Improving Traceability in Model-Driven Development of Business Applications

Andreas Rummler[1], Birgit Grammel[1] and Christoph Pohl[2]

[1] SAP Research CEC Dresden,
Chemnitzer Str. 48, D-01187 Dresden, Germany
[2] SAP Research CEC Karlsruhe,
Vincenz-Priessnitz-Str. 1, D-76131 Karlsruhe, Germany
E-Mail: {andreas.rummler,birgit.grammel,christoph.pohl}@sap.com
WWW: http://www.sap.com/research

Abstract. This paper gives an overview of the support for traceability in an industrial context. The state of best practices are described and shortcomings are identified. Furthermore the AMPLE project is introduced, encompassing an overview on our approach towards traceability in the context of Model-Driven Development (MDD) and Software Product Lines (SPL) in conjuction with Aspect-Oriented Software Development (AOSD). AMPLE can be considered as a joint effort to tackle the shortcomings of current industrial practice. The paper is of positional nature and outlines our work currently in progress.

1 Introduction

During the development stages of complex software systems many artifacts are generated, either manually or in an automatic manner. The nature of these artifacts ranges from requirements expressed in text documents down to statements in source code. Understanding the mutual dependencies and the logical relationships among artifacts is a nontrivial task and becomes harder the more complex a system is. The problem is not only of academic interest - it has also been perceived in industry. According to an internal audit of customers of SAP, missing traceability during the whole development cycle is the top-rated weakness.[3] In addition missing traceability information was explicitly mentioned as weakness in 2005 in an external ISO certification audit.

In real world business applications traditional model-driven development approaches and software product line engineering techniques often do not reflect the decomposition of system features well enough. For instance, compliance checks of legal business regulations or late introduction of security properties often crosscut the architectural design of a system. To overcome these issues Aspect-Oriented Software Development (AOSD) technologies are gradually taking hold in industrial software development [1, 2].

[3] In 2004, 10 out of 11 customers criticized this.

AOSD techniques aim to modularize crosscutting concerns, that are scattered over the whole system into independent modules. These modules are called aspects. There is no clear mapping of aspects identified during the requirements stage to later development stages. This is due to the fact, that aspects have complex dependency relations, i.e. aspects are related to other artifacts within one or more phases. Secondly tracing of aspects is aggravated through their versatile nature, e.g. appearing or disappearing within a phase. This results in increased complexity for the task of providing traceability in systems utilizing AOSD techniques [3]. Model and code weaving techniques make decisions that a tracing mechanism needs to take care of and needs to keep track of.

While traceability in MDD seems to be relatively well understood, traceability in conjunction with AOSD techniques is a rather new field of interest. In MDD several approaches have been implemented and are used in practice. To mention one, traceability is supported in Query View Transformation (QVT), [4, 5], where instances of trace classes store the record of transformations. However, in these approaches, traceability starts in the design stage. For this reason tracing of artifacts created in earlier stages like requirements engineering to later development stages is still poorly understood.

The remainder of this paper is structured as follows. First, we will give an overview on the traceability support, which is currently implemented at SAP. In section 3, we then continue to introduce the AMPLE project and state the main objectives. In section 4, the focus is shifted to potential means and ways of improving traceability in the context of AMPLE. Last, but not least a few concluding remarks are given in the last section.

2 Tracebility Support inside SAP

The product development process inside SAP is backed up by an integrated process model called *Product Innovation Lifecycle* (in the following abbreviated as *PIL*). PIL subdivides the whole product lifecycle into several stages, reaching from the initial idea to deployment and maintenance. The individual stages are *Invent*, *Define*, *Develop*, *Deploy* and *Optimize* and are shown in figure 1. Although not explicitly shown in the mentioned figure, PIL is an interative process. Unlike development processes that lean back against the classical waterfall model, PIL contains iteration cycles that may be passed several times.

The *Invent*-stage is the initial stage. Here ideas that could be interesting to a specific market are collected, evaluated and categorized. Typical activities during this stage incorporate the definition of market and solution requirements and the analysis of how a potential product idea fits into the overall corporate and solution strategies. The requirements originating from this stage are transferred to development actions in the succeeding *Define*-stage and realized in concrete development projects in the third stage, the *Develop*-stage. This stage incorporates all typical activities of a development process: From architecture definition and high level planning over detailed design to implementation, integration and unit/acceptance test. The last two stages include the actual roll-out

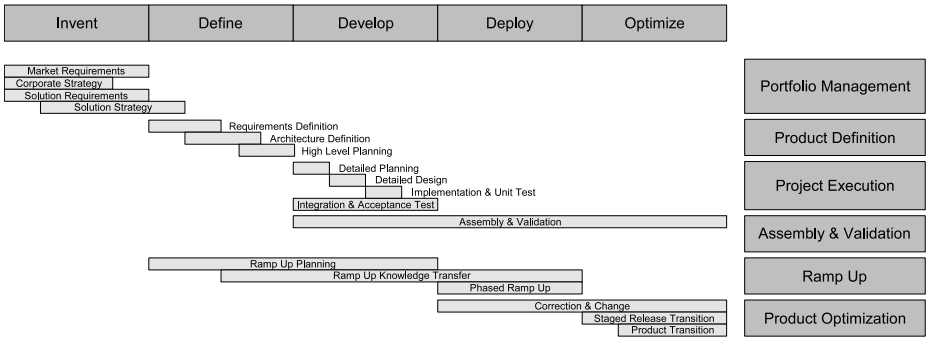

Fig. 1. Product Innovation Lifecycle Outline

to the customer and a continuous on site optimization, including corrections and late changes.

PIL is supported by several tools, which as such are only partially available as commercial products. There is tool support assigned to the planning stages; a user is aided in his tasks in product portfolio management and product definition. This covers the stages of *Invent* and *Define*. Here a coarse-grained view on a certain product or product family is facilitated. Market requirements can be grouped together into so-called portfolio cases and assigned to work packages.

More interesting in this context is the tool support for the next stage. This incorporates project planning, execution, accounting, resource and time management. The already defined work packages are assigned to newly created projects which can be linked to actual software requirements for the purpose of providing traceability. The idea behind this is to allow tracing of information from the definition of a product to its delivery; requirements are meant to be arranged in a way to enable their own management and the management of their interdependencies. Artifacts that are created to fulfill requirements are arranged in a continuous path. At certain points along this path deliverables are provided; the contained information inside is reused later on. The requirements themselves serve as anchors for traceable paths. To enable this, requirements must be structured into coherent pieces, which can be interlinked. Such formalized outline reduces the danger of misinterpretation, which, of course, cannot be fully eliminated.

The tool support provides templates for the creation of new requirement specification documents. These documents are structured into a header, several chapters and subchapters. The chapters contain requirements tables, that allow the definition and linking of single requirements, the definition of identifiers, types, trace-to statements and appropriate scope. These documents are finally linked to projects where the actual activities are performed to fulfill the requirements. On the back-end a relation engine allows the linking of requirements to test cases to ensure that a certain requirement is covered by one or more tests in the final product.

The result of each stage is a set of particular deliverables. Documenting the transition between single deliverables could be done in graphical way supported by dedicated tools. At the moment this is not implemented, instead standard office tools are used for these tasks. Examples are the manual creation of text or spreadsheet documents. Some steps in this process are performed in a model-based way, but are in fact still far away from a truly model-driven approach.

Looking at the full traceability picture of the entire development process, which is currently implemented, this incorporates the linking of market requirements to software requirements specifications documents which inturn are linked to their entailing software requirements. These software requirements can be linked to test cases. This is shown in figure 2. Further linking of requirements to design and/or development artefacts, respectively the linking of those artefacts to test cases is possible in general, but not sufficiently supported by tools. This is indicated in figure 2 by dashed lines and is subject to future work. In the current state several interesting questions that arise during the development process cannot be answered in an automatic and reasonable way:

- How can it be ensured, that the design which was produced by developers really covers all requirements?
- Are the market requirements, which were defined on a very high and abstract level, really implemented and how can this implementation be tested against these requirements?
- Have artifacts been developed in one or more stages, such that they cause a behaviour which is not covered by the requirements specification or even is unwanted?

These questions can be answered on a fine-grained level by experts involved in the development process, but not by people dealing with a coarse-grained view. Therefore tools are needed that are able to give reasonable answers to these questions or at least help people in finding those answers. The current tool support inside PIL is based on conventional development methodologies, which do not cover new approaches like AOSD, which may help easing the task of tracing artifacts. This is where the AMPLE project comes into play, which is described in more detail in the next section.

3 The AMPLE Project

AMPLE stands for Aspect-Oriented, Model-Driven Product Line Engineering[4]. The project is funded by the European Union and incorporates nine project partners, both from academic science as well as from industry, including SAP. The overall aim of the project is to develop a methodology for the design of Software Product Lines (SPLs) that offers improved modularisation of variations to support their traceability across the entire SPL lifecycle. To achieve this a novel combination of AOSD and MDD techniques is applied. The focus is on providing

[4] The website for the project can be found at http://www.ample-project.net

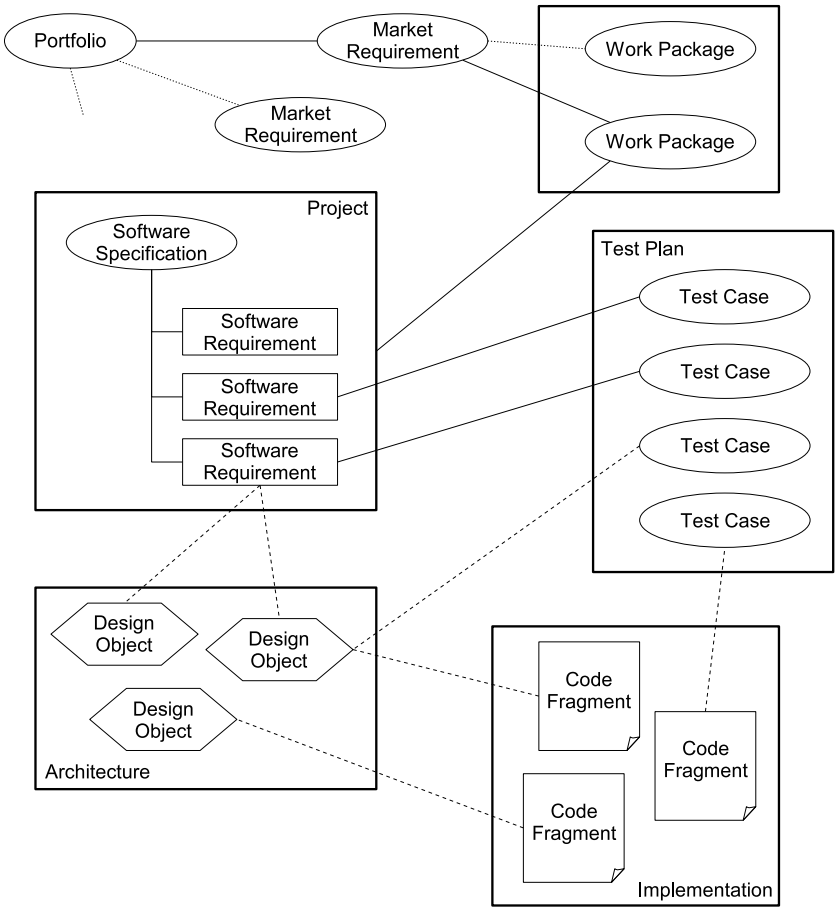

Fig. 2. Linking of Artifacts in the Development Process

a holistic treatment of variability by not only addressing variability at each stage in the SPL life cycle but also managing variations in associated artefacts such as requirements documentation, manuals and reports. Furthermore, AMPLE aims to bind the variation points in various development stages and dimensions into a coherent variability framework across the SPL engineering life cycle thus providing effective forward and backward traceability of variations and their impact. This makes it possible to develop resilient yet adaptable software product line architectures for exploitation in industrial SPL engineering processes.

The objectives of AMPLE cover all stages of the development process from the early stages of requirements analysis and engineering over design to implementation. Most interesting in this context is a separate work package with the goal to develop aspect-oriented models representing the interfaces between requirement analysis – design, design – implementation and implementation –

execution phases in order to provide an abstract view of the refinement of the software product line in order to maintain forward and backward traceability of the variations. Along the associated infrastructure and methodologies for software product line engineering for smooth transition to industrial software product line engineering processes should be developed.

4 Improving Traceability in AMPLE

Within the AMPLE project SAP is involved – to a large degree – in research on traceability. Questions originating from industry practice should be investigated further to come to practicable solutions. Besides the general goals in AMPLE, namely improving transition from model-based to model-driven development and the modularization of cross-cutting concerns into reuseable aspects, SAP is especially interested in synergies arising from the combination of SPL engineering approaches in existing SAP solutions with model-driven approaches and of course in the traceability of all modelled artifacts across the whole software lifecycle. Although traceability has been recognized as an important issue in SPL engineering, reliable and industrial-strength tool support is missing, from the commercial side as well as in form of academic prototypes.

Traceability refers to the linking of different artifacts on the same and on different abstraction levels and the ability to follow those links. Artifacts may include documents, stakeholders, modelled objects or code fragments. It can be easily noticed, that these vary in nature and structure. Artifacts may be divided into different kinds of subgroups: As an example, documents may be classified as requirements, within a glossary, use cases or source files. In addition, links between artifacts may also be of different kinds, the semantic meaning of links may include *refine*, *implement* or *substitute*.

A recent survey of general approaches on traceability done in the AMPLE project found several open problems and shortcomings in existing techniques:

- SPL engineering approaches are not capable of providing traceabilty between artifacts on different abstraction levels, i.e. between market requirements and software requirements [6], [7], [8].
- Although linking between software requirements and test cases is already applied in industry, the applied techniques are usually tailored to special cases and are not appliable within a general context.
- The tracing of information about product derivation is a complicated task. This information may include things like configuration data, build information or version numbers and is scattered over project files with proprietary data formats. In addition available tool support in general is poor.
- There is no integrated connection between traceability tooling and software configuration management systems.
- Change impact and coverage information is not available end-to-end for crosscutting concerns, subtle changes at both ends of the development stages may impact artifacts in the whole product line [9], [10].

– It is impossible to analyse generated artefacts for potentially unwanted side-effects.

These problems are to be addressed in the AMPLE project. There are approaches to Aspect-Oriented Requirements Engineering (AORE) [11, 12] that can help address some of these shortcomings. For instance relationships, dependencies and interactions among existing requirements can be identified at early stages of the development lifecycle. However, AORE approaches do not explicitly define mechanisms for mapping information gathered at the requirements level to later development phases. There is a need for defining mapping guidelines, rules, and heuristics for mapping of entities and trace information across the entire development lifecycle. Asset repositories are also required that may collect and maintain product line assets and the mapping rules, guidelines, and heuristics. In addition, there is a need for a traceability meta-model, that defines which trace information, i.e. assets, concerns, relationships, dependencies, behaviours, compositions and mappings, needs to be captured and managed.

The approach followed in the AMPLE project relies on the modularization of cross cutting concerns at model level. Starting already at the stage of requirements engineering will foster traceability. To be able to track dependencies between AO and non-AO artifacts along the development cycle, explicit aspect interfaces need to be defined. To rely on earlier work already made available in [13] and [14] seems to be promising. The intrusive nature of AO techniques is reduced in its intensity by defining aspect interfaces that form some kind of contract between the to-be-extended system and the extending aspects.

First experiences in combining aspect-oriented principles with model-driven software development yielded promising results. This work was based on openArchitectureWare [15, 16]. Work in progress will leverage on these experiments on aspect interfaces, extend this approach and incorporate support for tracing.

Ongoing work consists of the definition of a metamodel for variability in SPL including the support of AO concepts and appropriate tracing information. Based on this metamodel a tool chain is designed that supports the definition of SPL, product generation and full support for tracing relationships and dependencies among automatically generated or manually created artefacts. A suitable use case that is currently implemented consist of a complete example originating from SAPs core business. Here, the ideas and concepts elaborated in the AMPLE project are examined and evaluated from a very practical perspective.

5 Conclusion

Enabling traceability of artefacts throughout all stages of the development cycle of a software system has been recognized as a crucial task. But there is a large gap between needs and best practices in industry on one side and published solutions from academic research on the other. This article gives an introduction to work related to traceability in the AMPLE project, which intends to solve some of the problems related to this field of research. Inside AMPLE an approach is fostered where explicit extension points are defined that can be complemented

by appropriate aspects to moderate the highly intrusive nature of AO techniques. Following such an approach will remove the 'obliviousness' of aspect orientation up to a certain degree, but is to the authors opinion the only way to achieve greater acceptance of these techniques in an industrial context. A developer is now forced to take care of potential aspect-oriented extensions to the code he is creating. In addition this concept should not only be utilized in implementation but also raised to the levels of modelling or even requirements engineering. This will increase maintainability and along with that traceability in general, but especially vertical traceability among artefacts on the same abstraction level. Based on this approach a tool chain is developed that includes complete support of traceability from one end of the development process to the other.

6 Acknowledgements

This work has been partially funded by the European Union in project AMPLE, FP6 IST SO 2.5.5.

References

1. Kiczales, G., Irwin, J., Lamping, J., Loingtier, J.M., Lopes, C.V., Maeda, C., Mendhekar, A.: Aspect-oriented programming. In: Proceedings of European Conference of Object-Oriented Programming ECOOP 97. (1997)
2. Filman, R.E., Elrad, T., Clarke, S., Aksit, M.: Aspect-Oriented Software Development. Addison-Wesley (2005)
3. Katz, S., Rashid, A.: From aspectual requirements to proof obligations for aspect-oriented systems. In: Proceedings of the 12th IEEE International Conference on Requirements Engineering. (2004)
4. Helsen, S.: Model Transformations with QVT. In: Model Driven Software Development. John Wiley & Sons (2006) 203 – 222
5. Object Management Group: MOF 2.0 Query View Transformation (ad/2005-03-02). (2005)
6. Rashid, A., Moreira, A., Araujo, J.: Modularisation and composition of aspectual requirements. In: Proceedings of the 2nd International Conference on Aspect-Oriented Software Development. (2003)
7. Moreira, A., Rashid, A., Araujo, J.: Multi-dimensional separation of concerns in requirements engineering. In: Proceedings of the International Conference on Requirements Engineering. (2005)
8. Ruzanna Chitchyan, e.: Semantics-based composition for aspect oriented requirements engineering. In: Proceedings of the Sixth International Conference on Aspect-Oriented Software Development. (2007)
9. Yu, Y., d. P. Leite, J.C.S., Mylopoulos, J.: From goals to aspects: Discovering aspects from requirements goal models. In: Proceedings of the 12th IEEE International Requirements Engineering Conference. (2003)
10. Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., Christina, S.: Goal-centric traceability for managing non-functional requirements. In: Proceedings of the 27th International Conference on Software Engineering ICSE 05. (2005)

11. Grundy, J.: Aspect-oriented requirements engineering for component-based software systems. In: Proceedings of the 4th IEEE Symposium on Requirements Engineering. (1999)
12. Tekinerdogan, B., Moreira, A., Araujo, J., Clements, P.: Early aspects: Aspect-oriented requirements engineering and architecture design. In: Workshop Proceedings at AOSD Conference. (2005)
13. Kiczales, G., Mezini, M.: Aspect-oriented programming and modular reasoning. In: Proceedings of the ACM International Conference on Software Engineering. (2005)
14. Kulesza, U., Alves, V., Garcia, A., Lucena, C., Borba, P.: Improving extensibility of object-oriented frameworks with aspect-oriented programming. In: Proceedings of the International Conference on Software Reuse ICSR 06. (2006)
15. Völter, M., Kolb, B.: OpenArchitectureWare und Eclipse. Eclipse Magazin (7) (2005)
16. Völter, M.: MDSD und/oder AOSD? Java Spektrum (2) (2005)