Architecture-Centric Evolution: New Issues and Trends
Report on the Workshop ACE at ECOOP'06

Paris Avgeriou[1], Uwe Zdun[2], and Isabelle Borne[3]

[1] Department of Mathematics and Computing Science,
University of Groningen, the Netherlands
paris@cs.rug.nl
[2] Distributed Systems Group,
Vienna University of Technology, Austria
zdun@acm.org
[3] VALORIA Laboratory,
University of South-Brittany, France
Isabelle.Borne@univ-ubs.fr

Abstract. Software evolution has largely been focused on low-level implementation artefacts through refactoring techniques rather than the architectural level. However code-centric evolution techniques have not managed to effectively solve the problems that software evolution entails. Instead a paradigm shift is emerging, where the evolution approaches put software architecture on the spotlight. This shift focuses on effectively documenting and modifying the architectural design decisions during system evolution, while synchronizing them with both the requirements and the implementation. The second workshop on the theme of Architecture-Centric Evolution attempted to explore the issues that such evolution approaches are dealing with, as well as the trends that emerge in this area. The workshop delved into this field, by presenting the latest research advances and by facilitating discussions between experts.

1 Introduction

Industry and academia have reached consensus that investing into the architecture of a system during the early phases of the system's lifecycle is of paramount importance to object-oriented software development. Moreover, there is an undoubted tendency to create an engineering discipline on the field of software architecture if we consider the published textbooks, the international conferences devoted to it, and recognition of architecting software systems as a professional practice. Evidently, there have been advances in the field, especially concerning design and evaluation methods, as well as reusable architectural artefacts such as architectural patterns and frameworks. And there is growing consensus nowadays about certain aspects of the task of software architecture description, such as the satisfaction of stakeholders' concerns through multiple views, and the use of UML for modelling architectures. Software architecture has become a key issue in the object-oriented community, as architecture is praised for facilitating effective communication between the stakeholders of the system, early analysis of the system, support of qualities and successful evolution of the system.

M. Südholt and C. Consel (Eds.): ECOOP 2006 Ws , LNCS 4379, pp. 97–105, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Unfortunately, in practice, the evolution of software systems largely takes place at the code level. For example, a substantial part of industrial practice of software evolution concerns storing code artefacts in configuration management systems and applying refactoring techniques on them. This hinders the development team from having an overview of the "big picture" and grasping the significant design decisions that appear only at a higher level of abstraction [4]. As a result, the new design decisions that are taken during evolution may compromise or even contradict fundamental principles or constraints of the system's architecture. Moreover, the most substantial properties of the system are its non-functional requirements, the so-called "quality attributes", and the evolution of such properties can only be tackled at the level of architecture. In essence, software architecture is good means for facilitating the synchronization of the system requirements, their evolution, and their implementation during evolution cycles of the system [2].

The theme of architecture-centric evolution is complex and multi-faceted, both in its core and in its relevance to other advances of software engineering. Essentially, it involves at least the following topics:

- Modeling architectures to support evolution of software systems
- Using ADLs or UML to model evolution
- Quality attributes and architectural evaluation in evolution
- Meta-modeling of architectural refactoring
- Architecture model transformations
- Evolution through software architecture patterns
- Architectural design decisions and architectural knowledge in evolution
- Evolution of legacy software through its architecture
- Architecture-centric evolution in the context of service-oriented architectures (SOA) or model-driven engineering (MDE)
- Software engineering processes and methods for architecture-centric evolution
- Theoretical aspects of architecture-centric evolution, e.g. causes of architectural changes
- Synchronizing requirements, architecture and code during evolution
- Evolution in product lines and system families
- Case studies of architecture-centric evolution
- Tools that foster architecture-centric evolution

This paper reports on the second workshop in the series of Architecture-Centric Evolution. The first workshop [5] had focused on the following topics: the metaphor of evolution, evolving components and product lines, languages to support evolution, and consistency of artifacts. The second workshop was of a more interactive nature and the discussion was directed towards the practical means to support evolution. The workshop consisted of four sessions, each lasting 1.5 hours. The first session included the presentation of three papers on formal methods, tools and frameworks to support architecture-centric evolution. The second session revolved around quality attributes in the evolution context, as well as the impact of aspects in the architectural design. The third session revolved around the industrial approach to architecture-centric evolution: tools and methods for architecture-centric evolution that are used in a large company

were presented. The remaining time of the workshop was used to discuss the various issues and challenges that came out of the presentations and identify trends and challenges.

The rest of this workshop report is organized as follows: Section 2 outlines the contents of the papers that were presented in the workshop, as well as some of the discussion they raised. Section 3 describes the findings of the dialogue triggered by the previous sessions and the conclusions reached by the participants. Finally Section 4 concludes with a brief synopsis of the state-of-the-art and future trends.

2 Issues in Architecture-Centric Evolution

The essence of each paper as well as the key points of the raised discussions are summarized in the sub-sections below. The heading of each sub-section is the title of the corresponding paper. The papers presented during the workshop are available on-line at: *http://www.cs.rug.nl/ paris/ACE2006/*.

2.1 Modelling Software Evolution Using Algebraic Graph Rewriting

Selim Ciraci and Pim van den Broek have worked on an approach to formalize evolution requests with the help of algebraic graph rewriting. They considered UML class and interaction diagrams (especially classes, methods and parameters) as the main source of architectural information and proposed the transformation of these diagrams into colored graphs. Changes in these diagrams represent changes during system evolution, which can then be formalized by algebraic graph rewrite rules.

Ciraci and van den Broek claimed that their approach is language-independent and thus extensible for other kinds of UML diagrams or other Architecture Description Language (ADL) representations. As a natural follow-up question, the speaker was asked whether tool support was possible, given the independence of modeling languages. The answer was positive and the speaker asserted that tool support can be automated at two levels: first by converting models made in the specific modeling language to marked or colored graphs; second by subsequent transformation of the graphs when evolution requests occur (through the push-out mechanism). There is no automated tool support but the authors plan to implement appropriate tooling, firstly for the Unified Modeling Language. At the level of code, refactoring issues can also be tackled by this approach, as long as there is a possible reverse engineering of the code into the appropriate ADL.

2.2 Meta-architecture Tools: The Working Engines of the Company's Architectural Evolution

Ethan Hadar presented an industrial approach on the subject of architectural evolution, based on the paper he co-authored with Irit Hadar. Their perspective comes from a large multinational enterprise, that works with multiple different methodologies, products, product lines, technologies, tools, communities etc. In this context, architecture-centric evolution becomes a rather complex and risky issue. The speaker presented the Meta-Architecture tools constructed within their company and the corresponding method of operations, as well as its challenges and solutions. They focus on Software Architecture

Analysis and mainly work with three types of diagrams: business services, component interfaces and deployment. They pay particular attention to the traceability of the different artifacts produced. The presenter professed a rather radical view of their approach: the project management model follows the waterfall lifecycle model, while their development is rather agile. Furthermore the models produced are mainly aimed for use by managers, while the developers work with the source code.

According to the speaker, an architecture-centric evolution approach is not only a matter of implementing the appropriate methods and tools. Most importantly it is a matter of aligning the practices and goals of the organization with the corresponding line of thought. Another important remark is that in the pragmatic constraints of big corporate software development, research approaches, hypes and fashions are temporary and thus not so important. Their approach is to combine the best of all worlds. For instance they try to combine different concepts taken from the fields of Model Driven Architecture, Aspect-Oriented Programming, Software Architecture, etc., with the ultimate goal to make them work in practice and when needed.

2.3 Architectural Stability and Middleware: An Architecture-Centric Evolution Perspective

Rami Bahsoon presented a paper, co-authored by Wolfgang Emmerich, that focuses on architecture stability: how far a software system can endure changes in requirements, while leaving the architecture of the software system intact. They argue that architectural stability is more threatened by changes in non-functional rather than in functional requirements. They studied in particular architectures that are strongly based on middleware, and put specific emphasis on the non-functional requirements that are important in this area, such as scalability, fault tolerance, etc. The goal is to facilitate their evolution over time. Bahsoon and Emmerich have shown through a case study, how a software architecture, when induced by distinct middleware, differs in coping with changes in these non-functional requirements.

Bahsoon was asked about the relation of their approach to architectural patterns and styles, since they, just as middlewares, determine a great part of the system architecture. The presenter responded that architectural patterns have a large impact on architectural stability and therefore the system evolution. However, which architectural patterns are more stable in face of change, is still an open research question. It would thus be fruitful to conduct impact studies of architectural patterns to the system evolution and perhaps classify the patterns in this respect.

The discussion then revolved around the applicability of their approach with respect to the architectural documentation of middleware. In specific the question was raised whether, there exist documented reference architectures for middleware that can be reused in designing and evaluating the overall software system architecture. According to the speaker's experience, middleware architectures were rarely publicly documented, and the non-functional requirements (quality attributes) are difficult to assess. The only viable way is to study and use the middleware extensively.

The evolution of the middleware itself is not tackled in this approach, even though it apparently has a large impact on the overall system evolution. In principle, one can

either treat it as a black box, waiting for the community to evolve it, or continue the development of the middleware with own resources.

2.4 Change Impact Analysis of Crosscutting in Software Architectural Design

Klaas van den Berg presented an approach for impact analysis of crosscutting dependencies in architectural design. According to the presenter, the analysis of the impact of changes in requirements can be based on the traceability of architectural design elements. Especially the crosscutting dependencies between the design elements may have a strong influence on the modifiability of software architectures. The proposed impact analysis is supported by a matrix representation of dependencies.

The presenter argued that crosscutting concerns must be represented at two levels: source and target. Source elements crosscut with respect to their mapping to the target elements. The type of dependencies between source and target are very important in this respect and one must explicitly define them or one must justify why one did not define them. Eventually there are numerous inter-level and intra-level dependencies that can be traced, but not all of them should be traced. Van de Berg stressed the importance to define the goal of traceability: why artifacts need to be traced (e.g. for testing or analysis purposes).

2.5 Using ATAM to Evaluate a Game-Based Architecture

Ahmed BinSubaih presented the next paper co-authored by Steve Maddock. Their research work examines the suitability of employing an off-the-shelf software architecture evaluation method, ATAM, in order to assess systems in the gaming domain. They conducted a case study of a specific game-based architecture, evaluating it for its key drivers. Their findings were encouraging, as the method can clearly reveal the strengths and weaknesses of the architecture, which can then be guarded and addressed respectively before evolving the architecture further. In addition they proposed a small extension to the method: a view that consolidates disparate outputs generated by ATAM.

The discussion that followed concerned the issue of traceability, and in particular that of the architectural decisions. The application of ATAM produced backwards traceability of the decisions to the requirements. However, the presenter agreed with the participants that architectural decisions should also be traced forward to the architectural models, to facilitate successful evolution.

2.6 Safe Integration of New Concerns in a Software Architecture: Overview of the Implementation

Olivier Barais presented a paper co-authored by Hanh-Missi Tran, Anne-Francoise Le Meur, and Laurence Duchien. This research concerned a framework for integrating new concerns into a software architecture, by factorizing the implementation of concerns into separate units, called ‘patterns’. Their approach describes a rule language, that prevents erroneous concern integrations from being expressed, detects others by static verifications and identifies compatible join points between a ‘pattern’ and a basis architecture.

The presenter claimed that reusing components depends on a number of factors: making explicit all the dependencies of a component; separating the concerns in the

software architecture; modularizing concerns in the form of aspects; applying an iterative integration process. The workshop participants discussed the notion of architectural mismatches caused by the assumptions made when integrating new components. The presenter argued that this approach provides a solution by making the assumptions explicit through both a static and a behavioral model, plus the necessary transformation rules. The presenter noted that the shortcoming of the approach is the focus on static analysis, since the mismatches cannot be explicitly documented and then tested at run-time.

The participants argued that the proposed approach has a shortcoming in the industrial context: one cannot really stop the development and freeze the code before a new component or aspect can be integrated. First, because there are many changes coming from different places that need to be accommodated simultaneously. Second, because not all 'patterns' and not all changes can be found and automatically accommodated; there needs to be some extent of manual intervention. The industrial point of view in such cases is to avoid automating everything; instead to allow for as much flexibility as possible.

This approach raises the process issues of architecture-centric evolution. The technical solutions need to be aligned to the pragmatic constraints, as they occur for instance in the industry. The importance of making the architectural knowledge explicit and sharing it with the stakeholders comes out as a necessity.

2.7 A Generic Framework for Integrating New Functionalities into Software Architectures

Guillaume Waignier presented a paper co-authored by Anne-Francoise Le Meur and Laurence Duchien, that was closely related to the previous presentation. The presentation concerned the evolution of software through a generic framework for automatically integrating new functionalities into an architecture description. In contrast to the previous presentation that was bound to a specific Architecture Description Language (ADL), this approach is ADL-independent. The research group conducted a domain analysis on ADLs and came up with a generic ADL model to manipulate and reason about architectural elements involved in integration. Their approach is complemented by high-level abstractions to describe different kinds of integration, as well as a generic integration engine.

The presenter stressed the added value of this approach: the focus on integration issues, when defining the common metamodel of Architecture Description Languages. The workshop participants noticed the lack of explicit Quality Attributes in the metamodel. The presenter explained that quality attributes can be dealt with in the traditional way: by combining their approach with an architecture evaluation method, such as ATAM. The proposed approach can however provide hints about a specific quality attribute: consistency of the architecture. Consistency can be automatically checked and furthermore it be associated to architectural tactics and patterns.

Finally, the participants discussed the issue of runtime reconfiguration with specific focus on integrating the components of the 'pattern' during runtime. This issue could also be dealt with in this approach, as long as the sequence of modifications is strictly kept.

3 Discussion and Outcomes

The last part of the workshop was a discussion session, in which we summarized the participants' insights into architecture-centric evolution gained from the presentations and discussions, and elaborated on the different concerns that arose.

First the workshop members agreed on the importance of tools on architecture-centric evolution. In fact, the different approaches seemed to have commonalities in the tools that support them. We can extract some useful conclusions by examining closer what exactly the different tools support and what kind of target users they are aimed at.

A recurring topic during the workshop was the gap between the industry and academia. It is a rather typical software engineering phenomenon: the distance between the researchers who propose novel methods and tools and the potential adopters in the industry is substantial. The academic approach on architecture-centric evolution is mostly theoretical and considers the processes and the models that need to be formalized to result in automatic or semi-automatic tools to help the developers. In the industry, economical and timing issues are crucial and consequently the area of software architecture and evolution is viewed from a pragmatic perspective.

Another significant parameter that was discussed is that of human behavior, e.g. communication issues between different stakeholders, the way architects take decisions etc. Just like any field of software engineering, the human aspect must be taken under consideration when dealing with architecture-centric evolution.

The workshop focused mostly on technological issues and approaches, but also there are important process issues and organization issues. It is quite common that these two worlds, technology and organization, are studied separately. To overcome this problem, the idea was discussed to apply something analogous to the architecture-business cycle [1] in order to bridge the gap between technology and process/organization. The result would be a paradigm that makes explicit the influence of one to another and vice versa in an iterative cycle.

During the workshop, some approaches were presented on automating the evolutionary mechanisms, e.g. the integration of aspects on an entire software architecture all at once. Integration concerns various kinds of elements, mainly components and objects, according to different techniques (e.g. marked graphs, separation of concerns). The participants agreed that we should not over-automate but rather allow for some flexibility in order to incorporate more human intervention and manual work into the processes. Architects and designers should be able to take decisions when appropriate. We should strike a balance between automation and manual work.

Another issue that is considered a 'hot' research topic, dynamic evolution, and particularly dynamic reconfiguration, is necessary and urgent for modern software development. However the workshop members were skeptical about how to implement such a mechanism due to unsolved problems, e.g. monitoring the run-time system, quality of service evaluation, etc. This is a crucial topic in cases where the system needs to evolve without being shut down, but indeed, its implementation is still problematic.

The workshop participants pondered over the meaning of architecture-centric evolution and how to achieve it. A simplistic but perhaps naive opinion that has been proposed in the past, is merely to synchronize architectural documentation when the software product is evolved. This can be achieved either by documenting the decisions

taken by the developers, or after the fact, through reverse-engineering. However, this leads to code-centric rather than architecture-centric evolution. Updating the architecture document in this case is a side-effect of changing the code. Architecture is not in the center but is a by-product, a documentation outcome. A paradigm shift is necessary in order to move away from this code-centric approach and strive towards a real architecture-centric one. In architecture-centric evolution, the role of architecture is central to facilitating evolution. It is the starting point of incorporating changes and all other artifacts should follow. There are two trends towards this goal, that were identified during the presentations but also during the plenary discussion: the importance of traceability, and the emergence of the aspect-oriented paradigm.

First, the lack of effective traceability, was a recurring problem identified that currently hinders architecture-centric evolution. Traceability can be defined as "the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another" [3]. In the case of architecture-centric evolution, traceability is key in synchronizing the requirements specification, the architecture description, and the code. These three types of artifacts need to be traced backward and forward in order to keep them synchronized when one of them changes. This issue was also largely discussed in the first ACE workshop [5], where the consistency of artifacts between evolution was deemed of paramount importance. It is common place that traceability methods still need to be developed before they can successfully support Architecture-Centric Evolution.

Second, the role of *aspects* as a means to better manage evolution was largely discussed during the workshop. The rationale is that when we try to incorporate new requirements or concerns, these may be crosscutting in the architecture and need to be implemented horizontally. On the one hand, we must integrate them into the architecture by the means of a weaver. On the other hand, architectural artifacts must be connected (i.e. traceable) to other levels (e.g. to requirements and to the implementation), as well as to evolution versions and we can define crosscutting dependencies between the architectural artifacts. It is, however, not necessary that roundtrip traceability is supported.

4 Epilogue

The second workshop on the theme of Architecture-Centric Evolution (ACE 2006) hosted interesting presentations and discussions. The authors of the papers mainly focused on specific techniques and approaches from a technical point of view. We can claim that there are good practical and research solutions for particular technical problems. However, architecture-centric evolution is still quite preliminary and there are numerous open research issues that must be addressed. Specific issues that recurrently came up during the presentations and discussions were: how to connect architectural evolution and quality attributes; how to bridge the gap between research and practice; what is the role of aspects in architectural evolution; how to include organizational aspects and keep consensus between all stakeholders of a system. Even though architecture-centric evolution is still a young approach, there was consent among the

participants that the approach is gaining importance both in research and practice. This is also apparent by the number of research projects, scientific papers, technical reports and tools that appear in this multi-faceted area.

References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice 2nd Edition*. Addison Wesley, Reading, MA, USA, 2003.
2. P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley, 2002.
3. Institute of Electrical and Electronics Engineers. *A Compilation of IEEE standard Computer Glossaries*. IEEE, New York, NY, USA, 1990.
4. A. G. J. Jansen, J. van der Ven, P. Avgeriou, and D. K. Hammer. Tool support for architectural decisions. In *6th IEEE/IFIP Working Conference on Software Architecture (WICSA)*, Mumbai, India, January 2007.
5. U. Zdun and P. Avgeriou. Architecture-centric evolution. In *ECOOP 2005 Workshop Reader*, LNCS. Springer, 2006.

Appendix: Acknowledgement

We extend our thanks to all those who have participated in the organization of this workshop, particularly to the program committee, which is comprised of:

– Goedicke Michael, University of Essen, Germany
– Yann-Gael Gueheneuc, University of Montreal, Canada
– Guelfi Nicolas, University of Luxembourg, Luxembourg
– Dieter Hammer, University of Groningen, the Netherlands
– Heckel Reiko, University of Leicester, UK
– Laemmel Ralf, Microsoft Corporation, USA
– Oberleitner Joe, Technical University of Vienna, Austria
– Nicolas Revault, University of Cergy Pontoise, France
– Wermelinger Michel, Open University, UK