A Method for Recovery and Maintenance of Software Architecture

Short Talk for Andrei Ershov Fourth International Conference

Dmitry Koznov, Konstantin Romanovsky, and Alexei Nikitin

{dim,kostet,lex}@tepkom.ru
St.–Petersburg State University, Faculty of Mathmatics and Mechanics
Department of Software Engineering
198504 Bibliotechnaya sq., 2, St.–Petersburg, Russia

Abstract. This paper proposes a method for recovery and subsequent maintenance of the architecture for actively evolving software systems. The method's underlying idea has to do with constructing a basic set of the architecture elements, which set would then be used for creating different views of the system. The responsibilities of the modules, which make up the software system, as well as elements of the system's data dictionary, are considered elements of this basic set. Of special meaning is the fact that the system is being actively maintained and developed, that the knowledge about it is accessible, but needs to be alienated from the respective bearing media, to be generalized and formalized.

1 Introduction

While many software products, for which the architecture was never formalized, may exist for years and be successfully maintained, still one can face a situation when formalization of the system architecture becomes a priority task. As a result, a lot of architectural imperfections in the system reveal themselves, and so there comes such moment in the life of the product, when its further development is impossible without improving the architecture, which requires formalizing the latter. Even though various methods of analyzing and designing software systems have been spreading widely [11, 9, 10], it is a great problem to use such methods in the situation like this. On the one hand this activity can cause serious internal restructuring of the system. On the other hand, we should take into account the peculiarities of the system being analyzed ad a "living" system, particularly we cannot ignore the commercial aspects of the software development process, issuing of new versions, service packs, and the implementation of new customer's requests.

Thus there is a problem: how to perform reverse engineering of the architecture of the actively evolving system, with most effective gathering and formalisation of the meta-information on the system. At the same time the architecture views should be supported in actual state while the system will evolve.

2 Related Works

There are many methods and tools of reverse engineering designed for recovering the knowledge about a system architecture based on source code parsing (Refine/C, Imagix 4D, Rigi, Sniff [6, 9], RescueWare). Also there are formal methods of the reverse engineering [3].

All these methods have the same weak point: absence of mechanism for synchronization of the model recovered with the source code of the software system. This problem makes very ineffective the use of such methods for actively evolving systems. From this point of view, there are more perspective Use-case driven methods [4], because the models that were derived with this method should be more stable to the system's changes on the practice. Data-mining methods [7], relying on the inner links between system elements are also interesting from the same viewpoint. However, all these methods are more suitable for the architecture of the "dead" system.

Round-Trip development methods, that are provided with some CASE-packages (e.g. Rational Rose, Together), enables the bi-directional connection of source codes and the architecture view. But the quality of the information visualized is unsatisfactory, because in fact, we do not get any new meta-information on the system, but only visualize the source code structure.

3 Starting Point

Let's formulate the basic problems that are to be solved with the presented method of architecture recovery:

1. The utilization of the features of the "living" system for the most effective architecture recovery;
2. The ability of the maintenance and further development of the architecture view of the software system;
3. The ability of step-by-step adopting of the process of development and support of the architecture without any serious damage to industrial requirements to the process.

4 The Method

The method propesed in this paper consists of the following parts:

1. Model - means or architecture formalization.
2. Principles of formalizing architecture by means of the model propesed.
3. Principles of maintaining the formalized architecture model.

4.1 Model

The system architecture model proposed with this method consists of the following views: Structure views, Dynamic views and Physical view.

The basis of describing the system architecture is a System Structure description, which it is proposed to implement on the physical level (Physical view) and the logical level (Structure views). The Dynamic views are needed in order to determine the main scenarios of the system's operation.

The necessity of different kinds of views of the software is well known [1, 5, 8]. With the method in discussion, it is proposed to construct also a set of various Structure and Dynamic views, which is motivated by the following considerations:

1. The participants of the project, whose positions are on different levels of the hierarchy (managers of different levels) need information about the system to be specially adapted;
2. There exist both a vertical division of the system (by business functions) and a horizontal one (by tiers – e. g., User Interface, Business-Logic, Data Access);
3. There are different modes of packaging and deployment of the system;
4. In order to compile the entire project, information about the structure of storing the source codes of the system is needed;
5. Organizational structure of the enterprise affects decomposition of the system.

Structure Views. It is proposed to organize Structure views in the form of UML class diagrams. We herewith associate some part of the system (subsystem) with a class. The subsystems are organized into a multiple containment hierar-

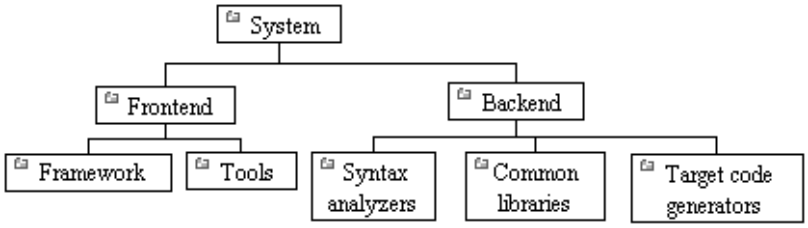

Fig. 1. Upper part of a Structure View for the reingineering system, based on the functional decomposition

chy, with the only restriction that the aggregated subsystems become invisible from the context, in which their aggregate exists. The different views should be constructed upon the same basic set of subsystems, but in other respects do not require any special matching. Associations between the subsystems, which reflect their semantic connections, are possible on each level.

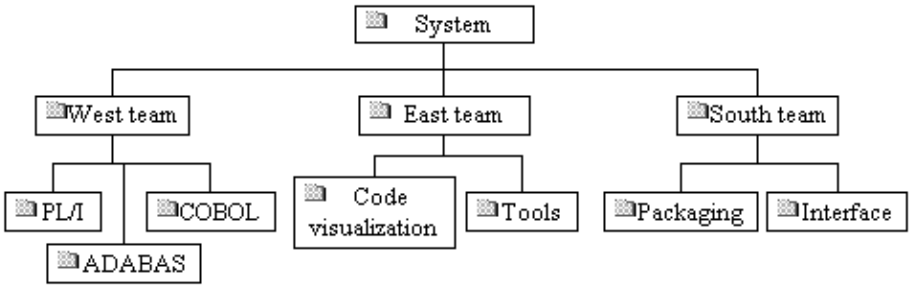

Fig. 2. Upper part of a Structure View for the reingineering system, based on the team location

On fig. 1 and 2 the fragments of two structure views for the reengineering systems are depicted. We consider an reengineering system that analyzes programs on COBOL, PL/I, Adabas/Natural and transforms them to modern platforms. On the fig. 1 you can see the decomposition of the system being modeled, based on the functional features. On the other hand, fig. 2 shows the decomposition of the same system, based on the teams, performing the development.

This sample demonstrates the profit of multiple Structure Views for the software, bacause they reflect different points of view, characterizing the system being analyzed. Synchronization of this views is achieved , because of the fact that all these views are built on the Basic Set, defined below.

Dynamic Views. With the help of the dynamic views it is proposed to represent the main scenarios of the system's operation. One can associate with each level of a structure view a dynamic view, which would explain how the subsystems interact with each other. For this, it is proposed to use the UML Collaboration Diagrams.

Physical View. This view is designed for inventory of the software source codes and for associating them with elements of the structure and dynamic views. In the view, the following items are considered:

1. Set of program modules (e. g., for Microsoft Visual C++ these are projects);
2. System data dictionary, which consists of:
   (a) Persistent data (logical data of the system and the corresponding physical media);
   (b) Channel data, with which subsystems exchange not through persistent-structures (physically media are absent, only logical elements of the data are there);
   (c) Configuration data that constitute a kind of persistent data, but are responsible for tuning the system algorithms (logical data and the corresponding physical. media).

4.2 Model construction principles

The main stage in the construction of the model is the Basic Set construction. All structure views will be built in terms of the Basic Set. In order to keep the correspondence of the views to each other and to make the maintenance easier, all elements of such views are to be subsets of the basic set of subsystems. So, the views themselves are hierarchical coverages of the basic set: the elements of each view form an aggregation hierarchy and the set of leaf nodes in that hierarchy is a usual coverage of the basic set. The basic set is constructed from responsibilities, which are assigned to the system modules (3–5 responsibilities per each module) and with elements of the system data dictionary.

Let's continue considering the example of the reengineering system. One of the main modules, containing the basic services, used by different system parts has the following responsibilities:

1. Memory management.
2. Abstract nodes of syntax trees for different input languages.
3. Supplementary data structures (configuration data, formatted files etc) and algorithms of their processing.

The process of the basic set construction is shown in fig. 3. First, all the existing software elements should be listed in a common inventory. The responsibilities of each element in inventory should be collected in a responsibilities set. Sometimes several software elements participate in the implementation of the same responsibility. After such responsibilities are discovered, they should be equijoined to construct the basic set.

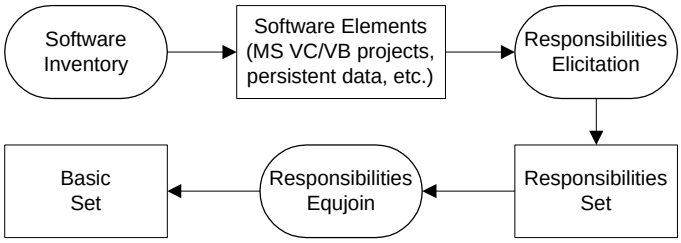

Fig. 3. Process of the Basic Set construction
Legend: on this figure the Data Flow Diagams notation is used, the process is depicted as an oval, and the data are depicted as a rectangle.

When construction of the basic set is finished, any participant of the development process may build for themselves a package of structure and dynamic views that they need. A coordination of different views is provided by due to integrity of the set of basic elements, which are placed on different diagrams.

4.3 Model Maintenance Principles

Because the system being modeled is a "living" system, changes to the software are inevitable. That's why we should study also the ways of keeping the models in actual state, i.e. the process of the model maintenance.

Let's discuss the architecture maintenance process, which scheme is presented on the fig. 4. After some changes were made to system software, staff should make the set of responsibilities consistent with its real state. All modifications could be classified as responsibility adition, deleting, change or migration. Then it is needed to modify corresponding Basic Set elements. After that the information about the subsystems, directly or indirectly containing modified elements, should be inspected and updated if needed. At last, Dynamic view elements, i.e. collaboration items, should be updated.

It should be noted that all views in proposed approach depend on the single source of changes - the Basic Set, this fact makes the maintenance of the model easier. Otherwise, when some changes occures in the system, there would be a lot of different views, that need to be changed separately, but this is very time-consumimg as well as integrity could be broken.

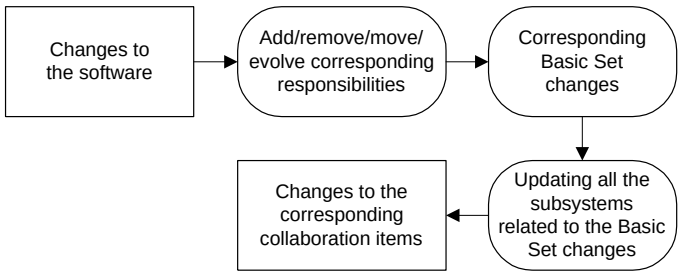

Fig. 4. Architecture Model Maintenance Scheme

5 Conclusions and Futher Research

Here are the conclusions and some directions of further work:

1. Responsibility gathering practice should be elaborated: common problem domain dictionary should be used, responsibilities should be of comparable detail and shouldn't intersect.
2. Architecture model representation based on UML should be detailed. Particularly, it is not obvious which UML element should be used for subsystem representation.
3. Additional experiments with large real projects should be carried out.
4. Special graphical tool for architecture recovery and maintenance may be implemented.

References

1. J.Rumbaugh, I.Jacobson, G.Booch. The Unified Modeling Language Reference Manual. – Addison-Wesley, 1999.
2. Daniel Aebi: Data Re-Engineering – A Case Study. – ADBIS 1997, 305–310
3. Liu, H. Yang H. Zedan : Formal Methods for the Re- Engineering of Computing Systems: A Comparison // X. COMPSAC '97 – 21st International Computer Software and Applications Conference, – 1997
4. Christian Lindig, Gregor Snelting: Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis. – ICSE 1997, 349–359
5. Philippe Kruchten : 4+1 view model of architecture. – IEEE Software, November 1995, 42–50
6. Berndt Bellay and Harald Gall : A Comparison of four Reverse Engineering Tools // Working Conference on Reverse Engineering (WCRE '97) – October 6-8, 1997
7. K. Sartipi, K. Kontogiannis, F. Mavaddat : Architectural Design Recovery using Data Mining Techniques. // IEEE European Conference on Software Maintenance and Reengineering (CSMR 2000), pages 129-139, 29 Feb - 3 March 2000, – Zurich, Switzerland, 2000
8. Philippe Kruchten : The Rational Unified Process. – Addison Wesley Longman, Inc, 1999
9. M.N. Armstrong, C. Trudeau : Evaluating Architectural Extractors. – IEEE Software 1998, 30–39
10. Ivar Jackobson : Object-Oriented Software Engineering. – Addison-Wesley, 1993
11. Grady Booch : Object-Oriented Analysis and Design. – The Benjamin/Cummings Publishing Company Inc., 1994