Workshop on Maintenance of Aspect Oriented Systems

Mario Luca Bernardi, Giuseppe A. Di Lucca
Department of Engineering
Research Centre on Software Technology
University of Sannio, Benevento, Italy
mlbernar/dilucca@unisannio.it

Mariano Ceccato
Fondazione Bruno Kessler
IRST Trento, Italy
ceccato@fbk.eu

Abstract

Aspect Oriented Programming (AOP) constructs introduce new kinds of relationships among traditional units, e.g. classes in Object Oriented (OO) code, and aspects due to the weaving among aspects' code fragments, such as advice or inter-type declarations, and the traditional system units. These relationships can be difficult to identify making hard and risky the maintenance operations because the impact of a change could be very difficult to evaluate. New methods, techniques and tools (or the adaptation of the existing ones) are needed to successfully face and overcome the new problems and difficulties that AOP constructs could pose to code maintenance.

This workshop intends to act as a forum for the presentation and discussion of new approaches to effectively support the maintenance of AO systems, and to promote joint researches and experimental studies about this topic.

1. Introduction

The Aspect Oriented (AO) paradigm has been proposed as a way to produce better modularized software systems that should be easier to understand. Aspect Oriented Programming (AOP) allows to better modularize and manage the crosscutting concerns by means of program units called "aspects". The code related to crosscutting concerns, encapsulated in aspects, is "woven" into the code of the traditional program units (e.g. classes in OO code) to build the overall system.

While there are several approaches to reengineer and evolve 'traditional' systems towards AO [5], there are still very few works on the maintenance of AO systems [1]. The maintenance of AO systems may pose new and different problems with respect to systems developed by traditional programming languages.

In the system resulting from weaving together aspects and classes, complex relationships among modules could lead to a significant modification of the structure and behaviour of the base system. Thus, due to the heavy and intrusive effects that AOP constructs have on the base code, it could be very difficult to evaluate the impact of a change on the whole system, as well as very hard to understand the side/ripple effects that a change could introduce.

Due to the growing adoption of the AO paradigm in software development, the maintenance of AO systems will become one of next challenges in software engineering. Software maintainers should be supported by adequate methods, techniques and tools to successfully face with and overcome the novel challenges that AOP constructs pose on maintenance. Some of these challenges are represented by these questions:

- are the current methods/techniques/tools proposed to maintain 'traditional' software systems (e.g. OO systems) adequate to maintain also AO systems?
- which, and in which way, AOP constructs could make maintenance tasks harder?
- how traditional maintenance approaches can be adapted to consider the new peculiarities of AOP?
- what are AOP "best practices" that actually help to develop more maintainable software systems?

New approaches (or the adaptation of the existing ones) taking into account the AOP specific features should be defined and adopted to effectively maintain AO systems.

Based on these issues, the workshop has the main aim of:

- making the software maintenance community more aware of the (novel) difficulties related to AO system maintenance;
- presenting and discussing proposals about the problems raising in the maintenance of AO systems, and how to effectively address them;
- acting as a forum for the promotion of joint

researches and experimental studies about the maintenance of AO systems.

2. Workshop's Main Topics

The following (not exhaustive list of) issues are addressed by maintenance approaches for traditional software system, but they should be reconsidered when coping with AO systems:

- *AO code analysis*: 'traditional' code analysis techniques need to be adapted to the peculiar features of AOP. Indeed, AOP features (such as implicit invocations and introductions) will affect the traditional ways the code is statically or dynamically analyzed. Existing techniques have to be modified to cover such new specific features;
- *AO system models*: new types of system models (or the adaptation of the existing ones) are required to represent aspects and their relationships and interactions with traditional software components. Proposals addressing how to model and represent these units and their interactions are needed to easily identify and analyze the novel kind of dependencies.
- *AO code representation forms*: the traditional forms used to represent the source code, either based on graph (e.g. Control flow graph, dependence graph, call graph, data flow graph) or using expressions (e.g. regular expression) should be adapted to take into account the way AO constructs can alter control flow, data flow and static code structure.
- *AOP specific quality models and metrics*: AOP constructs introduce new forms of coupling and cohesion that affect the quality of an AO system, such as comprehensibility, testability and maintainability. Specific metrics and quality models are needed to specifically evaluate the quality of an AO system and to allow an effective estimation of maintenance efforts.
- *Aspect mining*: in the last years several approaches have been proposed to identify code components participating in Crosscutting Concerns (CC) to be encapsulated into aspects [4], with the purpose of evolving traditional systems toward AO [5]. The quality of CC identification and their reengineering into AOP can affect the maintainability of the resulting AO system. Proposals about the quality and effectiveness of aspect mining techniques that support the evolution towards well design and maintainable AO systems are needed and welcome.

Some of these issues have been partially addressed by existing research. In particular, some representation forms of AOP code depicting them by graphs have been proposed mainly to support some testing tasks [7, 8], as well as some models of AO systems based on UML [3,6]. More effort has been spent on aspect mining approaches to migrate OO systems towards AO [4, 5].

The aim of the selected papers is to propose challenging ideas to provide some (initial) solutions to the identified issues and to stimulate the discussion, as well as to favor networking within participants.

3. Conclusions

The workshop aims to involve people both from academia and industry, and from both the AOP and software maintenance communities. Participants should be interested in presenting and debating on the identified topics, to collect insights useful to plan studies, researches and experiments by creating collaborations and networks of researchers.

This workshop is also intended to be the prosecution of the debate started in the maintenance community after the working session 'Comprehending Aspect-Oriented Programs: Challenges and Open Issues' held at the International Conference on Program Comprehension 2007 [2].

References

[1] M. L. Bernardi, and G. A. Di Lucca, "Modelling aspect and class interactions by an interprocedural aspect control flow graph", in Proc. of 23rd International Conference on Software Maintenance , IEEE C. S., Oct. 2007.

[2] G.A. Di Lucca, M. Smit, B. Fraser, E. Stroulia, J. Hoover, "Comprehending Aspect Oriented Programs: Challenges and Open Issues", in Proc. of 15th International Conference on Program Comprehension, IEEE C. S., June 2007.

[3] J. Evermann, "A Meta-Level Specification and Profile for AspectJ in UML", in Proc. of 10th International Workshop on Aspect-oriented modeling, ACM Press, March 2007.

[4] A. Kellens, K. Mens and P. Tonella, "A Survey of Automated Code-Level Aspect Mining Techniques", in Transactions on Aspect Oriented Software Development, Vol. 4, Springer-Verlag (2007).

[5] M. P. Monteiro and J. M. Fernandes, "Towards a catalog of aspect-oriented refactorings", in Proc. of 4th International Conference on Aspect-Oriented Software Development, pages 111-122, ACM Press, March 2005.

[6] D. Stein, S. Hanenberg, and R. Unland, "A UML-based aspect-oriented design notation for AspectJ", in Proc. of the International Conference on Aspect-Oriented Software Development, ACM Press, March 2002.

[7] W. Xu, and D. Xu, "State-based incremental testing of aspect-oriented programs", in Proc. of the 5th International Conference on Aspect-Oriented Software Development, ACM Press, March 2006

[8] J. Zhao, "Control-Flow Analysis and Representation for Aspect-Oriented Programs", in Proc. of 6th International Conference on Quality Software, IEEE C. S., Oct. 2006