Electronic Communications of the EASST
Volume 30 (2010)

International Colloquium on Graph and Model Transformation On the occasion of the 65th birthday of Hartmut Ehrig
(GraMoT 2010)

Position Paper: Formal Methods in Agile Development

Michael Löwe

6 pages

Guest Editors: Claudia Ermel, Hartmut Ehrig, Fernando Orejas, Gabriele Taentzer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Position Paper: Formal Methods in Agile Development

Michael Löwe

FHDW Hannover

Abstract: Modern software development must be agile. It has to accept that software systems undergo a lot of changes due to changes in the application context (for example changing conditions on the markets and changes due to the jurisdiction) and base technology (e.g. integration of new frameworks or updates of the platform) in their life cycle. Thus, most of the activities in the development process are redesign steps. Even requirements are not stable. They change in time as the context of the system changes. There is no time for complex correctness proofs of the implementation with respect to the requirements. Automatic (regression) testing has proved to be sufficient for correct system behaviour. Therefore the agile developer does not learn and apply formal methods himself. In order to be agile, however, he relies on tools for automatic refactoring of the system or of certain parts of it. These tools are able to change the system structure without changing its behaviour. We argue in this paper that, in order to build such tools, further research in the area of formal system modelling and development is needed.[1]

Keywords: Agile Software Development, Software Refactoring, Graph Transformation

1 Introduction

There have been two major trends in software engineering for the last decade:

1. Raising the level of abstraction for software systems design (vertical development) and
2. Providing (more sophisticated) methods for agile development (horizontal development).

Notions like ”Model-driven Development”[BBG05], ”Service-Oriented Architectures”[KBS05], and ”Business Process Modelling”[Wes07] are connected to the first trend. The second trend is characterized by concepts like ”Software Refactoring”[Fow99], ”Test-First”[Bec02], ”Extreme Programming”[JAH00] or ”Dynamic Systems Development”[Sta97].

In the first area, formal methods, especially graph transformations, have provided precise semantics for model specifications and transformation concepts from abstract to concrete system descriptions including correctness notions for static as well as dynamic models (i. e. data structures and process models respectively). The level of abstraction that is provided to the standard programmer today by software development environments, modern design and programming languages and especially by program generation tools can hardly be increased. And the mapping

[1] The position paper is comprehensive. The references just provide some hints for further reading. They are not meant to be complete or comprehensive for the research area of software evolution and the application of graph transformations in this field.

of abstract levels (with unique semantics) to concrete machine oriented levels can be performed almost automatically and without any interference of the designer. Vertical development of the functional aspects of a system from a very abstract level to the concrete level of execution is a high-level compilation process nowadays. Research in formal methods has done a good job here. Educating the designers such that they can handle the abstractions is the challenge today.

In the second area, formal methods have not been applied that much, yet [MT04, MVDJ05]. At first glance, agility and formal preciseness do not go together well. We argue in this position paper against this first impression and show that there is great potential for graph transformation techniques in agile contexts.

2 Agile development

The agenda for agile development is provided by the ”Manifesto for Agile Software Devolpment” by Kent Beck et al.:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.

Agile development accepts that nothing is stable in software development. Requirements might change dramatically if, for example, customers use first rapid system prototypes. They learn what they want by using what the thought they have wanted. Due to these rapid changes, there is no time for an orderly formal development process that enforces correctness proofs of the implemented system wrt. the requirements. If there were such proofs, not only the software has to be refactorized frequently but also these proofs would have to be rewritten over and over again. Thus, formal methods do not seem to be applicable in agile contexts. And agile developers are not very likely to appreciate education in these techniques.

But there is a different level, where formal methods can support agile processes. The rapid redesign of software systems is not chaotic. It is a continuous process that introduces, changes or removes system structure, mostly without changing the external (functional) behaviour of the system. Hence, what is needed is a catalogue of evolution patterns that improve the system’s structure to a certain extent and preserve system semantics (incl. proof). The application of these patterns needs to be automated by a tool (like for example refactorizations in eclipse) and delivered to the agile developer.

Practical applicability, however, requires that we do not restrict ourselves to the level of static and dynamic models only. Since agile development aims at quick system development and early production with the system under development, we have to take into account that the models are populated. This means that there is (typically giga-bytes of) data typed in the static model[2]

[2] For agile database development see [Amb03, Amb06].

and (lots of) running processes typed in the dynamic model[3]. Thus, evolution patterns have to provide canonically induced and correct migrations on the instance level as well. Therefore, formal methods that support agile development shall provide

1. Suitable models for ”populated” systems (model and instance),
2. Formal concepts for model refactorisations and induced instance migrations,
3. Notions of correctness for such refactorisations/migrations, and
4. A catalogue of practically useful and correct patterns.

The existing body of concepts and results within the research area of ”Graph Transformation” seems to be a good starting point for this programme: (1) Graphs and graph-like structures provide a good formal model for almost all software structures (e.g. class models, activity diagrams, state diagrams, call graphs, data flow structures, control flow or petri nets) and (2) the rule-based transformation process is able to provide semantics to practical rule-based transformations like XSL-Transformation for XML-based languages or xtend/xpand in openarchitectureWare for model-driven development[Ope].

3 Formal Model for Systems: Model and Instance

Agile software development modifies *complete running systems*. It is not only the information, the operation, or the process model that is changed by refactorisations. This change also comprises at least the current system state. This state is made up by all the data that is accessible by the system (usually in a database) and the current point (or points in the case of multi-threading) of execution. Therefore, suitable formal models must be able to specify system models together with system states. A formal model for instance for object-oriented concepts must comprise the class model, the specification of the operations and methods, the currently existing object world, and the current execution context, i. e. the already sent but not yet executed messages and their execution order.

If we include explicit process models (for example specified in the Business Process Modelling Notation BPMN [Wes07]) into our framework, the state can get even more complex. Having the process model at hand, the current state not only comprises information about the current execution context but also the process history that has led to the current state. Additionally, the indeterministic future of the process (starting at the current point of execution) can be thought of as part of the current state.

The model and the state cannot be considered separately. The state is always determined by the model which is usually expressed by a typing relation between state items and elements in the model. For a formal framework of agile development this typing relation is central, since model changes must lead to minimal state restructurings that allow correct retypings.

In the context of graph transformation, a suitable model for the typing relation can be given by a morphism from the state graph into the model graph.

[3] For process evolution see for example [AB02].

4 Refactorisations and Induced Migrations

Agile development demands automatic refactorizations of whole systems (models and states). If state migrations have to be calculated or performed manually (or by time consuming batch jobs), development becomes slow and looses its agility. Since the state continuously changes in a running system, the only way to initiate general changes is to change the model (which is constant while the state is changing). Therefore state migrations shall (1) be uniquely induced by model changes and (2) must be executable without any interactions of the developer.

It depends on the type of system that is developed whether it can be switched off during migration. Real-time embedded systems in critical applications for example can never be switched off. And service orientation requires minimal down-time also for modern information systems. Thus, a framework for agile development must provide some means for *migration on demand*: The state is not changed completely, it is changed step by step as the execution wrt. the new model proceeds and requires retyped state structures. This mechanism requires (i) model versioning, (ii) coexistence of different models within the running system, and (iii) (partial) typings of the same state into different models.

In the context of graph transformation, model changes can be expressed by simple graph transformation rules and their application. The canonical extension of the model change to the existing state requires some kind of universal quantification (perform the model change for *all* instances), which is not a standard mechanism in many approaches to graph transformations.

5 Correctness of Migrations

A formal framework for agile development can provide proof methods by which tool designers can show that their migrations do change the system structure but not its observable behaviour. Such proofs are valuable since the tool user can rely on the correctness of the transformation without knowing the formal languages in which the proof was formulated. The basis for such proof methods is formal semantics for complete systems. (The semantics depends on the chosen notion of state!) Here well-known notions from for example algebraic specification (observable equivalence) or process algebra (bisimulation) can be reused. If a transformation cannot be proven generally correct for all system states but only for a certain class of states, appropriate tool support shall be provided that checks the required properties of the state.

Graph transformation techniques for proving invariants of the generated graph language can support the efforts towards such proof methods.

6 Catalog of and Tool Support for Correct Evolution Patterns

All the work that has been sketched in the previous sections has one aim, namely a catalog of (partially) correct evolution patterns and its implementation within a software development environment or some software generation tool and - if migration on demand is realized - the runtime environment of the execution language. This catalog shall - amongst others - comprise patterns for the

- Introduction of new structure

- Removal of unused structure
- Introduction and removal of abstractions (observer, composite, state, etc.)
- Introduction (and removal) of structural indirection (adapter, proxy, visitor, etc.)
- Introduction (and removal) of operational indirection (command, event, etc.)
- Introduction (and removal) of transaction support
- Introduction (and removal) of locking strategies
- Introduction (and removal) of versioning and historization
- Introduction and removal of parallelism
- Decomposition of process steps
- Merging of process steps
- Introduction (and removal) of process alternatives
- Introduction (and removal) of remote communication and distribution structure

The documentation of the patterns can be provided as some sort of graph transformation rules.

7 Conclusion

In this position paper, we have argued that formal system modelling and transformation can support agile software development. It provides urgently needed concepts and tools for the consistent and correct transformation of complete and running systems. While object-oriented modelling and programming has become a quasi-standard in the software community, the approaches, languages, and methods in the research area of graph transformation are still very different[4] In order to produce some remarkable effect on the application domain of agile development (and other application areas), some standardization towards *the graph transformation language, framework and development environment* is needed.

Bibliography

[AB02] W. M. P. van der Aalst, T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.* 270(1-2):125–203, 2002.

[AGG] The AGG 1.5.0 Development Environment - The User Manual. http://user.cs.tu-berlin.de/ gragra/agg/AGG-ShortManual/AGG-ShortManual.html.

[4] There are many different languages. Even within one approach, there are some variants. In the algebraic approach, for example, there are the double-pushout [EEPT06], the single-pushout [Löw93], and the sesqui-pushout approach [CHHK06]. There are different tools with strengths and weaknesses that cannot be combined with each other easily, for example [AGG] or [FUJ].

[Amb03] S. Ambler. *Agile Database Techniques*. Wiley, 2003.

[Amb06] S. Ambler. *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley, 2006.

[BBG05] S. Beydeda, M. Book, V. Gruhn (eds.). *Model-Driven Software Development*. Springer, 2005.

[Bec02] K. Beck. *Test Driven Development - By Example*. Addison-Wesley, 2002.

[CHHK06] A. Corradini, T. Heindel, F. Hermann, B. König. Sesqui-Pushout Rewriting. In Corradini et al. (eds.), *ICGT*. Lecture Notes in Computer Science 4178, pp. 30–45. Springer, 2006.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.

[Fow99] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[FUJ] Fujaba Tool Suite. http://www.fujaba.de/about-fujaba.html.

[JAH00] R. Jeffries, A. Anderson, C. Hendrickson. *Extreme Programming Installed*. Addison-Wesley, 2000.

[KBS05] D. Krafzig, K. Banke, D. Slama. *Enterprise SOA: Service Oriented Architecture Best Practices*. Prentice Hall, 2005.

[Löw93] M. Löwe. Algebraic Approach to Single-Pushout Graph Transformation. *Theor. Comput. Sci.* 109(1&2):181–224, 1993.

[MT04] T. Mens, T. Tourwé. A Survey of Software Refactoring. *IEEE Trans. Software Eng.* 30(2):126–139, 2004.

[MVDJ05] T. Mens, N. Van Eetvelde, S. Demeyer, D. Janssens. Formalizing refactorings with graph transformations. *Journal on Software Maintenance and Evolution: Research and Practice*, 2005.

[Ope] OpenArchitectureWare Group. openArchitectureWare User Guide Version 4.3.1. www.openarchitectureware.org.

[Sta97] J. Stapleton. *DSDM - Business Focused Development: The Method in Practice*. Addison-Wesley, 1997.

[Wes07] M. Wespe. *Business Process Management*. Springer, 2007.