Report of the Fifth International Workshop on Object-Oriented Reengineering

Roel Wuyts[1], Stéphane Ducasse[2], Serge Demeyer[3], and Kim Mens[4]

[1] Université Libre de Bruxelles, Brussels, Belgium
roel.wuyts@ulb.ac.be,
URL: http://homepages.ulb.ac.be/~rowuyts/
[2] University of Bern, Bern, Switzerland
ducasse@iam.unibe.ch,
URL: http://www.iam.unibe.ch/~ducasse/
[3] University of Antwerp, Antwerp, Belgium
URL: http://win-www.uia.ac.be/u/sdemey/
[4] Université catholique de Louvain, Louvain-la-Neuve, Belgium
kim.mens@info.ucl.ac.be,
URL: http://www.info.ucl.ac.be/ingidocs/people/km/research/KimResearch.html

Abstract. This paper reports on the results of the *Fifth International Workshop on Object-Oriented Reengineering* in Oslo on June 15, 2004. It enumerates the presentations made, classifies the contributions and lists the main results of the discussions held at the workshop. As such it provides the context for future workshops around this topic.

1 Objectives of the Workshop

The workshop on *Object-Oriented Reengineering* was co-located with the *18th European Conference on Object-Oriented Programming*, and took place at Oslo (Norway) on June 15, 2004. There were 13 participants, most of which contributed with a position paper that was reviewed and revised before the workshop.

The workshop gathered people working on solutions to reengineer object-oriented legacy systems, a vital matter in today's software industry. We claim that software evolution and reengineering is a key issue of software engineering, be it object-oriented or not. This shift of importance is starting to be noticed in research and industrial efforts.

The workshop builds upon the following important related achievements:

1. a series of workshops on *Object-Oriented Software Evolution* and on
2. Object-Oriented Software Re-engineering held at OOPSLA'96, ECOOP'97, ESEC'97 [1], ECOOP'98 [2], ECOOP'99 [3], WSR1999, WSR2000, WSR2001, ECOOP'2003 [4] and ETAPS2003.
3. The work done in the FAMOOS project (Framework-based Approach for Mastering Object-Oriented Software Evolution), carried out within the ESPRIT IV framework.

4. The Scientific Research Network on Foundations of Software Evolution funded by the Fund for Scientific Research — Flanders, Belgium[5] and the related Scientific Network "Research Links to Explore and Advance Software Evolution (RELEASE)" funded by the European Science Foundation Scientific Network[6].

2 Contributions

Eight papers were accepted in the workshop. This section gives an overview of the contributions (in no particular order)[7].

Opportunities and challenges in deriving metric impacts from refactoring postconditions by Bart DuBois (bart.dubois@ua.ac.be).

Refactoring transforming the source-code of an object-oriented program without changing its external observable behaviour is a restructuring process aimed at resolving evolution obstacles. Currently however, the efficiency of the refactor process in terms of quality improvements remains unclear. Such quality improvement can be expressed in terms of an impact on Object-Oriented metrics. The formalization of these metrics is based on the same constructs as refactoring postconditions. Therefore, using a uniform formalism, we can analytically derive how refactorings affect OO metrics. The result of these derivations are conditional impact descriptions, which specify under which conditions a refactoring improves or degrades a specific OO metric.

Based on these conditional impact descriptions, refactoring guidelines can be composed which focus time-investment only in those refactoring opportunities that will improve known indicators for specific quality indicators. Such qualitative feedback helps to steer the refactoring process, which is essential to make refactoring a technique for supporting maintenance. More empirical research will verify the practical usefulness of these guidelines from an external quality perspective.

Logic and Trace-based Object-Oriented Application Testing by Stéphane Ducasse (ducasse@iam.unibe.ch), Michael Freidig and Roel Wuyts (homepages.ulb.ac.be/~rowuyts/).)

Due to the size and the extreme complexity of legacy systems, it is nearly impossible to write from scratch tests before refactoring them. In addition object-oriented legacy systems present specific requirements to test them. Indeed late-binding allow subclasses to change fundamental aspects of the superclass code and in particular call flows. Moreover Object-oriented programming promotes a distribution of the responsibilities to multiple entities leading to complex scenario to be tested. In such a context one of the few trustable source of information is the execution of the application itself. Traditional forward engineering approaches

[5] `http://prog.vub.ac.be/FFSE/network.html`

[6] `http://labmol.di.fc.ul.pt/projects/release/`

[7] The full papers can be found on the workshop's website: `http://kilana.unibe.ch:9090/WOOR`

such as unit testing do not really provide adequate solution to this problem. Therefore there is a need for a more expressive way of testing the execution of object-oriented applications. We propose to represent the trace of object-oriented applications as logic facts and express tests over the trace. This way complex sequences of message exchanges, sequence matching, or expression of negative information are expressed in compact form. We validated our approach by implementing the prototype tool *TestLog*, which uses MethodWrappers [5] to reify an execution trace, and reasons on this using Soul [6], a logic programming language that can directly reason over Smalltalk objects, and can execute Smalltalk code. We used *TestLog* to test the Moose reengineering environment [7] and a meta-interpreter.

Visualizing and Characterizing the Evolution of Class Hierarchies by Tudor Girba (girba@iam.unibe.ch) and Michele Lanza (lanza@iam.unibe.ch).

Analyzing historical information can show how a software system evolved into its current state, but it can also show which parts of the system are more evolution prone. Yet, historical analysis implies processing a vast amount of information which makes the interpretation difficult. To address this issue, we introduce the notion of history of source code artifacts as a first class entity and define measurements which summarize the evolution of such entities. We then use these measurements to define polymetric views [8] for visualizing the effect of time on class hierarchies [9]. We show the application of our approach on one large open source case study and show how we can classify the class hierarchies based on their history.

Analyzing large event traces with the help of a coupling metrics by Andy Zaidman (Andy.Zaidman@ua.ac.be) and Serge Demeyer.

Gaining understanding of a large-scale industrial program is often a daunting task. In this context dynamic analysis has proven it's usefulness for gaining insight in object-oriented software. However, collecting and analyzing the event trace of large-scale industrial applications remains a difficult task. In this paper we present a heuristic that identifies interesting starting points for further exploratory program understanding. The technique we propose is based on a dynamic coupling metric, that measures interaction between runtime objects.

Reverse Engineering Aspectual Views using Formal Concept Analysis by Kim Mens (kim.mens@info.ucl.ac.be) and Tom Tourwé (Tom.Tourwe@cwi.nl).

In this position paper, we report on an initial experiment using the technique of formal concept analysis for reverse engineering aspectual views from object-oriented source code. An aspectual view is a set of source code entities, such as class hierarchies, classes and methods, that are structurally related in some way, and often crosscut a particular application. Initially, we follow a lightweight approach, where we only consider the names of classes and methods. This simplistic technique already results in the discovery of interesting and meaningful aspectual views, leaving us con- fident that more complex approaches will perform even better, and should be studied in the future.

Automatic renovation of Java programs using ReRAGs — examples and ideas by Torbjörn Ekman (torbjorn@cs.lth.se) and Görel Hedin (gorel@cs.lth.se).

When a language evolves and new features are added, it is usually desirable to renovate existing programs to make use of the new features. Some new features could make old programs illegal, and make renovation necessary. Other new features may allow old idioms to be replaced by clearer code. The evolution need not always concern the language as such. It could as well concern the evolution of standard frameworks, for example by adding new operations that should be used in favor over others that are deprecated.

This paper outlines how these kinds of problems can be handled using Rewritable Reference Attributed Grammars (ReRAGs), an object-oriented translation technique that we have recently developed [10]. In ReRAGs, a program is represented as an object-oriented abstract syntax tree (AST). Computations on the AST, e.g., to support name analysis, type checking, etc., are easy to express in ReRAGs, and can be used by conditional rewrite rules that can transfrom the AST to a suitable new form.

In the evolution of the Java programming language from 1.4 to 1.5, a number of new language constructs are added, allowing many programs to be simplified. This paper uses the new for-loop construct, that allows many iterations to be expressed in a simpler way, as a running example illustrate a renovation technique based on ReRAGs.

Constructing a Project Model and a Metadata Model for Experience Extraction by Hei-Chia Wang (hcwang@mail.ncku.edu.tw).

This paper proposes a project model and an experience metadata model to remember project experiences for software re-engineering. The experiences could be any product in the process of a project, such as plan documents, UML diagrams, and source code. In a software project, many phases and activities will be gone through and many artefacts will be generated. Properly organizing these resources for reuse can facilitate software re-engineering. However, in the past, the metadata schema for describing experience has not been defined and this made the resources difficult to store in a unified format for public searching and reusing.

The paper proposes a way of keeping all related project experiences in an experience web to solve this problem. The experience stored in this repository can then be retrieved and reused. Once an experience is reused, a feedback will be given as a new experience and the relation will be linked. Consequently, an experience web can be established for later users. The user can therefore get different views from previous works.

Towards an Aspectual Analysis of Legacy Systems by Bedir Tekinerdogan (B.Tekinerdogan@ewi.utwente.nl).

Aspect-Oriented software development provides explicit mechanisms for coping with concerns that crosscut many components and are tangled within individual components. Current AOSD approaches have primarily focused on coping with crosscutting concerns in software systems that are developed from scratch. In this paper we will investigate the applicability of AOSD to the evolution of legacy information systems. Various approaches have been already proposed to enhance LIS, however, these approaches have not explicitly considered cross-

cutting concerns and/or AOP techniques. We provide a categorization of legacy systems and give some early results in identifying and specifying aspects in legacy systems.

We also had two researchers that provided us with a position statement instead of a full paper:

1. *Jonne Itkonen's position statement* Jonne's interest in object-oriented reengineering comes from the software evolution side. It is interesting to see how the software develops over time, what kind of structures, patterns and anti-patterns emerge, when and why. Not only should this data and knowledge be used as input for reengineering practices, but to validate the choices made on forward engineering. He would like to see this to help to design better tools for software developers. Tools, that can be used in a way that Charles Rich and Richard C. Waters explained in their article "Programmer's Apprentice". That is, tools, that report gently to the developer the suspicious structures that have have lurked into the code, possibly suggesting alternative designs, possibly even learning from the choices made.
   We are confirming the discovered methods and metrics, and tools written, by applying them to an in-house developed study enrolment and management system called Korppi. It is a web-based system written mostly in Java, and it has been in development since 2001. As many different projects have written, developed and changed it over the years, it's code has started to show some unwanted features. This is why it gives as a pretty good example of the so called "real world code" to reengineer. The Korppi development team has also been a valuable asset when evaluating the design of the tools, and the applicability of the tools to real world software development situations.
2. *Isabel Michiels's position statement.* In the context of the ARRIBA research project, we are exploring techniques and tools that can support the integration of large-scale software entities that have not necessarily been designed to coexist. As a basis for these techniques, we are doing research about *code mining*, i.e. the extraction of knowledge out of large-scale, poorly or non documented software applications. Therefore we create abstract representations of these systems, and then use a logical engine for reasoning about and extracting knowledge out of the systems .
   Recently, we started exploring the use of aspect technology as instrumentation mechanism to explore and register the behavior of legacy systems in order to understand what these systems do and how their components relate to one another. By now we established a mapping of COBOL code to an XML representation of that code, and we are able to go back from that XML representation to COBOL. And we are using a declarative language as a medium to express aspects and pointcuts to instrument the XML representation of the COBOL code.

All this material was available to the participants of the workshop, and they were asked to familiarize themselves with each others' positions and research before the start of the workshop. This eased the discussions.

3 Presentations

The morning session was devoted to selected presentations, subdivided in groups. The presentations were chosen by the organisers because they had a higher potential for generating issues that would stimulate the discussions.

To structure the morning presentations we decided to group the submitted papers in three topics. Note that other groups could have been chosen, but these choices distributed the presentations quite evenly.

- Topic 1: Metrics. Several papers used metrics to attempt to characterize evolution or aid with refactoring.
  - *Visualizing and Characterizing the Evolution of Class Hierarchies* uses metrics obtained from the version information of a system to visualize evolution of a system.
  - *Analyzing large event traces with the help of a coupling metrics* introduces a metric that is useful to quickly get evolution information on large event traces.
  - *Opportunities and challenges in deriving metric impacts from refactoring postconditions* wants to derive the impact from a refactoring on source code by studying its pre- and postconditions, and as such determining whether it is useful to apply it.
- Topic 2: Program understanding. Since systems to be reengineering are typically unknown, some submissions discussed techniques to aid with program understanding.
  - *Reverse Engineering Aspectual Views using Formal Concept Analysis* discusses how the mathematical technique of formal concept analysis could be used to understand an existing system.
  - *Logic and Trace-based Object-Oriented Application Testing* uses logic queries on the execution trace to help express and test on patterns found in the execution of the software.
- Topic 3: Various. Last but not least we were left with two papers that we could not fit into meaningful groups because they differed from the other submissions and each other.
  - *Automatic renovation of Java programs using ReRAGs* shows how a rewriting system can be used to help code evolve, for example from one version of a language to another one.
  - *Constructing a Project Model and a Metadata Model for Experience Extraction* proposed a meta model to capture and use experience.

4 Discussion

The afternoon session was entirely devoted to discussions. A common decision was made to discuss about the following 5 topics in a single group:

Transformations The participants identified that there clearly is a need for a more structured approach to transforming code. A lot of the participants were interested in refactorings in this context, and identified the *need for pre- and postconditions of refactorings to become explicit*. If this were the case, refactorings could be compared across tools (or even across languages, up to a certain extent), and they could be kept track of as semantic entities (which would help the analysis of evolution histories). The participants doubted that is was realistic to formalize the transformations themselves. As a result from a comparison between the ReRag rewrite system presented in the morning session, and Soul (a logic meta-programming language that allows one to reason on Smalltalk and Java code) an overview of Soul was presented by R. Wuyts. Soul has, both as advantage and disadvantage, that it is a logic programming language, meaning that users need to be able to write logic programs, while ReRags uses Java (which results in slightly larger programs, but in the same language). Both approaches stress the fact that they are declarative as a key asset. When discussing unit tests, and their relation to code, the participants decided that it would be beneficial to*co-evolve unit tests and code*.

Dynamic Analysis Several techniques exploit dynamic information (typically gathered from code instrumentation) to analyse systems. The lack of a common interchange format was perceived as problematic by some participants, since it meant that different existing tools were not interoperable.

Abdelwahab Hamou-lhadj mentioned that he was working on CTF, a common dynamic information common exchange trace format, which resulted in a lot of discussion, especially regarding the meta information used to represent the dynamic information and the comparisons with other models. Important is that the model allows annotations of events (and hence that it is open), that it has built-in compression by representing the trace as an acyclic graph, and that it is streaming. As a result, *participants agreed that CTF could be very interesting*, but that more work needs to be invested to determine to see whether their current approaches could be expressable in the format. For example one participant needs to explictly keep the state of objects, which is currently not supported by CTF (it reifies objects by their identifier only).

Visualization Since in a lot of cases large data sets needs to be represented somehow, visualizing is an important topic that needed to be discussed. A first question raised was why not many approaches use 3D visualization approaches, add sound (some research plays back programs, allowing a trained developer to recognize bugs audibly), or employ multiple monitors. The general feeling was that *more research in human-computer interaction is necessary to meaningfully use such advanced visualization techniques*. In the absence of this research, current tools that use such techniques should take care to stay functional. A second question raised was why even simple visualization technology is not used more often in the context of reengineering. The participants indicated that *vi-*

sual overload (resulting in too much scrolling) and the lack of properly integrated tools hampered the broader usage of visualization.

Meta Modeling A fourth topic that was discussed extensively focussed on meta modeling, and first of all on the requirements for a meta model for reengineering. First of all it is important to note that for space reasons alone a meta model representation is preferred over an abstract syntax tree (AST)-based model. Experience by several independent partners also agreed on the fact that *a common language independent AST representation is extremely complicated to obtain.* Next a number of *advantages and disadvantages of three existing meta-models were discussed: DMM, GXL and TA.*

- DMM: is a model for representing the static part for different system and program artefacts. It does not capture the history. But, as mentioned by Ducasse, this is an orthogonal issue. While DMM is used widely, the absence of an API means that tool support is lacking.
- GXL: an XML-based model for storing source code as a graph. It has the disadvantage that it is extremely verbose and not very readable, but the advantage that it is standardized. But it also lacks support by tools (as put by one participant: there is a difference between talking and doing).
- TA (tupple attributes): a nicer model, but not used a lot.

Last but not least S. Ducasse noted that languages that languages that support introspection make it easier to write tools to extract meta models. However most of them (with Smalltalk and CLOS as notable exception) *forget to add access and invocations of fields and methods.* As a result one cannot ask 'who references this particular item', and these rules have to be coded by individual rules (which, depending on the scoping rules of the language, can become very difficult and error-prone).

Tools and Techniques This topic was concerned with what experience do people have with tools and techniques useful to support object-oriented software engineering. A decision was made to *add a webpage to the website for the workshop to let people list the tools they make or use.* Following tools will certainly be listed: MOOSE (a language-independent reengineering environment that supports program visualization and has support for evolution analysis), SourceNavigator (a stable tool to parse C++ and Java, and was recommended by some participants), RERags (a rewrite tool for Java), Soul (a logic meta-programming language that supports Smalltalk and Java).

5 Conclusion

The workshop brought together researchers working on object-oriented reengineering. Apart from lots of experiences and information that was exchanged on the workshop, the following bigger questions were raised:

- pre- and postconditions of refactorings need to become explicit.
- support is need to co-evolve unit tests and code.
- the CTF (Common Trace Format) model to interchange execution traces is interesting but needs more work.
- more research in human-computer interaction is necessary to meaningfully use all kinds of advanced visualization techniques.
- visual overload (resulting in too much scrolling) and the lack of properly integrated tools hamper the acceptance of visualization techniques.
- a common language independent AST representation is extremely complicated to obtain.

Tackling these problems will certainly result in interesting discussions to be held at the following workshop that is being planned.

About the Organizers

Prof. Stéphane Ducasse , from the University of Berne, is a member of the Software Composition Group headed by Prof. Oscar Nierstrasz.

Prof. Serge Demeyer is leading a research group investigating the theme of "Software Reengineering" (LORE - Lab On REengineering). They are the authors of the book "Object-Oriented Reengineering Patterns" published by Morgan Kaufman [11].

Prof. Kim Mens is one of the originators of the "reuse contract" technique for automatically detecting conflicts in evolving software [12] and is currently interested in the problem of "co-evolution" between source code and earlier life-cycle software artifacts [13].

Prof. Roel Wuyts bootstrapped research in co-evolution of design and implementation with the declarative meta-programming language Soul [14]. He focusses now on language symbiosis, program composition and programming language support for unanticipated evolution.

References

1. Demeyer, S., Gall, H., eds.: Proceedings of the ESEC/FSE Workshop on Object-Oriented Re-engineering. TUV-1841-97-10. Technical University of Vienna — Information Systems Institute — Distributed Systems Group (1997)
2. Stéphane, D., Weisbrod, J.: Report of the ecoop'98 workshop on experiences in object-oriented re-engineering (1998)
3. Ducasse, S., Ciupke, O., eds.: Proceedings of the ECOOP'99 Workshop on Experiences in Object-Oriented Re-Engineering, Forschungszentrum Informatik, Karlsruhe, Germany (1999) FZI 2-6-6/99.

4. Demeyer, S., Ducasse, S., Mens, K., Trifu, A., Vasa, R.: Report of the ecoop'03 workshop on object-oriented reengineering (2003)
5. Brant, J., Foote, B., Johnson, R., Roberts, D.: Wrappers to the Rescue. In: Proceedings ECOOP '98. Volume 1445 of LNCS., Springer-Verlag (1998) 396–417
6. Wuyts, R.: Declarative reasoning about the structure object-oriented systems. In: Proceedings of the TOOLS USA '98 Conference, IEEE Computer Society Press (1998) 112–124
7. Ducasse, S., Gîrba, T., Lanza, M.: Moose: a collaborative and extensible reengineering environment. In: Reengineering Environments. to be filled (2004)
8. Lanza, M., Ducasse, S.: Polymetric views — a lightweight visual approach to reverse engineering. IEEE Transactions on Software Engineering **29** (2003) 782–795
9. Gîrba, T., Ducasse, S., Lanza, M.: Yesterday's weather: Guiding early reverse engineering efforts by summarizing the evolution of changes. In: Proceedings of ICSM 2004 (International Conference on Software Maintenance). (2004)
10. Ekman, T., Hedin, G.: Rewritable Reference Attributed Grammars. In: Proceedings of ECOOP 2004. Volume 3086 of Lecture Notes in Computer Science., Springer-Verlag (2004)
11. Demeyer, S., Ducasse, S., Nierstrasz, O.: Object-Oriented Reengineering Patterns. Morgan Kaufmann (2002)
12. Lucas, C.: Documenting Reuse and Evolution with Reuse Contracts. PhD thesis, Programming Technology Lab, Vrije Universiteit Brussel, Brussels, Belgium (1997)
13. D'Hondt, T., De Volder, K., Mens, K., Wuyts, R.: Co-evolution of object-oriented software design and implementation. In: Proceedings of the international symposium on Software Architectures and Component Technology 2000. (2000)
14. Wuyts, R.: A Logic Meta-Programming Approach to Support the Co-Evolution of Object-Oriented Design and Implementation. PhD thesis, Vrije Universiteit Brussel (2001)