A Lightweight Architecture Recovery Process

Davor Svetinovic and Michael Godfrey
Software Architecture Group
University of Waterloo
Waterloo, Ontario, N2L 3G1, CANADA
Tel. (519) 888-4567 x5388
{dsvetinovic, migod}@uwaterloo.ca

ABSTRACT
In this paper, we present an overview of a lightweight approach for software architecture recovery. The main advantages of the process are the lightweight recovery of architectural semantics, and the compatibility with the highly iterative adaptive development processes that involve extensive architectural refactorings.

Keywords
Software Architecture, Reverse Engineering, Product Lines, Agile Development Processes

1 Introduction

One of the most controversial aspects of Extreme Programming(XP) and other lightweight development methodologies [5] is an almost complete rejection of formal and semi-formal upfront design activities. This rejection is backed up by the arguments that these design activities do not work very well in practice due to a variety of technological, management and human factors:

- Most developers like to concentrate exclusively on programming.
- Design documents are almost never in synchronization with code.
- During active development, lot of time is spent updating design documents.
- The produced documents are not used much by programmers.
- Tool support for roundtrip engineering is not good enough.

The main design activities and principles practiced in lightweight processes are:

- Use of CRC cards [4] for initial, quick design.
- No design documentation. Code and programming conventions are considered to be sufficient for design recovery. CRC cards are not kept.
- Use of system metaphor (common problem domain vocabulary) for design abstraction purposes.
- Simplicity and adaptive design approach — Simple design and refactoring make system easier to change to accommodate new requirements than to try to predict them and build an infrastructure to support them [5].
- Continual refactoring [6].

This approach is most useful for adaptive design [5] of small and mid-sized applications. However, in order to make the process scalable, a more systematic approach to architecture design must be introduced.

2 Agile Architecture Development Process

We are currently working on a process whose main goal is to be an efficient, lightweight, and cost effective choice for architecture development. The target development processes are the lightweight ones, and ones without a formal architecture design component. Besides preserving the lightweight development principles [11], the process aims to satisfy following goals:

- Minimal additional developer training involved.
- Risk-free incorporation in the development process
- Robust roundtrip support.
- As much as possible based on the most accepted programming and design practices.
- Minimize design activities and maximize programming.
- Minimal design documentation.

The process consists of three sub-processes in order to increase the compatibility and integration with different development processes. The process presented in the rest of this paper is concerned with architecture recovery. The second process is a lightweight forward architecture design process that balances predictive and adaptive design principles. The third one is an incremental product line design process, which is aimed at the design of product lines for a set of small to mid-sized systems. Some of the methodologies and principles that have influenced the development of the method are described in related work [1, 2, 3, 4, 7, 8, 9, 10].

All three processes emphasize accommodation of very short development cycles, extensive refactoring, and incremental work.

3 Architecture Recovery

Very short development cycles and refactoring result in constantly changing architecture. In order to deal with architecture drift and erosion in such short cycles, there is a need for an efficient architecture recovery process, which recovers not only the structure of the system, but also the rationale of that structure.

Architectural rationale is recovered and presented through the use of functional and quality attributes and goals. Goal and attribute recovery is performed through the merging of the problem domain architecture and the concrete system architecture. All the information is encapsulated within the code and can be automatically recovered through the use of tools, which are described later.

The main recovery activities are:

1. Derivation of an initial set of goals and conceptual architecture, achieved through the study of available documentation and stakeholder interviews.

2. Extraction of the actual code level structure of the system using a tool like PBS [10]. This information is used in later steps to present the concrete static architecture of the system.

3. Iterative interface-driven goal-responsibility code instrumentation, which is used to recover and analyze concrete system use-cases. This use-case recovery can be automated with proper tool support. Preconditions, post-conditions, and invariants are introduced to capture the goal and attribute constraints.

4. Documentation of architecture supported attributes using separate descriptions, which include traceability information.

5. Representation of the concrete architecture and its rationale using the information obtained in previously mentioned activities. This information can be presented using various formats (i.e., decision/rationale architecture diagrams).

The tool mentioned in step 3 can be enhanced to recover and manage the information captured in step 4, and to provide full navigability and traceability of the architectural presentation.

The major advantages of the approach are that it can be used to recover the architecture of the system incrementally, and that it relies only upon the use of goals and attributes to capture the semantics of the architecture. If the responsibilities and constraints are used and documented during forward system design, the architecture and its semantics can be automatically recovered and kept updated.

In addition to the recovery of the architecture, the application of the process results in the higher quality of the code since it relies upon the use of proven software engineering practices.

4 Conclusion

The integration of responsibility-driven design and Design by Contract with attribute theory, and their automated reverse application provide a lightweight and direct approach to the recovery and preservation of architectural semantics.

5 Future Work

Future work consists of building the tools for process automation, application of the process to the large software systems, and refinement of the process based on the experiences and integrations with different development processes.

REFERENCES

[1] F. Bachmann, L. Bass, G. Chastek, P. Donohoe, and F. Peruzzi. The Architecture Based Design Method. CMU/SEI-2000-TR-001.

[2] B. Boehm, and H. In. Identifying Quality-Requirement Conflicts. *IEEE Software*, Vol. 13, No. 2, (March 1996).

[3] I. T. Bowman, R. C. Holt, and N. V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. *ICSE '99: International Conference on Software Engineering*, Los Angeles,California, May 1999.

[4] A. Cockburn. Responsibility-based Modeling. http://members.aol.com/humansandt/techniques/responsibility.htm.

[5] M. Fowler. The New Methodology. http://www.martinfowler.com/articles/newMethodology.html.

[6] M. Fowler. Refactoring: Improving the Design of Existing Code *Addison-Wesley Pub Co*, ISBN: 0201485672, 1999.

[7] R. Kazman, M. Klein, and P. Clements. ATAM: Method for Architecture Evaluation. CMU/SEI-2000-TR-004.

[8] M. H. Klein, R. Kazman, L. Bass, J. Carriere, M. Barbacci, and H. Lipson. Attribute-Based Architecture Styles. *Proceedings of the First Working IFIP Conference on Software Architecture (WICSA1)*, San Antonio, TX, 225-243, (February 1999).

[9] P. Kruchten. The 4+1 View Model of Architecture. http://www.rational.com/products/whitepapers/350.jsp.

[10] PBS: The Portable Bookshelf. http://swag.uwaterloo.ca/pbs/.

[11] Principles: The Agile Alliance http://www.agilealliance.org/principles.html.