations of software evolution: workshop report. *ACM SIGSOFT Software Engineering Notes* 2001; **26**(4):27-30.