Software Evolution: Analysis and Visualization

Harald C. Gall
Department of Informatics
University of Zurich
gall@ifi.unizh.ch

Michele Lanza
Faculty of Informatics
University of Lugano, Switzerland
michele.lanza@unisi.ch

ABSTRACT

Gaining higher level evolutionary information about large software systems is a key challenge in dealing with increasing complexity and decreasing software quality. Software repositories such as modifications, changes, or release information are rich sources for distinctive kinds of analyses: They reflect the reasons and effects of particular changes made to the software system over a certain period of time. If we can analyze these repositories in an effective way, we get a clearer picture of the status of the software. Software repositories can be analyzed to provide information about the problems concerning a particular feature or a set of features. Hidden dependencies of structurally unrelated but over time logically coupled files exhibit a high potential to illustrate software evolution and possible architectural deterioration. In this tutorial, we describe the investigation of software evolution by taking a step towards reflecting the analysis results against software quality attributes. Different kinds of analyses (from architecture to code) and their interpretation will be presented and discussed in relation to quality attributes. This will show our vision of where such evolution investigations can lead and how they can support development. For that, the tutorial will touch issues such as meta-models for evolution data, data analysis and history mining, software quality attributes, as well as visualization of analysis results.

Categories and Subject Descriptors:
D.2.7 [Software Engineering]: Restructuring, reverse engineering, and reengineering, version control
General Terms: Design
Keywords: Software evolution, visualization

1. OBJECTIVES

The goal of this tutorial is to investigate means to *analyze* and *control* the evolution of a software system at various levels. Specifically, it aims to answer the following questions:

1. How does the architecture of a software system evolve over time? What are signs of architectural decay and how can they be tracked down?
2. How can hidden dependencies in a system that complicate and hinder its evolution be discovered? How can existing analysis methods be adapted, revised, or enhanced to enable that?
3. How can the plethora of software data (*e.g.* source code, change and bug history, release data) be filtered and visualized? What are effective visualization models and techniques for that?

We propose to tackle these questions by exploiting and understanding the huge amounts of information which reside in versioning and bug tracking systems, but which seem to be largely ignored by software industry. We focus on large-scale software systems both from the open source and the industrial area. The tutorial will provide participants with:

- An overview of the domain of software analysis, software evolution, software visualization
- A survey of methodologies for software analysis
- Quantitative analyses and software evolution metrics
- Qualitative analyses
- Studies of Open Source Software projects
- Evolution data mining
- Evolution visualization tools

Our research is motivated by the conviction that *software evolution* is the key to productivity in software engineering processes, and that consequently we should focus our research efforts on activities that enable and facilitate the evolution of in complex software systems.
In our recent work we have investigated *retrospective* evaluation of software evolution by looking at successive releases of a software system to analyze how smoothly the evolution took place. Intuitively, one wants to see if the system's architectural decisions remained intact throughout the evolution of the system, that is, through successive releases of the software. This intuitive idea is called "architectural stability" [10] and our research is concerned with this aspect.
In [5, 6] we analyzed the history of changes in software systems to detect the hidden dependencies between modules. However, their analysis was at the file level, rather than dealing with the real code and considered release and version information of software units (modules, files, etc.) as well as change reports. In [11] we described a visualization approach to allow an engineer to quickly grasp the evolution "nature" of modules, differentiating stable from more volatile ones with respect to change and growth trends.

ICSE'06, May 20–28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

In our further work we concentrated on integrating change requests, bug reports and modification reports to reason about *logical couplings*—couplings that stem from software parts being changed together over several releases and in contrast to couplings that stem from the source code. As a result we have defined a filtering mechanism and a data scheme for such an integration in [3].

In [2] we analyzed the relation to bug reports to track the hidden dependencies between system features. By instrumenting the code we showed how features are scattered over the project tree and how features are logically coupled over releases. We provided some specific visualizations for the Mozilla system and integrated both problem and modification reports with a multidimensional scaling technique [1]. This approach allows an engineer to uncover hidden dependencies among different features over many releases.

In [7] we discovered logical couplings by analyzing CVS data: developers checking in and out files within certain periods of time and the relationship of these files discovered dependencies that are difficult to detect by other means and pointed to several bad code smells [4] by means of visualizations.

We have investigated software evolution visualization also from a completely different angle: Lanza's Evolution Matrix [12] displays the system's history in a matrix in which each row is the history of a class. A cell in the Evolution Matrix represents a class and the dimensions of the cell are given by evolutionary measurements computed on subsequent versions. In [13] we have demonstrated the corresponding tool *CodeCrawler*.

In [9] we defined the history of software as a first class entity and then we defined history measurements which summarize the evolution of an entity. We used the measurements to display evolutionary polymetric views. More sophisticated views on several metrics have been described in [14] in which Kiviat diagrams have been used to show snapshots of accumulated evolution history in many metrics dimensions.

In [8] the term *Yesterday's Weather* was used to depict the retrospective empirical observation that at least one of the classes which were heavily changed in the last period will also be among the most changed classes in the near future. We computed it on two case studies and showed how it can summarize the changes in the history of a system. We used the approach to pinpoint classes in the latest versions which would make good candidates for a first step in reverse engineering.

In this tutorial, we describe the investigation of software evolution by taking a step towards reflecting the analysis results against software quality attributes. Different kinds of analyses (from architecture to code) and their interpretation will be presented and discussed in relation to quality attributes. This will show our vision of where such evolution investigations can lead and how they can support development.

For that, the tutorial will touch issues such as meta-models for evolution data, data analysis and history mining, software quality attributes, as well as visualization of analysis results.

2. REFERENCES

[1] M. Fischer and H. Gall. Visualizing Feature Evolution of Large-Scale Software based on Problem and Modification Report Data. *Journal of Software Maintenance and Evolution*, 2004.

[2] M. Fischer, M. Pinzger, and H. Gall. Analyzing and relating bug report data for feature tracking. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003)*, pages 90–99, Los Alamitos CA, Nov. 2003. IEEE Computer Society Press.

[3] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings International Conference on Software Maintenance (ICSM 2003)*, pages 23–32, Los Alamitos CA, Sept. 2003. IEEE Computer Society Press.

[4] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[5] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings International Conference on Software Maintenance (ICSM '98)*, pages 190–198, Los Alamitos CA, 1998. IEEE Computer Society Press.

[6] H. Gall, M. Jazayeri, R. Klösch, and G. Trausmuth. Software Evolution Observations Based on Product Release History. In *Proceedings of the International Conference on Software Maintenance*, pages 160–166. IEEE Computer Society Press, 1997.

[7] H. Gall, J. Krajewski, and M. Jazayeri. CVS Release History Data for Detecting Logical Couplings. In *Proceedings of IWPSE 2003 (International Workshop on Principles of Software Evolution)*, pages 13–23, September 2003.

[8] T. Gîrba, S. Ducasse, and M. Lanza. Yesterday's Weather: Guiding early reverse engineering efforts by summarizing the evolution of changes. In *Proceedings 20th IEEE International Conference on Software Maintenance (ICSM'04)*, pages 40–49, Los Alamitos CA, 2004. IEEE Computer Society Press.

[9] T. Gîrba and M. Lanza. Visualizing and characterizing the evolution of class hierarchies. In *WOOR 2004 (5th ECOOP Workshop on Object-Oriented Reengineering)*, 2004.

[10] M. Jazayeri. On architectural stability and evolution. In *Reliable Software Technologies-Ada-Europe 2002*, pages 13–23, Berlin, 2002. Springer Verlag.

[11] M. Jazayeri, H. Gall, and C. Riva. Visualizing Software Release Histories: The Use of Color and Third Dimension. In *Proceedings of ICSM '99 (International Conference on Software Maintenance)*, pages 99–108. IEEE Computer Society Press, 1999.

[12] M. Lanza and S. Ducasse. Understanding software evolution using a combination of software visualization and software metrics. In *Proceedings of Langages et Modèles à Objets (LMO 2002)*, pages 135–149, Paris, 2002. Lavoisier.

[13] M. Lanza, S. Ducasse, H. Gall, and M. Pinzger. Codecrawler — an information visualization tool for program comprehension. In *Proceedings of ICSE 2005 (27th IEEE International Conference on Software Engineering)*, pages 672–673. ACM Press, 2005.

[14] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing multiple evolution metrics. In *Proceedings of SoftVis 2005 (2nd ACM Symposium on Software Visualization)*, pages 67–75, St. Louis, Missouri, USA, May 2005.