IN PARTNERSHIP WITH:
CNRS

Université des sciences et technologies de Lille (Lille 1)

Activity Report 2011

Project-Team RMOD

Analyses and Languages Constructs for Object-Oriented Application Evolution

IN COLLABORATION WITH: Laboratoire d'informatique fondamentale de Lille (LIFL)

RESEARCH CENTER
Lille - Nord Europe

THEME
Distributed Systems and Services

Table of contents

Project-Team RMOD

Keywords: Software Engineering, Software Evolution, Maintenance, Reflective Programming Languages, Dynamic Languages

1. Members

Research Scientists
Stéphane Ducasse [Team leader, Research Director (DR2) Inria, HdR]
Marcus Denker [Research Associate (CR2), Inria]

Faculty Members
Nicolas Anquetil [Associate Professor (MCF) USTL – IUT]
Damien Pollet [Associate Professor (MCF) USTL – Telecom Lille 1]

Technical Staff
Igor Stasenko [Expert Engineer]
Cyrille Delaunay [until Oct. 2011]
Andre Hora [Inria]

PhD Students
Jean-Baptiste Arnaud [Contrat doctoral Lille 1]
Camillo Bruni [Inria]
Gwenael Casaccio [Bourse Région]
Aaron Jimenez Govea [Universidad de Guadalajara, until Jun. 2011]
Jannik Laval [Inria, until Jul. 2011]
Nick Papoylias [Ecole des Mines de Douai, co-supervision]
Mariano Martinez-Peck [Ecole des Mines de Douai, co-supervision]
Veronica Uquillas-Gomez [Vrije Universiteit Brussel, co-tutelle]

Post-Doctoral Fellow
Usman Bhatti [PostDoc Inria]

Administrative Assistant
Marie-Bénédicte Dernoncourt [Secretary (SAR) Inria]

Others
Andy Kellens [Visiting Scientist – Vrije Universiteit Brussel, Belgium, Oct.–Dec. 2011]
Guido Chari [Internship, 3 months]
Javier Pinas [Internship, 3 months]
Benjamin Van Ryseghem [Internship, 3 months]

2. Overall Objectives

2.1. Introduction

Keywords: Software evolution, Maintenance, Program visualization, Program analyses, Meta modelling, Software metrics, Quality models, Object-oriented programming, Reflective programming, Traits, Dynamically typed languages, Smalltalk.

RMoD's general vision is defined in two objectives: remodularization and modularity constructs. These two views are the two faces of a same coin: maintenance could be eased with better engineering and analysis tools and programming language constructs could let programmers define more modular applications.

2.2. Reengineering and remodularization

While applications must evolve to meet new requirements, few approaches analyze the implications of their original structure (modules, packages, classes) and their transformation to support their evolution. Our research will focus on the *remodularization* of object-oriented applications. Automated approaches including clustering algorithms are not satisfactory because they often ignore user inputs. Our vision is that we need better approaches to support the transformation of existing software. The reengineering challenge tackled by RMoD is formulated as follows:

How to help remodularize existing software applications?

We are going to develop analyses and algorithms to remodularize object-oriented applications. This is why in a first period, we are going to study and build tools to support the *understanding of applications* at the level of packages and modules. This will allow us to understand the results of the *analyses* that we will build in a second period.

2.3. Constructs for modular and secure programming

Programming languages traditionally assume that the world is consistent. Although different parts of a complex system may only have access to restricted views of the system, the system as a whole is assumed to be globally consistent. Unfortunately, this means that unanticipated changes may have far-reached, harmful consequences for the global health of the system. With the pressure to have more secure systems, a modular system is the foundation to support secure applications. However, in the context of dynamic languages, there is a definitive tension between security (confidentiality and integrity) and language flexibility and openness. The language construct challenge tackled by RMoD is formulated as follows:

What are the language modularity constructs to support reuse and security?

We are going to continue our research effort on traits [1] by assessing trait scalability and reuse on a large case study and developing a pure trait-based language. In addition, some efforts will be dedicated to remodularizing a meta-level architecture in the context of the design of a secure dynamic language. Indeed at the extreme, modules and structural control of reflective features are the first steps towards flexible, dynamic, yet secure, languages. As a result, we expect to demonstrate that having adequate composable units and scoping units will help the evolution and recomposition of an application.

2.4. Highlights

- Four versions of Moose (our open-source reengineering platform) were released (4.2-4.5) (http://www.moosetechnology.org/).
- Veronica Uquillas-Gomez received the 2011 MoVES Most Promising Young Research Award during this year's MoVES Annual Event. MoVES (Modelling, Verification and Evolution of Software) is part of an Interuniversity Attraction Poles Programme funded by the Belgian State, Belgian Science Policy.
- Pharo 1.2 and 1.3 were released (http://www.pharo-project.org) with an accompanying book [36] (http://www.pharobyexample.org) translated in french and spanish. A japanese translation is under way.
- Damien Pollet's first peer-reviewed publication, co-authored during his masters in the Triskell group, has received the Ten Years Most Influential Paper Award at the Models 2011 conference in Wellington, NZ.
- Stéphane Ducasse got Distinguished Visiting Fellowship of the Royal Academy of Engineering.
- Fuel, a fast binary serializer, won the first prize at this year ESUG Innovation Technology Awards.

[1] Traits are groups of methods that can be composed orthogonally to simple inheritance. However contrary to mixin, the class has the control of the composition and conflict management.

- RMoD organized the CEA-EDF-Inria "Deep into Smalltalk" school in March. The school had over 40 participants and is available on Youtube (over 27 hours of tutorials).
- RMoD participated to the organization of the ESUG conference at Edinburg in August (150 participants).

3. Scientific Foundations

3.1. Software Reengineering

Strong coupling among the parts of an application severely hampers its evolution. Therefore, it is crucial to answer the following questions: How to support the substitution of certain parts while limiting the impact on others? How to identify reusable parts? How to modularize an object-oriented application?

Having good classes does not imply a good application layering, absence of cycles between packages and reuse of well-identified parts. Which notion of cohesion makes sense in presence of late-binding and frameworks? Indeed, frameworks define a context that can be extended by subclassing or composition: in this case, packages can have a low cohesion without being a problem for evolution. How to obtain algorithms that can be used on real cases? Which criteria should we select for a given remodularization?

We plan to enrich Moose, our reengineering environment, with a new set of analyses [51], [49]. We decompose our approach in three main and potentially overlapping steps:

1. Tools for understanding applications at large: packages/modules,
2. Remodularization analyses, and
3. Software Quality and Open DashBoard.

3.1.1. Tools for understanding applications at large: packages/modules

Context and Problems. As we are going to design and evaluate several algorithms and analyses to remodularize applications, we need a way to understand and assess the results we will obtain. Our experience on real application analyses taught us that analyses tend to produce a huge amount of data that we should understand and correlate to the original source code situation [50]. The problem is that understanding large systems is already difficult [102], [74], [76], [69], but in our case we need to understand an existing system *and* the results of our analysis. Parallelism between software programs and cities is commonly used to reason about evolution [69], [103]. While interesting, this metaphor does not scale because location of houses does not have any semantics information related to the connection between classes. A notion of radar has also been proposed [46], but this mechanism suffers from the same problem.

Therefore, there is a definitive need to have ways to support the understanding of large applications at the level of their structure.

Research Agenda. We are going to study the problems raised by the understanding of applications at the larger level of granularity such as packages/modules. We will develop a set of conceptual tools to support this understanding. These tools will certainly be visual such as the Distribution Map Software visualization [50] or based on the definition of new metrics taking into account the complexity of packages. Such a step is really crucial as a support for the remodularization analyses that we want to perform. The following tasks are currently ongoing:

MoQam. The Qualixo model has been originally implemented on top of the Java platform. An implementation of this model, named MoQam (Moose Quality Assessment Model), is under development in the Moose open-source and free reengineering environment. A first experiment has been conducted [70]. Exporters from Moose to the Squale software are under development.

Cohesion Metric Assessment. We are assessing the metrics and practices used originally in the Qualixo model. We are also compiling a number of metrics for cohesion and coupling assessment. We want to assess for each of these metrics their relevance in a software quality setting.

DSM. Dependency Structure Matrix (DSM), an approach developed in the context of process optimization, has been successfully applied to identify software dependencies among packages and subsystems. A number of algorithms helps organizing the matrix in a form that reflects the architecture and highlights patterns and problematic dependencies between subsystems. However, the existing DSM implementations often miss important information in their visualization to fully support a reengineering effort. We plan to enrich them to improve their usefulness to assess system general structure.

3.1.2. Remodularization analyses

Context and Problems. It is a well-known practice to layer applications with bottom layers being more stable than top layers [80]. Until now, few works have attempted to identify layers in practice: Mudpie [101] is a first cut at identifying cycles between packages as well as package groups potentially representing layers. DSM (dependency structure matrix) [100], [95] seems to be adapted for such a task but there is no serious empirical experience that validates this claim. From the side of remodularization algorithms, many were defined for procedural languages [68]. However, object-oriented programming languages bring some specific problems linked with late-binding and the fact that a package does not have to be systematically cohesive since it can be an extension of another one [104], [54].

Some approaches based on Formal Concept Analysis [99] show that such an analysis can be used to identify modules. However the presented example is small and not representative of real code. Other clustering algorithms [64], [65] have been proposed to identify modules [75], [86]. Once again, the specific characteristics of object-oriented programming are not taken into account. This is a challenge since object-oriented programming tends to scatter classes definitions over multiple packages and inheritance hierarchies. In addition, the existing algorithms or analyses often only work on toy applications. In the context of real applications, other constraints exist such as the least perturbation of the code, minimizing the hierarchy changes, paying attention to code ownership, layers, or library minimization. The approach will have to take into account these aspects.

Many different software metrics exist in the literature [73], [56], [61] such as the McCabe complexity metrics [81]. In the more specific case of object-oriented programming, assessing cohesion and coupling have been the focus of several metrics. However their success is rather questionable. For example, LCOM [43] has been highly criticized [55], [63], [62], [30], [39], [40]. Other approaches have been proposed such as RFC and CBO [43] to assess coupling between classes. However, many other metrics have not been the subject of careful analysis such as Data Abstraction Coupling (DAC) and Message Passing Coupling (MPC) [32], or some of metrics are not clearly specified (MCX, CCO, CCP, CRE) [73]. New cohesion measures were proposed [77], [89] taking into account class usage.

Research Agenda. We will work on the following items:

Characterization of "good" modularization. Any remodularization effort must use a quality function that allows the programmer to compare two possible decompositions of the system and choose which one represents a more desirable modularization. Remodularization consists in trying to maximize such a function. The typical function used by most researchers is some measure of cohesion/coupling. However, manual system modularization may rely on many different considerations: implemented functionalities, historical considerations, clients or markets served, ...We want to evaluate various modularization quality functions against existing modularizations to identify their respective strengths and weaknesses.

Cohesion and coupling metric evaluation and definition. Chidamber's well-known cohesion metric named LCOM has been strongly criticized [55], [63], [62], [40]. However, the solutions rarely take into account that a class is an incremental definition and, as such, can exist in a several packages at once. For example, LCOM* flattens inheritance to determine the cohesion of a class. In addition, these metrics are not adapted to packages. We will thus work on the assessment of existing cohesion metrics for classes, defining new ones if necessary for packages, and assess coupling metrics as well [30], [39]. This work is also related to the notion of software quality treated below.

Build an empirical validation of DSM and enhancements. We want to assess Dependency Structure Matrix (DSM) to support remodularization. DSM is good to identify cyclic dependencies. Now we want

to know if we can identify misplaced classes among groups of packages working as layers. For this purpose we will perform controlled experiments and in a second period apply DSM on one of the selected case studies. Based on these results, we will propose enhancements to the approach.

Layer identification. We want to propose an approach to identify layers based on a semi-automatic classification of package and class interrelationships that they contain. However, taking into account the wish or knowledge of the designer or maintainer should be supported. We will try to apply different algorithms and adapt them to the specific context of object-oriented programming [68].

3.1.3. Software Quality

Companies often look for the assessment of their software quality. Several models of software quality have been proposed: J.A. McCall [82] with his Factor-Criteria-Metrics has identified more than 50 candidate factors that may be used to assess software quality. Among those factors, only 11 were retained. Each of those has been characterized by 23 criteria that represent the internal project quality view. This approach is not easily used because of the high number of metrics —more than 300, some of which are not automatically computed. In an effort of conformance, the ISO (International Standardization Organization) and the IEC (International Electronical Commission) jointly defined the ISO 9126 norm in 1999. This norm, currently being restructured, will be composed of 4 parts: quality model (ISO 9126-1), external metrology (ISO 9126-2), Internal metrology (ISO 9126-3), Usage quality of metrology (ISO 9126-4). There is also a body of work focusing on design evaluation as quality criteria [83], [91], [87] and new quality models: QMOOD is for example a hierarchical quality model which proposes to link directly quality criteria to software metrics based on object-oriented software metrics [31] while other works focus on linking different high level criteria with software code metrics [78], [79].

Research Agenda. Since software quality is fuzzy by definition and a lot of parameters should be taken into account we consider that defining precisely a unique notion of software quality is definitively a Graal in the realm of software engineering. The question is still relevant and important. We plan to work on the two following items in the context of the Squale project in contact with the Qualixo company:

Quality Model. We want to study the existing quality models and develop in particular models that take into account (1) the possible overlaps in the source of information —it is important to know whether a model measures the same aspects several times, using different metrics (2) the combination of indicators —often, software quality models happily combine metrics, but at the price of losing the explicit relationships between the indicator contributions.

3.2. Language Constructs for Modular Design

While the previous axis focuses on how to help remodularizing existing software, this second research axis aims at providing new language constructs to build more flexible and recomposable software. We will build on our work on traits [97], [52] and classboxes [33] but also start to work on new areas such as security in dynamic languages. We will work on the following points: (1) Traits: behavioral units and (2) Modularization as a support for security.

3.2.1. Traits-based program reuse

Context and Problems. Inheritance is well-known and accepted as a mechanism for reuse in object-oriented languages. Unfortunately, due to the coarse granularity of inheritance, it may be difficult to decompose an application into an optimal class hierarchy that maximizes software reuse. Existing schemes based on single inheritance, multiple inheritance, or mixins, all pose numerous problems for reuse.

To overcome these problems, we designed a new composition mechanism called Traits [97], [52]. Traits are pure units of behavior that can be composed to form classes or other traits. The trait composition mechanism is an alternative to multiple or mixin inheritance in which the composer has full control over the trait composition. The result enables more reuse than single inheritance without introducing the drawbacks of multiple or mixin inheritance. Several extensions of the model have been proposed [48], [90], [34], [53] and several type systems were defined [57], [98], [92], [72].

Traits are reusable building blocks that can be explicitly composed to share methods across unrelated class hierarchies. In their original form, traits do not contain state and cannot express visibility control for methods. Two extensions, stateful traits and freezable traits, have been proposed to overcome these limitations. However, these extensions are complex and not simple to implement.

Research Agenda: Towards a pure trait language. We plan distinct actions: (1) a large application of traits, (2) assessment of the existing trait models and (3) bootstrapping a pure trait language.

- To evaluate the expressiveness of traits, some hierarchies were refactored, showing code reuse [37]. However, such large refactorings, while valuable, may not exhibit all possible composition problems, since the hierarchies were previously expressed using single inheritance and following certain patterns. We want to redesign from scratch the collection library of Smalltalk (or part of it). Such a redesign should on the one hand demonstrate the added value of traits on a real large and redesigned library and on the other hand foster new ideas for the bootstrapping of a pure trait-based language.

  In particular we want to reconsider the different models proposed (stateless [52], stateful [35], and freezable [53]) and their operators. We will compare these models by (1) implementing a trait-based collection hierarchy, (2) analyzing several existing applications that exhibit the need for traits —the Tweak UI library used in Sophie[2] and OpenCroquet[3], the Icalendar implementation of Seaside[4]. Traits may be flattened [88]. This is a fundamental property that confers to traits their simplicity and expressiveness over Eiffel's multiple inheritance. Keeping these aspects is one of our priority in forthcoming enhancements of traits.
- Alternative trait models. This work revisits the problem of adding state and visibility control to traits. Rather than extending the original trait model with additional operations, we use a fundamentally different approach by allowing traits to be lexically nested within other modules. This enables traits to express (shared) state and visibility control by hiding variables or methods in their lexical scope. Although the traits' "flattening property" no longer holds when they can be lexically nested, the combination of traits with lexical nesting results in a simple and more expressive trait model. We formally specify the operational semantics of this combination. Lexically nested traits are fully implemented in AmbientTalk, where they are used among others in the development of a Morphic-like UI framework.
- We want to evaluate how inheritance can be replaced by traits to form a new object model. For this purpose we will design a minimal reflective kernel, inspired first from ObjVlisp but with trait composition [45], then from Smalltalk [59].

3.2.2. Reconciling Dynamic Languages and Security

Context and Problems. More and more applications require dynamic behavior such as modification of their own execution (often implemented using reflective features [67]). For example, F-script allows one to script Cocoa Mac-OS X applications and Lua is used in Adobe Photoshop. Now in addition more and more applications are updated on the fly, potentially loading untrusted code or simply broken code. Bytecode checking and static code verification are used to enable security, however such approaches do not really work in presence of dynamic languages and reflective features. Therefore there is a tension between a need for flexibility and for security —here, by security we mean a mix between confidentiality and integrity.

Research Agenda: A secure dynamic and reflective language. To solve this tension, we will work on *Sure*, a language where security is provided by construction: as an example, if the language does not offer field access and its reflective facilities are controlled, then the possibility to access and modify private data is controlled. In this context, layering and modularizing the meta-level [38], as well as controlling the access to reflective features [41], [42] are important challenges. We plan to:

[2] http://www.sophieproject.org/
[3] http://www.opencroquet.org
[4] http://www.seaside.st

- Study the security abstractions available in erights[5] [85], [84], Java as well as classLoader strategies [71], [60].
- Categorize the different reflective features of languages such as CLOS [66], Python and Smalltalk [93] and identify suitable security mechanisms and infrastructure [58].
- Assess different security models (access rights, capabilities [94]...) and identify the ones adapted to our context as well as different access and right propagation.
- Define a language based on
  - the decomposition and restructuring of the reflective features [38],
  - the use encapsulation policies as a basis to restrict the interfaces of the controlled objects [96],
  - the definition of method modifiers to support privacy in the context of dynamic languages.

An open question is whether, instead of providing restricted interfaces, we could use traits to grant additional behavior to specific instances: without trait application, the instances would only exhibit default public behavior, but with additional traits applied, the instances would get extra behavior. We will develop *Sure*, a modular extension of the reflective kernel of Smalltalk (since it is one of the languages offering the largest set of reflective features such as pointer swapping, class changing, class definition...) [93].

4. Software

4.1. Moose

Participants: Stéphane Ducasse [correspondant], Usman Bhatti, Andre Hora, Nicolas Anquetil, Cyrille Delaunay, Jannik Laval, Tudor Gîrba [University of Bern].

Web: http://www.moosetechnology.org/

The platform. Moose is a language-independent environment for reverse- and re-engineering complex software systems. Moose provides a set of services including a common meta-model, metrics evaluation and visualization, a model repository, and generic GUI support for querying, browsing and grouping. The development of Moose began at the Software Composition Group in 1997, and is currently contributed to and used by researchers in at least seven European universities. Moose offers an extensible meta-described metamodel, a query engine, a metric engine and several visualizations. Moose is currently in its fourth release and comprises 55,000 lines of code in 700 classes.

The RMoD team is currently the main maintainer of the Moose platform. There are 200 publications (journal, international conferences, PhD theses) based on execution or use of the Moose environment.

The first version running on top of Pharo (Moose 4.0) was released in June 2010. In 2011, Moose saw five releases, with Moose 4.6 in beta since October 2011.
Here is the self-assessment of the team effort following the grid given at http://www.inria.fr/institut/organisation/instances/commission-d-evaluation.

- **(A5)** Audience : 5 – Moose is used by several research groups, a consulting company, and some companies using it in ad-hoc ways.
- **(SO4)** Software originality : 4 – Moose aggregates the last results of the teams that use it.
- **(SM3)** Software Maturity : 3 – Moose is developed since 1996 and got two main redesign phases.
- **(EM4)** Evolution and Maintenance : 4 – Moose will be used as a foundation of our start up so its maintenance is planned.
- **(SDL4)** Software Distribution and Licensing : 4 – BSD
- **(OC)** Own Contribution : (Design/Architecture)DA-4, (Coding/Debugging)-4,

[5] http://www.erights.org

(Maintenance/Support)-4, (Team/Project Management)-3

4.2. Pharo

Participants: Stéphane Ducasse, Marcus Denker [correspondant], Damien Pollet, Mariano Martinez-Peck, Veronica Uquillas-Gomez, Igor Stasenko.

Web: http://www.pharo-project.org/

The platform. Pharo is a new open-source Smalltalk-inspired language and environment. It provides a platform for innovative development both in industry and research. By providing a stable and small core system, excellent developer tools, and maintained releases, Pharo's goal is to be a platform to build and deploy mission critical Smalltalk applications.

The first stable version, Pharo 1.0, was released in 2010. The development of Pharo accelerated in 2011: Version 1.2 and 1.3 have been released, the development branch (1.4a) has seen already over 230 incremental releases as of mid November 2011. For 1.2 and 1.3, over 1000 bug tracker issues have been resolved. In 2011, the community organized five Pharo Sprints, RMoD organized the *Deep into Smalltalk* School in March 2011.

RMoD is the main maintainer and coordinator of Pharo. It is used widely in both research and industry. With Inria, RMoD is in the process of setting up a Pharo Consortium. There are 25 companies interested in supporting the consortium.
Here is the self-assessment of the team effort following the grid given at http://www.inria.fr/institut/organisation/instances/commission-d-evaluation.

- **(A5)** Audience: 5 – Used in many universities for teaching, more than 25 companies.
- **(SO3)** Software originality : 3 – Pharo offers a classical basis for some aspects (UI). It includes new frameworks and concepts compared to other implementations Smalltalk.
- **(SM4)** Software Maturity: 4 – Bug tracker, continuous integration, large test suite are on place.
- **(EM4)** Evolution and Maintenance: 4 – Active user group, consortium is being set up.
- **(SDL4)** Software Distribution and Licensing: 4 – Pharo is licensed under MIT.
- **(OC5)** Own Contribution: (Design/Architecture) DA-5, (Coding/Debugging) CD-5, (Maintenance/Support) MS-5, (Team/Project Management) TPM-5

4.3. Coral

Participants: Damien Pollet [correspondant], Camillo Bruni.

Web: http://rmod.lille.inria.fr/coral.

Coral extends the standard Pharo image, to integrate it into the host operating system shell environment and define system commands in Pharo. In term it will provide facilities for image preparation, configuration and deployment.

4.4. VerveineJ

Participants: Nicolas Anquetil [correspondant], Andre Hora.

Web: Inria project https://gforge.inria.fr/projects/verveinej/.

VerveineJ is a tool to export Java projects into the MSE format, which can then be imported inside Moose (see above). Although VerveineJ is not a research project in itself, it is an important building block for our research in that it allows us to run the Moose platform on legacy Java projects. Another similar tool, Infusion, already existed to fulfil the same needs, but it was closed sources and presented some errors that tainted the results we could obtain.

4.5. VerveineSharp

Participant: Usman Bhatti [correspondant].

Web: Inria project https://gforge.inria.fr/projects/verveinesharp/.

Similar to VervineJ (see above), VerveineSharp is a tool to export C# projects into the MSE format, which can then be imported inside Moose. The reasons for creating this project are the same as for VerveineJ: it is an important building block for our research in that it allows us to run the Moose platform on legacy C# projects. Because C# is a proprietary platform, there are no other tools that can give us the same functionality.

5. New Results

5.1. Package understanding and Assessing

Participants: Stéphane Ducasse, Nicolas Anquetil, Usman Bhatti, Jannik Laval.

To support the understanding of large systems is to offer ways to understand and fix dependencies between software elements. We worked on how to semi-automatically reorganize packages to minimize coupling.

Efficient Retrieval and Ranking of Undesired Package Cycles in Large Software Systems. Many design guidelines state that a software system architecture should avoid cycles between its packages. Yet such cycles appear again and again in many programs. We believe that the existing approaches for cycle detection are too coarse to assist the developers to remove cycles from their programs. We describe an efficient algorithm that performs a fine-grained analysis of the cycles among the packages of an application. In addition, we define a metric to rank cycles by their level of undesirability, prioritizing the cycles that seem the least desirable to the developers. Our approach is validated on two large and mature software systems in Java and Smalltalk. [19]

Legacy Software Restructuring: Analyzing a Concrete Case. Software re-modularization is an old preoccupation of reverse engineering research. The advantages of a well structured or modularized system are well known. Yet after so much time and efforts, the field seems unable to come up with solutions that make a clear difference in practice. Recently, some researchers started to question whether some basic assumptions of the field were not overrated. The main one consists in evaluating the high-cohesion/low-coupling dogma with metrics of unknown relevance. In this paper, we study a real structuring case (on the Eclipse platform) to try to better understand if (some) existing metrics would have helped the software engineers in the task. Results show that the cohesion and coupling metrics used in the experiment did not behave as expected and would probably not have helped the maintainers reach there goal. We also measured another possible restructuring which is to decrease the number of cyclic dependencies between modules. Again, the results did not meet expectations. [14]

An empirical model for continuous and weighted metric aggregation. It is now understood that software metrics alone are not enough to characterize software quality. To cope with this problem, most of advanced and/or industrially validated quality models aggregate software metrics: for example, cyclomatic complexity is combined with test coverage to stress the fact that it is more important to cover complex methods than accessors. Yet, aggregating and weighting metrics to produce quality indexes is a difficult task. Indeed certain weighting approaches may lead to abnormal situations where a developer increasing the quality of a software component sees the overall quality degrade. Finally, mapping combinations of metric values to quality indexes may be a problem when using thresholds. In this paper [20], we present the problems we faced when designing the Squale quality model, then we present an empirical solution based on weighted aggregations and on continuous functions. The solution has been termed the Squale quality model and validated over 4 years with two large multinational companies: Air France-KLM and PSA Peugeot-Citroen.

Modularization Metrics: Assessing Package Organization in Legacy Large Object-Oriented Software. In systems consisting of several thousands of classes, classes cannot be considered as units for software modularization. In such context, packages are not simply classes containers, but they also play the role of modules: a package should focus on providing well identified services to the rest of the software system. Therefore, understanding and assessing package organization is primordial for software maintenance tasks. Although there exist a lot of works proposing metrics for the quality of a single class and/or the quality of inter-class relationships, there exist few works dealing with some aspects for the quality of package organization and relationship. We believe that additional investigations are required for assessing package modularity aspects. The goal of these papers [13], [28] is to provide a complementary set of metrics that assess some modularity principles for packages in large legacy object-oriented software: Information-Hiding, Changeability and Reusability principles. Our metrics are defined with respect to object-oriented dependencies that are caused by inheritance and method call. We validate our metrics theoretically through a careful study of the mathematical properties of each metric.

5.2. Tools and Tool Infrastructure

Participants: Stéphane Ducasse, Veronica Uquillas-Gomez, Jannik Laval.

Reengineering large applications implies an underlying tool infrastructure that can scale and also be extended.

Ring: a Unifying Meta-Model and Infrastructure for Smalltalk Source Code Analysis Tools. Source code management systems record different versions of code. Tool support can then compute deltas between versions. To ease version history analysis we need adequate models to represent source code entities. As a first step to provide an infrastructure to support history analysis, this article [12] presents Ring, a unifying source code meta-model that can be used to support several activities and proposes a unified and layered approach to be the foundation for building an infrastructure for version and stream of change analyses. We re-implemented three tools based on Ring to show that it can be used as the underlying meta-model for remote and off-image browsing, scoping refactoring, and visualizing and analyzing changes. As a future work and based on Ring we will build a new generation of history analysis tools.

AspectMaps: A Scalable Visualization of Join Point Shadows. When using Aspect-Oriented Programming, it is sometimes difficult to determine at which join point an aspect executes. Similarly, when considering one join point, knowing which aspects will execute there and in what order is non- trivial. This makes it difficult to understand how the application will behave. A number of visualizations have been proposed that attempt to provide support for such program understanding. However, they neither scale up to large code bases nor scale down to understanding what happens at a single join point. In this paper [18], we present AspectMaps - a visualization that scales in both directions, thanks to a multi-level selective structural zoom. We show how the use of AspectMaps allows for program understanding of code with aspects, revealing both a wealth of information of what can happen at one particular join point as well as allowing to see the “big picture” on a larger code base. We demonstrate the usefulness of AspectMaps on an example and present the results of a small user study that shows that AspectMaps outperforms other aspect visualization tools.

Challenges to support automated random testing for dynamically typed languages. Automated random testing is a proved way to identify bugs and precondition violations, and this even in well tested libraries. In the context of statically typed languages, current automated random testing tools heavily take advantage of static method declaration (argument types, thrown exceptions) to constrain input domains while testing and to identify errors. For such reason, automated random testing has not been investigated in the context of dynamically typed languages. We present the key challenges that have to be addressed to support automated testing in dynamic languages. [17]

SmartGroups, Focusing on Task-Relevant Source Artifacts in IDEs. Navigating large software systems, even when using a modern IDE (Integrated Development Environment) is difficult, since conceptually related software artifacts are distributed in a huge software space. For most software maintenance tasks, only a small fraction of the entire software space is actually relevant. The IDE, however, does not reveal the task relevancy of source artifacts, thus developers cannot easily focus on the artifacts required to accomplish their tasks. Smart

Groups help developers to perform software maintenance tasks by representing groups of source artifacts that are relevant for the current task. Relevancy is determined by analyzing historical navigation and modification activities, evolutionary information, and runtime information. The prediction quality of Smart Groups is validated with a benchmark evaluation using recorded development activities and evolutionary information from versioning systems. [24]

5.3. Constructs for Dynamic Languages

Participants: Stéphane Ducasse, Marcus Denker, Veronica Uquillas-Gomez, Gwenael Casaccio, Camillo Bruni, Jean-Baptiste Arnaud, Damien Pollet.

To support our research on secure dynamic languages, we focused on improving language infrastructure.

Efficient Proxies in Smalltalk. A proxy object is a surrogate or placeholder that controls access to another target object. Proxy objects are a widely used solution for different scenarios such as remote method invocation, future objects, behavioral reflection, object databases, inter-languages communications and bindings, access control, lazy or parallel evaluation, security, among others. Most proxy implementations support proxies for regular objects but they are unable to create proxies for classes or methods. Proxies can be complex to install, have a significant overhead, be limited to certain type of classes, etc. Moreover, most proxy implementations are not stratified at all and there is no separation between proxies and handlers. We present Ghost, a uniform, light-weight and stratified general purpose proxy model and its Smalltalk implementation. Ghost supports proxies for classes or methods. When a proxy takes the place of a class it intercepts both, messages received by the class and lookup of methods for messages received by instances. Similarly, if a proxy takes the place of a method, then the method execution is intercepted too. [22]

Bootstrapping a Smalltalk. Smalltalk is a reflective system. It means that it is defined in itself in a causally connected way. Traditionally, Smalltalk systems evolved by modifying and cloning what is called an image (a chunk of memory containing all the objects at a given point in time). During the evolution of the system, objects representing it are modified. However, such an image modification and cloning poses several problems: (1) There is no operational machine-executable algorithm that allows one to build a system from scratch. A system object may be modified but it may be difficult to reproduce its exact state before the changes. Therefore it is difficult to get a reproducible process. (2) As a consequence, certain classes may not have been initialized since years. (3) Finally, since the system acts as a living system, it is not simple to evolve the kernel for introducing new abstractions without performing some kind of brain surgery on oneself. There is a need to have a step by step process to build Smalltalk kernels from scratch. After an analysis of past and current practices to mutate or generate kernels, we describe a kernel bootstrap process step-by-step. First the illusion of the existence of a kernel is created via stubs objects. Second the classes and meta-classes hierarchy are generated. Code is compiled and finally information needed by the virtual machine and execution are generated and installed. [15]

Flexible Object Layouts: Enabling Lightweight Language Extensions by Intercepting Slot Access. Programming idioms, design patterns and application libraries often introduce cumbersome and repetitive boilerplate code to a software system. Language extensions and external DSLs (domain specific languages) are sometimes introduced to reduce the need for boilerplate code, but they also complicate the system by introducing the need for language dialects and inter-language mediation.To address this, we propose to extend the structural reflective model of the language with object layouts, layout scopes and slots. Based on the new reflective language model we can 1) provide behavioral hooks to object layouts that are triggered when the fields of an object are accessed and 2) simplify the implementation of state-related language extensions such as stateful traits. By doing this we show how many idiomatic use cases that normally require boilerplate code can be more effectively supported.We present an implementation in Smalltalk, and illustrate its usage through a series of extended examples. [25]

5.4. Resources

Participants: Stéphane Ducasse, Marcus Denker, Mariano Martinez-Peck, Nick Papoylias.

Resource management is important in the context of resource constrained devices as well as situations were a large amount of data is modeled but not accessed often. One example for resource constrained devices are autonomous robots. An example for large models that are accessed infrequently are typical models of systems for software re-engineering.

With Ecole des Mines de Douai we explore how to analyze and improve memory in the case of unused data.

Problems and Challenges when Building a Manager for Unused Objects. Large object-oriented applications may occupy hundreds of megabytes or even gigabytes of memory. During program execution, a large graph of objects is created and constantly changed. Most object runtimes support some kind of automatic memory management based on garbage collectors (GC) whose idea is the automatic destruction of unreferenced objects. However, there are referenced objects which are not used for a long period of time or that are used just once. These are not garbage-collected because they are still reachable and might be used in the future. Due to these unused objects, applications use much more resources than they actually need. We present the challenges and possible approaches towards an unused object manager for Pharo. The goal is to use less memory by swapping out the unused objects to secondary memory and leaving in primary memory only those objects that are needed and used. When one of the unused objects is needed, it is brought back into primary memory. [23]

Clustered Serialization with Fuel. Serializing object graphs is an important activity since objects should be stored and reloaded on different environments. There is a plethora of frameworks to serialize objects based on recursive parsing of the object graphs. However such approaches are often too slow. Most approaches are limited in their provided features. For example, several serializers do not support class shape changes, global references, transient references or hooks to execute something before or after being stored or loaded. Moreover, to be faster, some serializers are not written taking into account the object-oriented paradigm and they are sometimes even implemented in the Virtual Machine hampering code portability. VM-based serializers such as ImageSegment are difficult to understand, maintain, and fix. For the final user, it means a serializer which is difficult to customize, adapt or extend to his own needs. We present a general purpose object graph serializer based on a pickling format and algorithm. We implement and validate this approach in the Pharo Smalltalk environment. We demonstrate that we can build a really fast serializer without specific VM support, with a clean object-oriented design, and providing most possible required features for a serializer. We show that our approach is faster that traditional serializers and compare favorably with ImageSegment as soon as serialized objects are not in isolation. [16]

Towards Structural Decomposition of Reflection with Mirrors Mirrors are meta-level entities introduced to decouple reflection from the base-level system. Current mirror-based systems focus on functional decomposition of reflection. We advocate that mirrors should also address structural decomposition. Mirrors should not only be the entry points of reflective behavior but also be the storage entities of meta-information. This decomposition can help resolve issues in terms of resource constraints (e.g. embedded systems and robotics) or security. Indeed, structural decomposition enables discarding meta-information. [21]

5.5. Empirical Studies in Software Product Line Engineering

Participant: Nicolas Anquetil.

Software Product Line Engineering (SPLE) is a new developpement paradigm that promises to offer faster develppement, with better quality. The idea is to develop a generic application (the Software Product Line) with pre-defined variation points. From this, new applications are derived from the generic application and the options are chosen for the possible variation points. Because it is still a new paradigm, Software Product Line developement is still an active research domain where empirical research is useful to check the validity of the results.

These publications are the results of an earlier research project to which N. Anquetil participated.

Managing information flow in the SPL development processes. Traceability is a quality attribute in software engineering that establishes the ability to describe and follow the life of a requirement in both the forward and backward directions (i.e. from its origins throughout its specification, implementation, deployment, use and maintenance, and vice-versa). The IEEE Standard Glossary of Software Engineering Terminology defines traceability as "the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another". According to (Palmer, 1997) "traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design, and implementation". Thus, trace relationships help in identifying the origin and rationale for artefacts generated during development lifecycle and the links between these artefacts. Identification of sources helps understanding requirements evolution and validating implementation of stakeholders' requirements. The main advantages of traceability are: (i) to relate software artefacts and design decisions taken during the software development cycle; (ii) to give feedback to architects and designers about the current state of the development, allowing them to reconsider alternative design decisions, and to track and understand bugs; and (iii) to ease communication between stakeholders. [26]

Empirical research in software product line engineering. Empirical evaluation has for many years been utilized to validate theories in other science disciplines. One of the first well-known reported examples of empirical evaluation occurred when Galileo wanted to prove that the rate of descent of objects was independent of their mass. This would disprove a theory put forward by Aristotle that the rate of descent is directly proportional to their weight. To prove his theory Galileo dropped two balls made from the same material but different masses from the top of the Tower of Pisa. When the experiment was performed Galileo's theory was proved correct through the empirical evidence collected. What this story demonstrates is the importance of empirical validation to verify or disprove theories and hypotheses. The purpose of this publication [27] is to emphasize the importance and difficulties of empirical evaluation in the domain of SPLE.

6. Contracts and Grants with Industry

6.1. Contracts with Industry

Pharo Consortium. We are in the process of setting up a Pharo Consortium. Over 25 companies are interested in participating. Inria supports the consortium with one full time engineer starting in 2011.

6.2. Grants with Industry

As the Pharo Consortium is not yet active, 2Denker UG (Cologne, Germany) gave a grant of 2500 EUR to RMoD to support the development of Pharo.

7. Partnerships and Cooperations

7.1. Regional Initiatives

We have signed a convention with team DIA led by Noury Bouraqadi of Ecole des Mines de Douai. In such context we co-supervized two PhD students (Mariano Martinez-Peck and Nick Papoylias). The team is also an important contributor and supporting organization of the Pharo project.

7.2. National Initiatives

7.2.1. Cutter ANR Project

Stéphane Ducasse [correspondant], Nicolas Anquetil.

Participants are RMoD and the D'Oc (M. Huchard)–APR(J.C. Koenig) groups at Lirmm. The aim of Cutter is to develop, combine, and evaluate new techniques for analyzing and modularizing code. The innovation of Cutter is to: (1) combine different package decomposition techniques (graph decomposition, program visualization...); (2) support different levels of abstractions (system, packages, classes); and (3) be directed by the quality of the resulting remodularization and take into account expert input.

7.2.2. Resilience FUI Project

Stéphane Ducasse [correspondant], Marcus Denker. Participants: Nexedi, Morphom Alcatel-Lucent Bell Labs, Astrium Geo Information, Wallix, XWiki, Alixen, Alterway, Institut Télécom, Université Paris 13, CEA LIST, Inria. Started in September 2011-September 2014.

RESILIENCE's goal is to protect private data on the cloud, to reduce spying and data loss in case of natural problems. RESILIENCE propose to develop a decentralized cloud architecture: SafeOS. Safe OS is based on replication of servers. In addition a safe solution for document should be developed. Sandboxing for Javascript applications should be explored.

7.3. European Initiatives

Participants: Stéphane Ducasse [correspondant], Veronica Uquillas Gomez, Marcus Denker.

7.3.1. IAP MoVES

Participant: Stéphane Ducasse [correspondant].

The Belgium IAP (Interuniversity Attraction Poles) MoVES (Fundamental Issues in Software Engineering: Modeling, Verification and Evolution of Software) is a project whose partners are the Belgium universities (VUB, KUL, UA, UCB, ULB, FUNDP, ULg, UMH) and three European institutes (Inria, IC and TUD) respectively from France, Great Britain and Netherlands. This consortium combines the leading Belgian research teams and their neighbors in software engineering, with recognized scientific excellence in MDE, software evolution, formal modeling and verification, and AOSD. The project focusses on the development, integration and extension of state-of-the-art languages, formalisms and techniques for modeling and verifying dependable software systems and supporting the evolution of Software-intensive systems. The project has started in January 2007 and is scheduled for a 60-months period. Read more at http://moves.vub.ac.be.

7.3.2. Réseau ERCIM Software Evolution

We are involved in the ERCIM Software Evolution working group since its inception. We participated at his creation when we were at the University of Bern.

7.4. International Initiatives

7.4.1. Inria Associate Teams

7.4.1.1. PLOMO

- Title: Customizable Tools and Infrastructure for Software Development and Maintenance
- Inria principal investigator: Stéphane Ducasse
- International Partner:
  - Institution: Universidad de Chile (Chile)
  - Laboratory: PLEIAD
- Duration: 2011 - 2013
- See also: http://pleiad.dcc.uchile.cl/research/plomo

Project Description

Software maintenance is the process of maintaining a software system by removing bugs, fixing performance issues and adapting it to keep it useful and competitive in an ever-changing environment [44]. Performing effective software maintenance and development is best achieved with effective tool support, provided by a variety of tools, each one presenting a specific kind of information supporting the task at hand [47]. The goal of PLOMO is to develop new meta tools to improve and bring synergy in the existing infrastructure of Pharo (for software development) and the Moose software analysis platform (for software maintenance).

PLOMO will (1) enhance the Opal open compiler infrastructure to support plugin definition, (2) offer an infrastructure for change and event tracking as well as model to compose and manipulate them, (3) work on a layered library of algorithms for the Mondrian visualization engine of Moose, (4) work on new ways of profiling applications. All the efforts will be performed on Pharo http://www.pharo-project.org and Moose http://www.moosetechnology.org/, two platforms heavily used by the RMoD and PLEIAD team.

The outcomes of PLOMO will include new research advances in the field of (i) bytecode generation for dynamic language; (ii) change and event tracking; (iii) software visualization engine; (iv) agile profiling framework. These four topics are recurrently considered by the most prestigious and competitive conferences (e.g., ECOOP, OOPSLA, FSE, ESEC, ICSE, TOOLS) and journals (e.g., TSE, TOPLAS, ASE), to which the participants of the PLOMO project are used to publish.

A strong focus on publishing our results in relevant scientific forum will remain a top priority. The artifacts produced by PLOMO will strongly reinforce the Pharo programming language and the Moose software analysis platform. The development and progress of Pharo is structured by RMoD, which has successfully created a strong and dynamic community. Moose is being used to realize consulting activities and it is used as a research platform in about 10 Universities, worldwide. We expect PLOMO to have a strong impact in both the software products and the communities structured around them.

Publications

1. S. Ducasse, M. Oriol, A. Bergel, Challenges to support automated random testing for dynamically typed languages [17]
2. J. Fabry, A. Kellens, S. Denier, S. Ducasse. AspectMaps: A Scalable Visualization of Join Point Shadows [18]
3. Romain Robbes, Johan Fabry, Marcus Denker, DIE: A Domain Specific Aspect Language for IDE Events, in submission

Research Visits

- Vanessa Pena and A. Bergel, Aug 15 until Aug 20. From Aug 20 until Aug 28 they attended ESUG 2011, a conference co-organized by RMoD.
- Romain Robbes from July 18 until July 24.
- Esteban Allende from July 19 until October 2. Esteban's stays is founded by the French Embassy in Chile. He received a grant of 3180 euros.
- Marcus Denker visited Chile Nov 7th-28th.

7.4.2. Visits of International Scientists

Dr. Andy Kellens from the VUB is visiting us during 3 months.

In the context of the PLOMO associated Team with the University of Chile we got three visitors over a period of one week (V. Pena, A. Bergel, R. Robbes). Esteban Allende, a PhD Student from Pleiad University of Chile, visited from July-Sept 2011.

7.4.2.1. Internships

RMoD hosted students for internships:

- Javier Pimas, University of Buenos Aires, Argentina, Sept.-Dec. 2011

- Guido Chari, University of Buenos Aires, Argentina, Sept.-Dec. 2011
- Cesar Couto, Federal University of Minas Gerais, Brazil, Dec. 2011-Fev. 2012 as part of the Pequi project (see 7.4.3.2)

7.4.3. Participation In International Programs

7.4.3.1. STICAmsud

This project focuses on software remodularization. Aspects, Traits and Classboxes are proved software mechanisms to provide modules in software applications. However, reenginering-based methodologies using these mechanisms have not yet been explored so far. This project intends to show how visualization and clustering techniques (such as Formal Concept Analysis) are useful to cope with the comprehension and transformation of module-based applications to applications which could use these mechanisms (i.e. aspects, traits, classboxes). The research results will be applied in a common reenginering platform MOOSE to show the applicability of the concepts.

CoReA spans three research institutions: Inria (the Lille Nord Europe research center, France), University of Chile (Santiago, Chile), LIFIA - Universidad Nacional de La Plata (La Plata, Argentina). The three national project leaders are Dr. Gabriela Arévalo (LIFIA - UNLP), Dr. Alexandre Bergel (Inria), Prof. Dr. Johan Fabry (University of Chile). The international coordinator is Dr. Alexandre Bergel. Participants are: Prof. Dr. Eric Tanter (University of Chile), and Dr. Stéphane Ducasse (senior scientist at Inria).

Marcus Denker visited Argentina November 3rd to November 5th, 2011.

7.4.3.2. Project Pequi – Inria/CNPq Brésil

The Pequi project is a collaboration between Professor Marco T. Valente's team at the Federal University of Minas Gerais in Brazil and the RMoD team. It focuses in producing Metrics, Techniques, and Tools for Software Remodularization.

It is recognized that software systems must be continuously maintained and evolved to remain useful. However, ongoing maintenance over the years contributes to degrade the quality of a system. Thus reengineering activities, including remodularization activities, are necessary to restore or enhance the maintainability of the systems. To help in the remodularization of software systems, the project will be structured in two main research lines in which both teams have experience and participation: (i) Evaluation and Characterization of Metrics for Software Remodularization; and (ii) Tools and Techniques for Removal of Architectural Violations.

The project started in July 2011 with a visit of Dr. Nicolas Anquetil to the brazilian team. The project will last 24 months.

8. Dissemination

8.1. Animation of the scientific community

8.1.1. Examination Committees

Stéphane Ducasse was in the examination committee of the following PhD theses:

- *Package Dependencies Analysis and Remediation in Object-Oriented Systems*, Jannik Laval, University of Lille-1, France. 17/06/11 (advisor).
- *Analyse et optimisation de patterns de code*, Rim CHAABANE, University Paris VIII, France. 14/09/2011 (reporter)

Nicolas Anquetil was in the examination committee of the following PhD theses:

- *Package Dependencies Analysis and Remediation in Object-Oriented Systems*, Jannik Laval, University of Lille-1, France. 17/06/11 (co-advisor).
- *Analyse et optimisation de patterns de code*, Rim CHAABANE, University Paris VIII, France. 14/09/2011

8.1.2. Scientific Community Animation

Stéphane Ducasse has been/is :

- Associate Editor of JOT (Journal of Object Technology).
- Reviewer for the following journals:
  - IEEE Transactions on Software Engineering (TSE),
  - ACM Transactions on Software Engineering and Methodology (TOSEM),
  - Journal on Software Maintenance and Evolution (JSME),
  - Journal of Systems and Software (JSS),
  - IEEE Software,
  - International Journal on Information and Software Technology (IST)
- Member of the following committees:
  - International Conference on Software Maintenance (ICSM).
  - International Conference on Objects, Models, Components, Patterns (TOOLS).
  - Program Chair of the International Smalltalk Conference.
  - IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 11)

Nicolas Anquetil was :

- Reviewer for the following journals:
  - Journal of Systems and Software (JSS);
  - Software Quality journal (SQJ);
  - Empirical Software Engineering (EMSE);
  - Journal of Object Technology (JOT);
  - Journal of the Brazilian Computer Society (JBCS).
- Member of the following committees:
  - International Conference on Software Maintenance (ICSM).
  - European Conference on Software Maintenance and Reengineering (CSMR).
- Co-organizer for the *Journées nationales IDM, CAL, and GDR/GPL* (June 2011).

Marcus Denker has been/is :

- PC Member ICSE 2012 Tool Demonstration (International Conference on Software Engineering)
- PC Member Varicomp 2011 (2nd International Workshop on Variability and Composition)
- PC Member AOSD 2011 (9th Annual Aspect-Oriented Software Development Conference)
- PC Member RAM-SE 2011 (ECOOP Workshop on Reflection, AOP, and Meta-Data for Software Evolution)
- PC Member IWST 2011 (ESUG International Smalltalk Workshop)
- Reviewer TOOLS 2011 (International Conference Objects, Models, Components, Patterns)

Damien Pollet has been/is :

- Co-organizer for the *Journées nationales IDM, CAL, and GDR/GPL* (June 2011).
- Maintainer of the LaTeX style files for submissions to the Journal of Object Technology.

8.2. Teaching

Marcus Denker gave a lecture about Reflection and Context at Universite catholique de Louvain, March 2011.

Stéphane Ducasse teaches a course on advanced OO design in the Master of IMUS (University of Savoie) and a course on metamodeling and reflective systems at Ecole des Mines de Douai.

Damien Pollet organizes and teaches bachelor and master-level courses on algorithms, programming in C and Java, object-oriented design, remote objects and web applications at the engineering school Telecom Lille 1.

Nicolas Anquetil teaches at the IUT (Institut Universitaire Technologique) of Lille 1, courses on Graphical User Interface programming with Java and Software Quality and Tests.

Jean-Baptiste Arnaud and **Jannik Laval** are teaching assistants (moniteur) at the IUT.

9. Bibliography

Major publications by the team in recent years

[1] N. ANQUETIL, K. M. DE OLIVEIRA, K. D. DE SOUSA, M. G. BATISTA DIAS. *Software maintenance seen as a knowledge management issue*, in "Inf. Softw. Technol.", 2007, vol. 49, n° 5, p. 515–529, http://dx.doi.org/10.1016/j.infsof.2006.07.007.

[2] N. ANQUETIL, T. LETHBRIDGE. *Comparative study of clustering algorithms and abstract representations for software remodularization*, in "IEE Proceedings - Software", 2003, vol. 150, n° 3, p. 185-201.

[3] M. DENKER, S. DUCASSE, É. TANTER. *Runtime Bytecode Transformation for Smalltalk*, in "Journal of Computer Languages, Systems and Structures", July 2006, vol. 32, n° 2-3, p. 125–139 [*DOI :* 10.1016/J.CL.2005.10.002], http://scg.unibe.ch/archive/papers/Denk06aRuntimeByteCodeESUGJournal.pdf.

[4] S. DUCASSE, T. GÎRBA, A. KUHN, L. RENGGLI. *Meta-Environment and Executable Meta-Language using Smalltalk: an Experience Report*, in "Journal of Software and Systems Modeling (SOSYM)", February 2009, vol. 8, n° 1, p. 5–19 [*DOI :* 10.1007/S10270-008-0081-4], http://scg.unibe.ch/archive/drafts/Duca08a-Sosym-ExecutableMetaLanguage.pdf.

[5] S. DUCASSE, M. LANZA. *The Class Blueprint: Visually Supporting the Understanding of Classes*, in "Transactions on Software Engineering (TSE)", January 2005, vol. 31, n° 1, p. 75–90 [*DOI :* 10.1109/TSE.2005.14], http://scg.unibe.ch/archive/papers/Duca05bTSEClassBlueprint.pdf.

[6] S. DUCASSE, A. LIENHARD, L. RENGGLI. *Seaside: A Flexible Environment for Building Dynamic Web Applications*, in "IEEE Software", 2007, vol. 24, n° 5, p. 56–63 [*DOI :* 10.1109/MS.2007.144], http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4302687.

[7] S. DUCASSE, O. NIERSTRASZ, N. SCHÄRLI, R. WUYTS, A. P. BLACK. *Traits: A Mechanism for fine-grained Reuse*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", March 2006, vol. 28, n° 2, p. 331–388 [*DOI :* 10.1145/1119479.1119483], http://scg.unibe.ch/archive/papers/Duca06bTOPLASTraits.pdf.

[8] S. DUCASSE, D. POLLET. *Software Architecture Reconstruction: A Process-Oriented Taxonomy*, in "IEEE Transactions on Software Engineering", July 2009, vol. 35, n^{o} 4, p. 573-591 [*DOI :* 10.1109/TSE.2009.19], http://scg.unibe.ch/archive/external/Duca09x-SOAArchitectureExtraction.pdf.

[9] S. DUCASSE, D. POLLET, M. SUEN, H. ABDEEN, I. ALLOUI. *Package Surface Blueprints: Visually Supporting the Understanding of Package Relationships*, in "ICSM '07: Proceedings of the IEEE International Conference on Software Maintenance", 2007, p. 94–103, http://scg.unibe.ch/archive/papers/Duca07cPackageBlueprintICSM2007.pdf.

[10] N. SCHÄRLI, A. P. BLACK, S. DUCASSE. *Object-oriented Encapsulation for Dynamically Typed Languages*, in "Proceedings of 18th International Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'04)", October 2004, p. 130–149 [*DOI :* 10.1145/1028976.1028988], http://scg.unibe.ch/archive/papers/Scha04bOOEncapsulation.pdf.

Publications of the year

Doctoral Dissertations and Habilitation Theses

[11] J. LAVAL. *Package Dependencies Analysis and Remediation in Object-Oriented Systems*, Université de Lille, 2011, http://rmod.lille.inria.fr/archives/phd/PhD-2011-Laval.pdf.

Articles in International Peer-Reviewed Journal

[12] V. UQUILLAS GÓMEZ, S. DUCASSE, T. D'HONDT. *Ring: a Unifying Meta-Model and Infrastructure for Smalltalk Source Code Analysis Tools*, in "Computer Languages, Systems and Structures", 2011, http://rmod.lille.inria.fr/archives/papers/Uqui11a-RingJournalPaper-CSSJournal.pdf.

International Conferences with Proceedings

[13] H. ABDEEN, S. DUCASSE, H. A. SAHRAOUI. *Modularization Metrics: Assessing Package Organization in Legacy Large Object-Oriented Software*, in "International Working Conference on Reverse Engineering (WCRE'11)", Washington, DC, USA, IEEE Computer Society Press, 2011, http://hal.inria.fr/docs/00/61/45/83/PDF/ModularizationMetrics-INRIA.pdf.

[14] N. ANQUETIL, J. LAVAL. *Legacy Software Restructuring: Analyzing a Concrete Case*, in "Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR'11)", Oldenburg, Germany, 2011, http://rmod.lille.inria.fr/archives/papers/Anqu11a-CSMR2011-Coupling.pdf.

[15] G. CASACCIO, S. DUCASSE, L. FABRESSE, J.-B. ARNAUD, B. VAN RYSEGHEM. *Bootstrapping a Smalltalk*, in "Proceedings of Smalltalks 2011 International Workshop", Bernal, Buenos Aires, Argentina, 2011, http://rmod.lille.inria.fr/archives/workshops/Casa11a-Smalltalks-BootstrappingASmalltalk.pdf.

[16] M. DIAS, M. M. PECK, S. DUCASSE, G. ARÉVALO. *Clustered Serialization with Fuel*, in "Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)", Edinburgh, Scotland, 2011, http://rmod.lille.inria.fr/archives/workshops/Dia11a-IWST11-Fuel.pdf.

[17] S. DUCASSE, M. ORIOL, A. BERGEL. *Challenges to support automated random testing for dynamically typed languages*, in "Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)", Edinburgh, Scotland, 2011, http://rmod.lille.inria.fr/archives/workshops/Duca11a-IWST11-RandomTesting.

[18] J. FABRY, A. KELLENS, S. DENIER, S. DUCASSE. *AspectMaps: A Scalable Visualization of Join Point Shadows*, in "International Conference on Program Comprehension (ICPC)", IEEE Computer Society Press, 2011, p. 121-130, http://rmod.lille.inria.fr/archives/papers/Fabr11a-ICPC2011-AspectMaps.pdf.

[19] J. R. FALLERI, S. DENIER, J. LAVAL, P. VISMARA, S. DUCASSE. *Efficient Retrieval and Ranking of Undesired Package Cycles in Large Software Systems*, in "Proceedings of the 49th International Conference on Objects, Models, Components, Patterns (TOOLS'11)", Zurich, Switzerland, June 2011, http://rmod.lille.inria.fr/archives/papers/Fall11a-Tools2011-UndesirableCycles.pdf.

[20] K. MORDAL-MANET, J. LAVAL, S. DUCASSE, N. ANQUETIL, F. BALMAS, F. BELLINGARD, L. BOUHIER, P. VAILLERGUES, T. J. MCCABE. *An empirical model for continuous and weighted metric aggregation*, in "CSMR 2011: Proceedings of the 15th European Conference on Software Maintenance and Reengineering", Oldenburg, Germany, 2011, http://rmod.lille.inria.fr/archives/papers/Mord11a-CSMR2011-Squale.pdf.

[21] P. NIKOLAOS, N. BOURAQADI, M. DENKER, S. DUCASSE, L. FABRESSE. *Towards Structural Decomposition of Reflection with Mirrors*, in "Proceedings of International Workshop on Smalltalk Technologies (IWST 2011)", Edingburgh, United Kingdom, 2011, http://hal.inria.fr/inria-00629175/en/.

[22] M. M. PECK, N. BOURAQADI, M. DENKER, S. DUCASSE, L. FABRESSE. *Efficient Proxies in Smalltalk*, in "Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2011)", Edinburgh, Scotland, 2011, http://rmod.lille.inria.fr/archives/workshops/Mart11a-IWST11-Ghost.pdf.

[23] M. M. PECK, N. BOURAQADI, M. DENKER, S. DUCASSE, L. FABRESSE. *Problems and Challenges when Building a Manager for Unused Objects*, in "Proceedings of Smalltalks 2011 International Workshop", Bernal, Buenos Aires, Argentina, 2011, http://rmod.lille.inria.fr/archives/workshops/Mart11b-Smalltalks2011-UOM.pdf.

[24] D. ROTHLISBERGER, O. NIERSTRASZ, S. DUCASSE. *SmartGroups: Focusing on Task-Relevant Source Artifacts in IDEs*, in "International Conference on Program Comprehension (ICPC)", IEEE Computer Society Press, 2011, http://rmod.lille.inria.fr/archives/papers/Roet11a-ICPC2011-smartGroups.pdf.

[25] T. VERWAEST, C. BRUNI, M. LUNGU, O. NIERSTRASZ. *Flexible object layouts: enabling lightweight language extensions by intercepting slot access*, in "Proceedings of 26th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '11)", New York, NY, USA, ACM, 2011, p. 959–972 [*DOI :* 10.1145/2048066.2048138], http://rmod.lille.inria.fr/archives/papers/Verw11a-OOSPLA11-FlexibleObjectLayouts.pdf.

Scientific Books (or Scientific Book chapters)

[26] N. ANQUETIL, U. KULESZA, R. MATEUS, R. MITSCHKE, A. MOREIRA, J.-C. ROYER, A. RUMMLER. *Managing information flow in the SPL development processes*, in "Aspect-Oriented, Model-Driven Software Product Lines – The AMPLE way", A. RASHID, J.-C. ROYER, A. RUMMLER (editors), Cambridge University Press, 2011, chap. 8, p. 222–262.

[27] P. GREENWOOD, V. ALVES, J. HUTCHINSON, C. SCHWANNINGER, N. ANQUETIL. *Empirical research in software product line engineering*, in "Aspect-Oriented, Model-Driven Software Product Lines – The AMPLE way", A. RASHID, J.-C. ROYER, A. RUMMLER (editors), Cambridge University Press, 2011, chap. 14, p. 411–443.

Research Reports

[28] H. ABDEEN, S. DUCASSE, H. A. SAHRAOUI. *Modularization Metrics: Assessing Package Organization in Legacy Large Object-Oriented Software*, RMod – INRIA Lille-Nord Europe, 2011, http://rmod.lille.inria.fr/archives/reports/Abde11a-TechReport-ModularizationMetrics-INRIA.pdf.

Scientific Popularization

[29] O. AUVERLOT, S. DUCASSE. *Construire un service Rest avec Pharo et Seaside-Rest*, in "Linux Magazine", September 2011, vol. 1, n° 141.

References in notes

[30] E. ARISHOLM, L. C. BRIAND, A. FOYEN. *Dynamic Coupling Measurement for Object-Oriented Software*, in "IEEE Transactions on Software Engineering", 2004, vol. 30, n° 8, p. 491–506, http://csdl.computer.org/comp/trans/ts/2004/08/e0491abs.htm.

[31] J. BANSIYA, C. DAVIS. *A Hierarchical Model for Object-Oriented Design Quality Assessment*, in "IEEE Transactions on Software Engineering", January 2002, vol. 28, n° 1, p. 4–17.

[32] J. BANSIYA, L. ETZKORN, C. DAVIS, W. LI. *A Class Cohesion Metric for Object-Oriented Designs*, in "Journal of Object-Oriented Programming", January 1999, vol. 11, n° 8, p. 47–52.

[33] A. BERGEL, S. DUCASSE, O. NIERSTRASZ. *Classbox/J: Controlling the Scope of Change in Java*, in "Proceedings of 20th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05)", New York, NY, USA, ACM Press, 2005, p. 177–189 [*DOI :* 10.1145/1094811.1094826], http://scg.unibe.ch/archive/papers/Berg05bclassboxjOOPSLA.pdf.

[34] A. BERGEL, S. DUCASSE, O. NIERSTRASZ, R. WUYTS. *Stateful Traits*, in "Advances in Smalltalk — Proceedings of 14th International Smalltalk Conference (ISC 2006)", LNCS, Springer, August 2007, vol. 4406, p. 66–90, http://dx.doi.org/10.1007/978-3-540-71836-9_3.

[35] A. BERGEL, S. DUCASSE, O. NIERSTRASZ, R. WUYTS. *Stateful Traits and their Formalization*, in "Journal of Computer Languages, Systems and Structures", 2008, vol. 34, n° 2-3, p. 83–108, http://dx.doi.org/10.1016/j.cl.2007.05.003.

[36] A. P. BLACK, S. DUCASSE, O. NIERSTRASZ, D. POLLET, D. CASSOU, M. DENKER. *Pharo by Example*, Square Bracket Associates, 2009, http://pharobyexample.org/.

[37] A. P. BLACK, N. SCHÄRLI, S. DUCASSE. *Applying Traits to the Smalltalk Collection Hierarchy*, in "Proceedings of 17th International Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'03)", October 2003, vol. 38, p. 47–64 [*DOI :* 10.1145/949305.949311], http://scg.unibe.ch/archive/papers/Blac03aTraitsHierarchy.pdf.

[38] G. BRACHA, D. UNGAR. *Mirrors: design principles for meta-level facilities of object-oriented programming languages*, in "Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04), ACM SIGPLAN Notices", New York, NY, USA, ACM Press, 2004, p. 331–344, http://bracha.org/mirrors.pdf.

[39] L. C. BRIAND, J. W. DALY, J. K. WÜST. *A Unified Framework for Coupling Measurement in Object-Oriented Systems*, in "IEEE Transactions on Software Engineering", 1999, vol. 25, n^{o} 1, p. 91–121, http://dx.doi.org/10.1109/32.748920.

[40] L. C. BRIAND, J. W. DALY, J. K. WÜST. *A Unified Framework for Cohesion Measurement in Object-Oriented Systems*, in "Empirical Software Engineering: An International Journal", 1998, vol. 3, n^{o} 1, p. 65–117.

[41] D. CAROMEL, J. VAYSSIÈRE. *Reflections on MOPs, Components, and Java Security*, in "ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming", Springer-Verlag, 2001, p. 256–274.

[42] D. CAROMEL, J. VAYSSIÈRE. *A security framework for reflective Java applications*, in "Software: Practice and Experience", 2003, vol. 33, n^{o} 9, p. 821–846, http://dx.doi.org/10.1002/spe.528.

[43] S. R. CHIDAMBER, C. F. KEMERER. *A Metrics Suite for Object Oriented Design*, in "IEEE Transactions on Software Engineering", June 1994, vol. 20, n^{o} 6, p. 476–493.

[44] E. CHIKOFSKY, J. CROSS II. *Reverse Engineering and Design Recovery: A Taxonomy*, in "IEEE Software", January 1990, vol. 7, n^{o} 1, p. 13–17, http://dx.doi.org/10.1109/52.43044.

[45] P. COINTE. *Metaclasses are First Class: the ObjVlisp Model*, in "Proceedings OOPSLA '87, ACM SIGPLAN Notices", December 1987, vol. 22, p. 156–167.

[46] M. D'AMBROS, M. LANZA. *Reverse Engineering with Logical Coupling*, in "Proceedings of WCRE 2006 (13th Working Conference on Reverse Engineering)", 2006, p. 189 - 198.

[47] S. DEMEYER, S. DUCASSE, O. NIERSTRASZ. *Object-Oriented Reengineering Patterns*, Morgan Kaufmann, 2002, http://www.iam.unibe.ch/~scg/OORP.

[48] S. DENIER. *Traits Programming with AspectJ*, in "Actes de la Première Journée Francophone sur le Développement du Logiciel par Aspects (JFDLPA'04)", Paris, France, P. COINTE (editor), September 2004, p. 62–78.

[49] S. DUCASSE, T. GÎRBA. *Using Smalltalk as a Reflective Executable Meta-Language*, in "International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006)", Berlin, Germany, LNCS, Springer-Verlag, 2006, vol. 4199, p. 604–618 [*DOI :* 10.1007/11880240_42], http://scg.unibe.ch/archive/papers/Duca06dMOOSEMODELS2006.pdf.

[50] S. DUCASSE, T. GÎRBA, A. KUHN. *Distribution Map*, in "Proceedings of 22nd IEEE International Conference on Software Maintenance (ICSM '06)", Los Alamitos CA, IEEE Computer Society, 2006, p. 203–212 [*DOI :* 10.1109/ICSM.2006.22], http://scg.unibe.ch/archive/papers/Duca06cDistributionMap.pdf.

[51] S. DUCASSE, T. GÎRBA, M. LANZA, S. DEMEYER. *Moose: a Collaborative and Extensible Reengineering Environment*, in "Tools for Software Maintenance and Reengineering", Milano, RCOST / Software Technology Series, Franco Angeli, Milano, 2005, p. 55–71, http://scg.unibe.ch/archive/papers/Duca05aMooseBookChapter.pdf.

[52] S. DUCASSE, O. NIERSTRASZ, N. SCHÄRLI, R. WUYTS, A. P. BLACK. *Traits: A Mechanism for fine-grained Reuse*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", March 2006, vol. 28, n^{o} 2, p. 331–388 [*DOI :* 10.1145/1119479.1119483], http://scg.unibe.ch/archive/papers/Duca06bTOPLASTraits.pdf.

[53] S. DUCASSE, R. WUYTS, A. BERGEL, O. NIERSTRASZ. *User-Changeable Visibility: Resolving Unanticipated Name Clashes in Traits*, in "Proceedings of 22nd International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'07)", New York, NY, USA, ACM Press, October 2007, p. 171–190 [*DOI :* 10.1145/1297027.1297040], http://scg.unibe.ch/archive/papers/Duca07b-FreezableTrait.pdf.

[54] A. DUNSMORE, M. ROPER, M. WOOD. *Object-Oriented Inspection in the Face of Delocalisation*, in "Proceedings of ICSE '00 (22nd International Conference on Software Engineering)", ACM Press, 2000, p. 467–476.

[55] L. ETZKORN, C. DAVIS, W. LI. *A Practical Look at the Lack of Cohesion in Methods Metric*, in "Journal of Object-Oriented Programming", September 1998, vol. 11, n^{o} 5, p. 27–34.

[56] N. FENTON, S. L. PFLEEGER. *Software Metrics: A Rigorous and Practical Approach*, Second, International Thomson Computer Press, London, UK, 1996, 06-8147-I*, envoye a l'inria lille le 19 aout.

[57] K. FISHER, J. REPPY. *Statically typed traits*, University of Chicago, Department of Computer Science, December 2003, n^{o} TR-2003-13, http://www.cs.uchicago.edu/research/publications/techreports/TR-2003-13.

[58] P. W. L. FONG, C. ZHANG. *Capabilities as alias control: Secure cooperation in dynamically extensible systems*, Department of Computer Science, University of Regina, 2004.

[59] A. GOLDBERG. *Smalltalk 80: the Interactive Programming Environment*, Addison Wesley, Reading, Mass., 1984.

[60] L. GONG. *New security architectural directions for Java*, in "compcon", 1997, vol. 0, 97, http://dx.doi.org/10.1109/CMPCON.1997.584679.

[61] B. HENDERSON-SELLERS. *Object-Oriented Metrics: Measures of Complexity*, Prentice-Hall, 1996.

[62] M. HITZ, B. MONTAZERI. *Measure Coupling and Cohesion in Object-Oriented Systems*, in "Proceedings of International Symposium on Applied Corporate Computing (ISAAC '95)", October 1995.

[63] M. HITZ, B. MONTAZERI. *Chidamber and Kemerer's Metrics Suite; A Measurement Theory Perspective*, in "IEEE Transactions on Software Engineering", April 1996, vol. 22, n^{o} 4, p. 267–271.

[64] A. K. JAIN, R. C. DUBES. *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, 1988.

[65] A. K. JAIN, M. N. MURTY, P. J. FLYNN. *Data Clustering: a Review*, in "ACM Computing Surveys", 1999, vol. 31, n^{o} 3, p. 264–323, http://dx.doi.org/10.1145/331499.331504.

[66] G. KICZALES, J. DES RIVIÈRES, D. G. BOBROW. *The Art of the Metaobject Protocol*, MIT Press, 1991.

[67] G. KICZALES, L. RODRIGUEZ. *Efficient Method Dispatch in PCL*, in "Proceedings of ACM conference on Lisp and Functional Programming", Nice, 1990, p. 99–105.

[68] R. KOSCHKE. *Atomic Architectural Component Recovery for Program Understanding and Evolution*, Universität Stuttgart, 2000, http://www.informatik.uni-stuttgart.de/ifi/ps/bauhaus/papers/koschke.thesis.2000.html.

[69] G. LANGELIER, H. A. SAHRAOUI, P. POULIN. *Visualization-based analysis of quality for large-scale software systems*, in "ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering", New York, NY, USA, ACM, 2005, p. 214–223, http://dx.doi.org/10.1145/1101908.1101941.

[70] J. LAVAL, A. BERGEL, S. DUCASSE. *Assessing the Quality of your Software with MoQam*, in "FAMOOSr, 2nd Workshop on FAMIX and Moose in Reengineering", 2008, http://rmod.lille.inria.fr/archives/workshops/Lava08a-Famoosr2008-MoQam.pdf.

[71] S. LIANG, G. BRACHA. *Dynamic Class Loading in the Java Virtual Machine*, in "Proceedings of OOPSLA '98, ACM SIGPLAN Notices", 1998, p. 36–44.

[72] L. LIQUORI, A. SPIWACK. *FeatherTrait: A Modest Extension of Featherweight Java*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", 2008, vol. 30, n^{o} 2, p. 1–32 [*DOI : 10.1145/1330017.1330022*], http://www-sop.inria.fr/members/Luigi.Liquori/PAPERS/toplas-07.pdf.

[73] M. LORENZ, J. KIDD. *Object-Oriented Software Metrics: A Practical Guide*, Prentice-Hall, 1994.

[74] J. I. MALETIC, A. MARCUS. *Supporting Program Comprehension Using Semantic and Structural Information*, in "Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)", May 2001, p. 103–112.

[75] S. MANCORIDIS, B. S. MITCHELL, Y. CHEN, E. R. GANSNER. *Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures*, in "Proceedings of ICSM '99 (International Conference on Software Maintenance)", Oxford, England, IEEE Computer Society Press, 1999.

[76] A. MARCUS, L. FENG, J. I. MALETIC. *3D Representations for Software Visualization*, in "Proceedings of the ACM Symposium on Software Visualization", IEEE, 2003, p. 27-ff.

[77] A. MARCUS, D. POSHYVANYK. *The Conceptual Cohesion of Classes*, in "Proceedings International Conference on Software Maintenance (ICSM 2005)", Los Alamitos CA, IEEE Computer Society Press, 2005, p. 133–142.

[78] R. MARINESCU. *Measurement and Quality in Object-Oriented Design*, Department of Computer Science, Politehnica University of Timişoara, 2002.

[79] R. MARINESCU. *Detection Strategies: Metrics-Based Rules for Detecting Design Flaws*, in "20th IEEE International Conference on Software Maintenance (ICSM'04)", Los Alamitos CA, IEEE Computer Society Press, 2004, p. 350–359.

[80] R. C. MARTIN. *Agile Software Development. Principles, Patterns, and Practices*, Prentice-Hall, 2002.

[81] T. J. McCabe. *A Measure of Complexity*, in "IEEE Transactions on Software Engineering", December 1976, vol. 2, n^{o} 4, p. 308–320.

[82] J. McCall, P. Richards, G. Walters. *Factors in Software Quality*, NTIS Springfield, 1976.

[83] T. Miceli, H. A. Sahraoui, R. Godin. *A Metric Based Technique For Design Flaws Detection And Correction*, in "Proceedings IEEE Automated Software Engineering Conference (ASE)", 1999.

[84] M. S. Miller. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*, Johns Hopkins University, Baltimore, Maryland, USA, May 2006.

[85] M. S. Miller, C. Morningstar, B. Frantz. *Capability-based Financial Instruments*, in "FC '00: Proceedings of the 4th International Conference on Financial Cryptography", Springer-Verlag, 2001, vol. 1962, p. 349–378.

[86] B. S. Mitchell, S. Mancoridis. *On the Automatic Modularization of Software Systems Using the Bunch Tool*, in "IEEE Transactions on Software Engineering", 2006, vol. 32, n^{o} 3, p. 193–208.

[87] N. Moha, D. loc Huynh, Y.-G. Gueheneuc. *Une taxonomie et un métamodèle pour la détection des défauts de conception*, in "Langages et Modèles à Objets", 2006, p. 201–216.

[88] O. Nierstrasz, S. Ducasse, N. Schärli. *Flattening Traits*, in "Journal of Object Technology", May 2006, vol. 5, n^{o} 4, p. 129–148, http://www.jot.fm/issues/issue_2006_05/article4.

[89] L. Ponisio, O. Nierstrasz. *Using Context Information to Re-architect a System*, in "Proceedings of the 3rd Software Measurement European Forum 2006 (SMEF'06)", 2006, p. 91–103, http://scg.unibe.ch/archive/papers/Poni06aSimulatedAnnealing.pdf.

[90] P. J. Quitslund. *Java Traits — Improving Opportunities for Reuse*, OGI School of Science & Engineering, Beaverton, Oregon, USA, September 2004, n^{o} CSE-04-005.

[91] D. Raţiu, S. Ducasse, T. Gîrba, R. Marinescu. *Using History Information to Improve Design Flaws Detection*, in "Proceedings of 8th European Conference on Software Maintenance and Reengineering (CSMR'04)", Los Alamitos CA, IEEE Computer Society, 2004, p. 223–232, http://scg.unibe.ch/archive/papers/Rati04aHistoryImproveFlawsDetection.pdf.

[92] J. Reppy, A. Turon. *A Foundation for Trait-based Metaprogramming*, in "International Workshop on Foundations and Developments of Object-Oriented Languages", 2006.

[93] F. Rivard. *Pour un lien d'instanciation dynamique dans les langages à classes*, in "JFLA96", INRIA — collection didactique, January 1996.

[94] J. H. Saltzer, M. D. Schoroeder. *The Protection of Information in Computer Systems*, in "Fourth ACM Symposium on Operating System Principles", IEEE, September 1975, vol. 63, p. 1278–1308.

[95] N. Sangal, E. Jordan, V. Sinha, D. Jackson. *Using Dependency Models to Manage Complex Software Architecture*, in "Proceedings of OOPSLA'05", 2005, p. 167–176.

[96] N. SCHÄRLI, A. P. BLACK, S. DUCASSE. *Object-oriented Encapsulation for Dynamically Typed Languages*, in "Proceedings of 18th International Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'04)", October 2004, p. 130–149 [*DOI :* 10.1145/1028976.1028988], http://scg.unibe.ch/archive/papers/Scha04bOOEncapsulation.pdf.

[97] N. SCHÄRLI, S. DUCASSE, O. NIERSTRASZ, A. P. BLACK. *Traits: Composable Units of Behavior*, in "Proceedings of European Conference on Object-Oriented Programming (ECOOP'03)", LNCS, Springer Verlag, July 2003, vol. 2743, p. 248–274 [*DOI :* 10.1007/B11832], http://scg.unibe.ch/archive/papers/Scha03aTraits.pdf.

[98] C. SMITH, S. DROSSOPOULOU. *Chai: Typed Traits in Java*, in "Proceedings ECOOP 2005", 2005.

[99] G. SNELTING, F. TIP. *Reengineering Class Hierarchies using Concept Analysis*, in "ACM Trans. Programming Languages and Systems", 1998.

[100] K. J. SULLIVAN, W. G. GRISWOLD, Y. CAI, B. HALLEN. *The Structure and Value of Modularity in SOftware Design*, in "ESEC/FSE 2001", 2001.

[101] D. VAINSENCHER. *MudPie: layers in the ball of mud.*, in "Computer Languages, Systems & Structures", 2004, vol. 30, n^{o} 1-2, p. 5–19.

[102] R. WETTEL, M. LANZA. *Program Comprehension through Software Habitability*, in "Proceedings of ICPC 2007 (15th International Conference on Program Comprehension)", IEEE CS Press, 2007, p. 231–240.

[103] R. WETTEL, M. LANZA. *Visualizing Software Systems as Cities*, in "Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis)", 2007, p. 92–99, http://dx.doi.org/10.1109/VISSOF.2007.4290706.

[104] N. WILDE, R. HUITT. *Maintenance Support for Object-Oriented Programs*, in "IEEE Transactions on Software Engineering", December 1992, vol. SE-18, n^{o} 12, p. 1038–1044.