7th Workshop on Object-Oriented Reengineering

Roel Wuyts[1], Serge Demeyer[2], Yann-Gaël Guéhéneuc[3],, Kim Mens[4], and Stéphane Ducasse[5]

[1] Département d'Informatique, Université Libre de Bruxelles — Belgium
[2] Department of Mathematics and Computer Science, University of Antwerp — Belgium
[3] Group of Open and Distributed Systems, Université de Montréal — Canada
[4] Département d'Ingénierie Informatique, Université catholique de Louvain — Belgium
[5] LISTIC Laboratory, University of Savoie — France

1 Introduction

The ability to reengineer object-oriented legacy systems has become a vital matter in today's software industry. Early adopters of the object-oriented programming paradigm are now facing the problems of transforming their object-oriented "legacy" systems into full-fledged frameworks.

To address this problem, a series of workshops have been organised to set up a forum for exchanging experiences, discussing solutions, and exploring new ideas. Typically, these workshops are organised as satellite events of major software engineering conferences, such as ECOOP'97 [1], ESEC/FSE'97 [3], ECOOP'98 [6], ECOOP'99 [5], ESEC/FSE'99 [4], ECOOP'03 [2], ECOOP'04 [7], , ECOOP'05 [8]. The last of this series so far has been organised in conjunction with ECOOP'06 and this report summarises the key discussions and outcome of that workshop.

As preparation to the workshop, participants were asked to submit a position paper which would help in steering the workshop discussions. As a result, we received seven position papers, of which nine authors were present during the workshop. Together with four organisers and four participants without position paper, the workshop numbered seventeen participants. The position papers, the list of participants, and other information about the workshop are available on the web-site of the workshop at http://smallwiki.unibe.ch/WOOR.

For the workshop itself, we chose a format that balanced presentation of position papers and time for discussions, using the morning for presentations of position papers and the afternoon for discussions in working groups.

Taking into account positive feedback from participants from the previous years, we decided to let each paper be presented by the authors of another paper. We have observed over the years that this system assures that more people read the position papers of one another, and invest time to study at least one other paper in detail. For the authors it is also interesting to hear what another researcher in the field has understood from the submission and the approach. As in previous years, this formated resulted in vivid discussions during the presentations, which formed a good foundation for the afternoon discussions.

Before the workshop, the workshop organisers (Serge Demeyer, Kim Mens, Roel Wuyts, and Stéphane Ducasse) classified the position papers in two groups, one group

on *Getting the Models* and one group on *Using Refactorings*. In the afternoon, the workshop participants separated in two working sessions, during which they could discuss and advance their ideas. The workshop was concluded with a plenary session where the results of the working groups were exposed and discussed in the larger group. Finally, we discussed practical issues, the most important one being the idea to organise a similar workshop next year which would be the 10th anniversary edition.

2 Summary of Position Papers

In preparation to the workshop, we received seven position papers (none of them was rejected), which naturally fitted into two categories: (a) *Getting the Models* and (b) *Using Refactorings*. The first category encompassed position papers that described tools and techniques to extract models from the source code or from dynamic information. The second category bundled papers that are dealing with refactoring of systems.

For each of these categories, we asked one reporter to summarise the position papers; summaries of the position papers are presented in the next two subsections.

2.1 Position Papers on *Getting the Models*

ModelExtractor: An Automatic Parametric Model Extractor by Régis Chevrel, Jean Bézivin, Hugo Brunelière, Albin Jossic, William Piers, and Frédéric Jouault.

The authors of the paper are working on a forthcoming model-driven reverse engineering development toolkit with an automatic model mining facility for systems developed in languages that have reflective capabilities. Their approach uses an annotated meta model, which is used to generate an extractor that is then capable of extracting the model from a given system. The authors have implemented a prototype for VB.Net, and plan to further extend this to be able to use parametric meta-models (instead of just the one that describes a simplified form of VB.Net) and to add dynamic information.

Program Comprehension and Design Pattern Detection: An Experience Report by Claudia Raibulet and Francesca Arcelli.

This paper reports on applying reverse engineering tools (primarily CodeCrawler and PTIDEJ, but also Fujaba) for understanding the large system *Java PathFinder*. The underlying motivation was the construction of an Eclipse plugin for the Java Pathfinder system, which therefore needed to be propoerly understood. The experiment showed that CodeCrawler scaled to the large system at hand, and was particularly useful to get an overview of the entire application and to guide further analysis steps. PTIDEJ was then used on interesting parts revealed by CodeCrawler in order to recognize design patterns and gain a more detailed understanding of these parts of the code. Fujaba was also used to do design pattern detection, but it gave too many false positives to be useful. The next conclusion was that the experiment was successful, since the system was better understood and the recovery of a number of software structures proved to be helpfull during all the plug-in development and in particular in the integration phase.

Difference Visualization: Impact Interaction between Code and Model by Susanne Jucknath-John and Sebastian Doltze.

When doing reverse engineering, it is sometimes needed to visualize the gap between a (desired) model and the existing system, for example when doing impact estimation. Current tools (like Rational Rose) have such functionality but simply dump this information in a (typically very large) text file. The papers presents two approaches to visualize the difference between a model and a system: (a) a quick, simple analysis and visualization based on a cross-table and (b) a more sophisticated visualization based on a matching between model-graph and code-graph. While the approach has a number of drawbacks (the decision to use sequence diagrams as model, and the fact that the tests were performed on open-source systems for which no models existed and first had to be extracted), it shows much promise and is definetely an improvement over just producing a big text file.

Reverse-engineering of UML 2.0 Sequence Diagrams from Execution Traces by Romain Delamare, Benoit Baudry, and Yves Le Traon.

To fully understand the behavior of a program, it is crucial to have efficient techniques to reverse dynamic views of the program. This paper focusses on the reverse engineering of UML 2.0 sequence diagrams showing loops and alternatives from execution traces. To build these complete sequence diagrams, the system's state is captured through dynamic analysis. The paper discusses the usage of state vectors (extracted from the trace), and how the state is captured in the first place. Using state vectors it can be detected when two different sequence diagrams are in the same state, or when there is an iteration on a single sequence diagram. The system is implemented as an adaptable trace analysis tool, that is also shown.

2.2 Position Papers on *Using Refactorings*

Correction of High-Level Design Defects with Refactorings by Naouel Moha, Saliha Bouden, and Yann-Gaël Guéhéneuc.

This paper defines design defects as ÒpoorÓ design solutions that hinder the maintenance of programs. Thus, their detection and correction are important to improve the maintainability and reduce the cost of maintenance. The detection of design defects has been actively investigated by the community. However, their correction stil l remains a problem to solve. This paper propose a first method to correct these defects systematically using refactorings. To this end a number of steps is described, and work is being carried out to fully support this steps through a language for specifying rules to correct design defects, an enriched meta-modelling tool able to properly capture design defects and being able to propose and apply the corrections.

Refactoring of Assertions in Design by Contract by Daniel Rodriguez, Manonjaran Satpathy, Josep Covas, and Juan-Jose Cuadrado.

Assertions are formal constraints over the state variables of a source program which are inserted as annotations in the program text. When some code has been annotated with assertions and is then subjected to refactoring, original assertions would no longer be consistent with the refactored code. The main focus of this paper is how to augment refactorings to become aware of assertions. Two of such extended refactorings are shown: *pull-up method* and *self encapsulate field*. Future work will consist in suport more refactorings and integrating them in tools, such as Eclipse.

Extending a Taxonomy of Bad Code Smells with Metrics by Raœl Marticorena, Carlos López, and Yania Crespo.

Bad smells are a usefull technique to describe code flaws, but are typically informal and need to be found manually, which can be very difficult to do in large systems that are not well-understood. Metrics exist that can be used to discover bad smells from code. Also taxonomies of bad smells have been created to group code smells. This paper presents an extended taxonomy for bad smells and links this with metrics. this is used to help a user choose one metric tool over another, which is shown on a concrete case.

3 Working Groups

Taking into account the presented position papers and the discussions that followed them, we decided to split up in two working groups for the afternoon. The working groups would focus on ...

The first working group discussed various issues that came up during the morning session. The second group discussed the question: *Are implicit design patterns useful in program comprehension?*.

3.1 Working Group 1

The first working group only consisted of four people that wanted to continue discussing a number of questions that came up during the morning session. Most questions had to do with the process of applying reengineering in a practical context and the continuing problem of lacking appropriate tools and integrating or comparing tools.

The discussions were open-ended, without clear answers. Given the fact that the information was in the discussions themselves (and not so much in the actual issues adressed), and that there no clear conclusions to be drawn, we refrain to include much detail here.

3.2 Working Group 2

The second group included 13 people with a common interest in sharing ideas and working out a solution to the question of the *usefulness of design patterns for program comprehension*. This question had already been prompted by several participants during the morning session (and other researchers in other forums) because several research states this usefulness as a premise for their study of the semi-automatic identification of design patterns in programs and models, but there is no (empirical) proof besides common sense that it is actually a valid premise.

Establishing an answer to this question is important because the participants feel that it could provide the strong basis to the expanding community working on the identification of design patterns (and other pattern-like artifacts).

However, giving an answer to this question is a non-trivial task because of (amongst others):

- The loose definition of design patterns.

– The lack of theory on program comprehension to assess the use of design patterns by software engineers during comprehension.
– The difficulties in performing experiments to assess the use of design patterns.

The participants to the workshop decided to work out together possible experiments to be carried out in parallel in their respective universities to attempt to provide an answer to this question. The more-than-four-hour long lively discussions among the participants allowed to identify the main problems in performing such experiments, including: the choice of the patterns to be used, the choice of the systems to be analysed, and the choice of the program comprehension task to be performed to initiate the experiment. Another main difficulty is identifying the right measures to assess the usefulness (or lack thereof) of design patterns.

Although this working group did not produce publishable result after its discussions, it has been agreed by all participants to keep on working together on this subject to develop an experimental process to answer the question, to apply this process in their universities, and to submit these results to an up-coming conference. A dedicated Web page has been setup for sharing work, at `http://cs.msi.vxu.se:9001/projects/dpm`. Results are expected before the next edition of WOOR.

4 Conclusion

In this report, we have listed the main ideas that were generated during the workshop on object-oriented reengineering. Based on a full day of fruitful work, we can make the following recommendations:

– *Viable Research Area.* Object-Oriented Reengineering remains an interesting research field with lots of problems to be solved and with plenty of possibilities to interact with other research communities. Therefore its vital that we organise such workshops outside of the traditional reengineering community (with conferences like ICSM, WCRE, CSMR, ...).
– *Research Community.* All participants agreed that it would be wise to organise a similar workshop at next year's ECOOP. There is an open invitation for everyone who wants to join in organising it: just contact the current organisers.
– *Workshop Format.* The workshop format, where some authors were invited to summarise position papers of others worked particularly well.

Next year will be the 10th anniversary edition of the Workshop on Object-Oriented Reengineering. We hope to welcome even more participants then, and will even try to provide a little surprise.

References

1. E. Casais, A. Jaasksi, and T. Lindner. FAMOOS workshop on object-oriented software evolution and re-engineering. In J. Bosch and S. Mitchell, editors, *Object-Oriented Technology (ECOOP'97 Workshop Reader)*, volume 1357 of *Lecture Notes in Computer Science*, pages 256–288. Springer-Verlag, Dec. 1997.

2. S. Demeyer, S. Ducasse, and K. Mens. Workshop on object-oriented re-engineering (WOOR'03). In F. Buschmann, A. P. Buchmann, and M. Cilia, editors, *Object-Oriented Technology (ECOOP'03 Workshop Reader)*, volume 3013 of *Lecture Notes in Computer Science*, pages 72–85. Springer-Verlag, July 2003.
3. S. Demeyer and H. Gall. Report: Workshop on object-oriented re-engineering (WOOR'97). *ACM SIGSOFT Software Engineering Notes*, 23(1):28–29, Jan. 1998.
4. S. Demeyer and H. Gall, editors. *Proceedings of the ESEC/FSE'99 Workshop on Object-Oriented Re-engineering (WOOR'99)*, TUV-1841-99-13. Technical University of Vienna - Information Systems Institute - Distributed Systems Group, Sept. 1999.
5. S. Ducasse and O. Ciupke. Experiences in object-oriented re-engineering. In A. Moreira and S. Demeyer, editors, *Object-Oriented Technology (ECOOP'99 Workshop Reader)*, volume 1743 of *Lecture Notes in Computer Science*, pages 164–183. Springer-Verlag, Dec. 1999.
6. S. Ducasse and J. Weisbrod. Experiences in object-oriented reengineering. In S. Demeyer and J. Bosch, editors, *Object-Oriented Technology (ECOOP'98 Workshop Reader)*, volume 1543 of *Lecture Notes in Computer Science*, pages 72–98. Springer-Verlag, Dec. 1998.
7. R. Wuyts, S. Ducasse, S. Demeyer, and K. Mens. Workshop on object-oriented re-engineering (WOOR'04). In J. Malenfant and B. M. Østvold, editors, *Object-Oriented Technology (ECOOP'04 Workshop Reader)*, volume 3344 of *Lecture Notes in Computer Science*, pages 177–186. Springer-Verlag, June 2004.
8. R. Wuyts, S. Ducasse, S. Demeyer, and K. Mens. Workshop on object-oriented re-engineering (WOOR'05). In *Object-Oriented Technology (ECOOP'05 Workshop Reader)*, Lecture Notes in Computer Science. Springer-Verlag, 2005.