Tightly Integrated Views: Overcoming the Gap between Software Engineering Concepts and Practice in Software Architecture – A Position Paper

Michael Goedicke, University of Duisburg Essen, Paluno – The Ruhr Institute for Software Technology, 45117 Essen, Germany, michael.goedicke@s3.uni-due.de

Abstract: It has to be stated that the software industry has not picked up software architecture concepts and methods, which have been developed during recent decades. While the key requirements for modular software structures are known since the 1970ties and various Architecture Description Languages (ADLs) have been developed, these concepts have not been used in designing programming languages and/or frameworks for large scale software developments. Software Industry, however, has accepted the approach to use frameworks and patterns to impose some structure on their software systems in order to keep workable development artefacts. This position paper proposes a research path, which uses the inclination of practitioners towards program code, frameworks and patterns on the one hand and abstract models on the other hand as a way to specify and document high level design decisions. This is accomplished by embedding abstract specifications into code. Such annotated and structured code serves as the basis for program development and architecting the code at an abstract level using the embedded specifications. This is regarded as a first step towards a more software architecture centric way of structuring code and entire software systems during design and run time.

Introduction

We see the software development process is structured into stages and not phases. This means, that the stages as depicted in fig. 1 interact and the sketch is not meant to indicate a temporal ordering of activities.

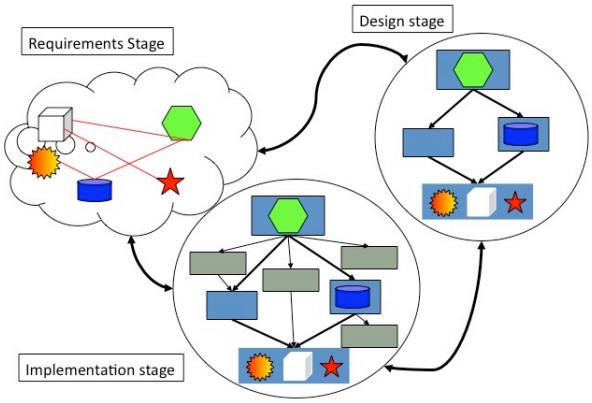

Fig. 1 Stages of Software Development

An important issue of contemporary software development processes can also be interpreted in terms of the picture sketched in fig. 1. Usually software engineering methods and techniques assume green field development. The stages of runtime and maintenance/evolution are regarded as a less interesting special case of a green field development. This however, neglects the important data regarding actual usage scenarios and problems detected during the so-called production phase of a software system. In addition, it has to be noted that the connection between abstract specification and design models of the system in question and the information gathered during the system's runtime is hardly made. This is due to the fact that the last minute

fixes of the system (before it goes into "production") are not reflected in those development artefacts, not to mention the development steps occurring during evolution after the first release. Model driven Software Development is in its current form limited by the lack of good round trip engineering concepts, which really work [0].

We have pointed out in [1] this kind of software aging which requires new attempts to cover the entire lifecycle including production till decommission of the software system. Also the new way of constructing software system using many pre-existing so-called services provides new challenges.

In summary, a thorough understanding of the relationships between the stages depicted in fig.1 provide the basis for new approaches to explain the nature of software development and infer related concepts, methods and tools.

Here we would like to focus on architecture and code in order to sketch how high level specifications can be tightly integrated into code. This provides an opportunity to study the relationships in more detail and give hints for new concepts and methods in this area of software development.

General Approach

The general approach is depicted in fig. 2. It shows for the various stages related to design and implementation that specifications (models) and code are intertwined in such a way that the abstract specifications are available during runtime. Thus they can be used to execute pieces of the specified software fragment and are instrumental during the evolution for the purpose of design recovery.

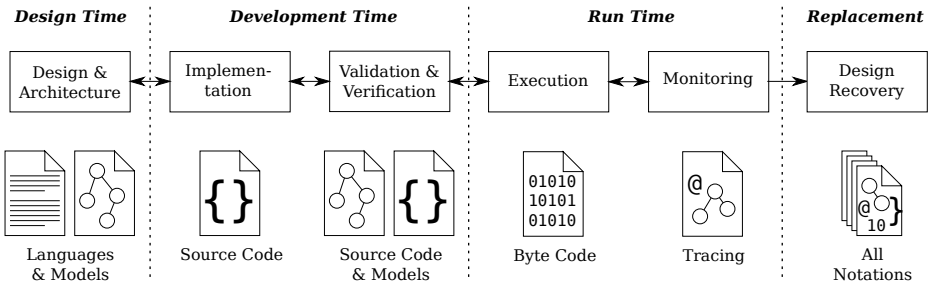

Fig. 2 Relationship between specifications and program code over the lifetime of a software system

The relationship between specifications and code [2] is formulated as annotations contained in the code i.e. code fragments are arranged in patterns derived from concepts and construct of the specification language / models used at the abstract description level.

This approach can also be viewed as an attempt to use abstract specification concepts in order to form specific patterns at the implementation level. Thus the specification statement contained in a piece of program code provides the frame / pattern for the code at the detailed level. The advantage is that the code can be checked wrt. the required properties of the used specification construct contained in the annotation. A usually small framework uses the annotation during runtime via reflection and executes the fragment encapsulated by the annotations.

Example

As a first experiment in this direction we have used state machines to express abstract behavioural models in Java Code [3]. Here Java classes serve as a representation of states, transition are specified using appropriately annotated methods. A number of tools have been developed, which support this approach at various levels of specification and implementation. A transformation tool extracts the state machine model, which can be used in the UPPAL- tool for further analysis. Design changes are reflected back at the implementation stage. A Java fragment specified and implemented using this approach can be monitored at runtime. This allows to observe the program behaviour not only in terms of the used programming language (Java) concepts like assignments but also in terms of states and transitions and related conditions enabling / disabling transitions.

Work in progress is to complement this by a component concept, which is also embeddable into program code in a similar way. First analysis results are in [4] and related experiments are available soon, which will be an extension to OSGi.

Next step: beyond current programming languages

Of course, the approach sketched above is aiming at the challenges of today's software development. I.e. we limit ourselves to contemporary programming languages and available specification concepts. As such, this approach serves as an immediate step to join both worlds: abstract specifications and implementation patterns. Of course this will not be the final answer.

The immediate goal beyond the current experimentation is to find proper component models including aspects like persistence, transactions, performance, and security, which address a whole range of software development issues. This is meant to address not only the enterprise level but also the embedded world with its own peculiar characteristics.

This will entail in our view the development of new programming language concepts, which allow the tight integration of architecture views. Related tools allow then the extraction of abstract views for design and analysis actions. This will also help to deepen our knowledge regarding the relationship between architecture and design stages.

Conclusion

We believe that the approach sketched above will lead to a deeper understanding how (architectural) specification and implementation are intertwined and how design time and run time can benefit from each other. The current approach is based on current technologies and will provide insights how current concepts can be developed further in order to increase the productivity of software engineers by related powerful tools.

Of course, the stage of requirements management needs to be integrated properly as well. Here it is necessary to address issues like constant change, ambiguity and diversity of different stakeholder views and/or the simple explicit lack of knowledge in additional views (c.f. [5]). Such views are certainly not embeddable as sketched above for the architecture and implementation stage. But a deeper understanding of the relationships and how requirements are mapped and traced to the architectural/implementation stage and reverse will provide more effective ways of development here as well.

References

[0] Martin Fowler, PlatformIndependentMalapropism (2003) http://www.martinfowler.com/bliki/PlatformIndependentMalapropism.html.

[1] Gregor Engels, Michael Goedicke, Ursula Goltz, Andreas Rausch, Ralf Reussner: Design for Future - Legacy-Probleme von morgen vermeidbar? in: Informatik-Spektrum: Volume 32, Issue 5 (2009), Page 393, 2009

[2] Michael Goedicke, Michael Striewe, Moritz Balz: Support for Evolution of Software Systems using Embedded Models, in Proc. Workshop "Design for Future - Langlebige Softwaresysteme", October 15th-16th, Karlsruhe, CEUR Workshop Proceedings Vol 537, 2009

[3] Moritz Balz, Michael Striewe, Michael Goedicke: Embedding Behavioral Models into Object-Oriented Source Code, in: Proceedings of "Software Engineering 2009", Kaiserslautern, Germany, LNI Vol P-143, 2009

[4] Marco Müller, Moritz Balz, Michael Goedicke: Representing Formal Component Models, in OSGi, in: Proceedings of "Software Engineering 2010", Paderborn, Germany, pages 45-56, LNI Vol P- 159, 2010

[5] Michael Goedicke, Thomas Herrmann: A Case for ViewPoints and Documents, in: Innovations for Requirement Analysis. From StakeholdersNeeds to Formal Designs 14th Monterey Workshop 2007, Monterey, CA, USA, September 10-13, 2007. Revised Selected Papers LNCS 5320, pp 62-84, Springer Verlag 2008