INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team RMoD

Analyses and Language Constructs for Object-Oriented Application Evolution

Lille - Nord Europe

Theme : Distributed Systems and Services

Table of contents

The RMod team has been created on February 2009 and became an Inria project on October 2009 TODO: to check.

1. Team

Research Scientist

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

Faculty Member

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

Technical Staff

Cyrille Delaunay [Associate Engineer from INRIA]

PhD Student

Hani Abdeen [INRIA]
Jannik Laval [INRIA]
Jean-Baptiste Arnaud [Ministère de l'éducation et de la recherche scientifique]
Veronica Uquillaz-Gomez [Vrije Universiteit Brussels]
Gwenael Casaccio [Bourse Région]

Post-Doctoral Fellow

Simon Denier [PostDoc Inria]
Jean-Rémy Falleri [PostDoc Inria]

Administrative Assistant

Marie-Bénédicte Dernoncourt [Secretary (SAR) Inria]

2. Overall Objectives

2.1. Introduction

RMoD's general vision is summarized as follows:

> *How to help development teams to maintain and evolve their software programs?*

This vision is refined in two objectives which are treated from two complementary perspectives: 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 for the coming years will focus on the *remodularization* of object-oriented applications. There is a limited number of approaches to understand, analyze and restructure large legacy systems. 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: either at the same paradigmatic level (modules/packages) or for a migration towards other abstractions such as components or aspects. 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. Finally we are going to work on migrations directed by the results of our analysis. We plan to support the migration process by providing feedback on the impact of the suggested transformations. These steps will be implemented on top of the Moose reengineering environment and validated on large open-source applications in different languages such as JBoss, ArgoUML or Squeak.

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. Using class inheritance to support unanticipated software evolution is proved to be the source of broken typing relations and code duplication, leading to a fragile and complex system . At a micro level, single class inheritance suffers from lack of adaptation, and the complexity of multiple inheritance is known to not be an alternative. At a macro-level, traditional packaging systems mainly deal with name resolution. In addition, 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

- Moose 4.0, our open-source reengineering platform, was released (http://moose.unibe.ch/)
- Pharo 1.0, a new open-source Smalltalk, was released (http://www.pharo-project.org) with accompanying book [38] (http://www.pharobyexample.org).
- Stéphane Ducasse and Damien Pollet have published a survey of the state of the art in software architecture reconstruction [8] in the IEEE Transactions on Software Engineering (ranked A+). This paper classifies 181 publications and is an extension of earlier work that was itself nominated best paper at the Conference on Software Maintenance and Reengineering in 2007 [97].
- The ongoing collaboration on Traits between RMoDand the VUB Software Language Lab lead to an approach of state and visibility control for traits in a language with lexical nesting. The paper [29] was accepted at the ECOOP 2009 conference (ranked A+).

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?

[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.

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 [59], [57]. 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 [58]. The problem is that understanding large systems is already difficult [111], [82], [84], [77], 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 [77], [112]. 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 [55], 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 [58] 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 on going work:

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 [78]. 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 that top layers [88]. Until now, few works have attempted to identify layers in practice: Mudpie [110] is a first cut at identifying cycles between packages as well as package groups potentially representing layers. DSM (dependency structure matrix) [109], [104] 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 [76]. 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 [113], [62].

Some approaches based on Formal Concept Analysis [108] 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 [72], [73] have been proposed to identify modules [83], [94]. 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 [81], [64], [69] such as the McCabe complexity metrics [89]. In the more specific case of object-oriented programming, assessing cohesion and coupling have been the focus of several metrics. However their success are rather mitigated as the number of critics raised. For example, LCOM [53] has been highly criticized [63], [71], [70], [41], [49], [50]. Other approaches have been proposed such as RFC and CBO [53] 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) [43], or some of metrics are not clearly specified (MCX, CCO, CCP, CRE) [81]. New cohesion measures were proposed [85], [98] 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 allow the programer 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 researcher 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 [63], [71], [70], [50]. 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, define new ones if necessary for packages and work as well as coupling metrics [41], [49]. 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, groups of packages working as layers. For this purpose we will perform controlled experiments and in a second period apply it on one of the selected case study. Based on these results, we will propose enhanced or a specific DSM.

Layer identification. We want to propose an approach to identify layer 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 [76].

Within the context of the REMOOSE Associated team with the Geodes group of the DIRO, we plan to use explanation-based constraint programming to help remodularization. It is not clear if this will work but several research questions are worth to be investigated: how to model a remodularization situation as a declarative set of constraints? Is this model explanation-based constraint programming useful and scalable?

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 [90] 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 for which some of them are not automatically computed. In an effort of conformance, the ISO (International Standardisation Organisation) and the IEC (International Electronical Commission) are conjointly defined in 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 family of work focusing on design evaluation as quality criteria [91], [100], [95] and new quality models: QMOOD is for example an hierarchical quality model which proposes to link directly quality criteria to software metrics based on object-oriented software metrics [42] while other work focus on linking different high level criteria with software code metrics [86], [87].

Research Agenda. Since software quality is fuzzy by definition and that 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 [106], [60] and classboxes [44] 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 [106], [60]. 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 [56], [99], [45], [61] and several type systems where defined [65], [107], [101], [80].

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 [47]. 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 [60], stateful [46], and freezable [61]) and their operators. We will compare these models by (1) implementing a trait-based collection hierarchy, (2) analyzing several existing applications 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 [96]. This is 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 traits 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 [54], then from Smalltalk [67].

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 [75]). 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 [48], as well as controlling the access to reflective features [51], [52] are important challenges. We plan to:

- Study the security abstractions available in erights[5] [93], [92], Java as well as classLoader strategies [79], [68].

- Categorize the different reflective features of languages such as CLOS [74], Python and Smalltalk

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

[102] and identify suitable security mechanisms and infrastructure [66].

- Assess different security models (access rights, capabilities [103]...) 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 [48],
  - the use encapsulation policies as a basis to restrict the interfaces of the controlled objects [105],
  - 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 the language offering the largest set of reflective features such as pointer swapping, class changing, class definition...) [102].

4. Software

4.1. Moose

Participants: Stéphane Ducasse [correspondant], Simon Denier, Jannik Laval, Hani Abdeen, Tudor Girba [Software Composition Group (University of Bern)].

See also the web page http://moose.unibe.ch/.

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 RMoDteam is currently the main maintainer of the Moose platform. There have been 150 publications (journal, international conferences, PhD theses) based on the Moose environment.

4.2. Pharo

Participants: Stéphane Ducasse [correspondant], Marcus Denker, Adrian Lienhard [Software Composition Group (University of Bern)].

See also the web page http://www.pharo-project.org/.

The platform. Pharo is a new, slimmed down, implementation of Smalltalk that is free software. Pharo is based on the Squeak dialect of Smalltalk, but focuses to provide a platform for professional development both in industry and research.

Pharo's goal is to deliver a clean, innovative, free open-source Smalltalk environment. By providing a stable and small core system, excellent developer tools, and maintained releases, Pharo is a platform to build and deploy mission critical Smalltalk applications.

A first stable version, *Pharo 1.0*, will be released by the end of 2009. In Parallel, development continues with the unstable development branch 1.1 which has seen already over 40 incremental releases as of mid November 2009.

Traits. Traits are a simple composition mechanism for structuring object-oriented programs. A Trait is essentially a parameterized set of methods; it serves as a behavioral building block for classes and is the primitive unit of code reuse. With Traits, classes are still organized in a single inheritance hierarchy, but they can make use of Traits to specify the incremental difference in behavior with respect to their superclasses. Traits have been implemented in several object-oriented programming languages including Perl, Scala, C#, Squeak, Visualworks and Fortress.

5. New Results

5.1. Architecture

Participants: Stéphane Ducasse, Damien Pollet, Azadeh Razavizadeh.

Software Architecture Reconstruction taxonomy. To maintain and understand large applications, it is crucial to know their architecture. The first problem is that unlike classes and packages, architecture is not explicitly represented in the code. The second problem is that successful applications evolve over time, so their architecture inevitably drifts. Reconstructing the architecture and checking whether it is still valid is therefore an important aid. While there is a plethora of approaches and techniques supporting architecture reconstruction, there is no comprehensive state of the art and it is often difficult to compare the approaches. We defined and presented a state of the art in software architecture reconstruction approaches [8].

Multiple viewpoints. A software system's architecture, its elements and the way they interact, constitute valuable assets for comprehending the system. Many approaches have been developed to help comprehending software systems in different manners. Most of them focus on structural aspects. We believe offering multiple views of the same system, using domain knowledge, helps understanding a software system as a whole. To correlate domain information and existing software systems, different viewpoints are considered and modeled. Viewpoints guide the extraction of architectural views, the later representing different system facets. In [24], [25], we propose a recursive framework, an approach that expresses domain knowledge as viewpoints to guide the extraction process. It provides multiple architectural views according to multiple given viewpoints.

5.2. Package understanding and Assessing

Participants: Stéphane Ducasse, Simon Denier, Nicolas Anquetil, Jannik Laval, Hani Abdeen.

Another way 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 reorganized packages to minimize coupling.

DSM Cycle identification and support. Dependency Structure Matrix (DSM) has been successfully applied to identify software dependencies among packages and subsystems. A number of algorithms were proposed to compute the matrix so that it highlights patterns and problematic dependencies between subsystems. However, existing DSM implementations often miss important information to fully support reengineering effort. For example, they do not clearly qualify and quantify problematic relationships, an information which is crucial to support remediation tasks. In [31], [20] we presented an enriched DSM (eDSM) where cells are enriched with contextual information about (i) the type of dependencies (inheritance, class reference...), (ii) the proportion of referencing entities, (iii) the proportion of referenced entities. We distinguish independent cycles and stress potentially simple fixes for cycles using color conventions. This work is language independent and has been implemented on top of the Moose reengineering environment. It has been applied to non-trivial case studies among which ArgoUML, and Morphic the UI framework available in two open-source Smalltalks (Squeak and Pharo). Solutions to actual modularization problems identified using our eDSM approach have been applied and retrofitted in the Pharo main distribution.

Automatic Package Coupling and Cycle Minimization. Object-oriented software is usually organized into subsystems using the concepts of package or module. Such modular structure helps applications to evolve when facing new requirements. However, studies show that as software evolves to meet requirements and environment changes, modularization quality degrades. To help maintainers improve the quality of software modularization we have designed and implemented a heuristic search-based approach for automatically optimizing inter-package connectivity (i.e., dependencies). In [13] we present our approach and its underlying techniques and algorithm based on Simulated Annealing technique. We show through a case study how it enables maintainers to optimize OO package structure of source code.

5.3. Software Quality

Participants: Stéphane Ducasse, Simon Denier, Nicolas Anquetil, Jannik Laval, Hani Abdeen.

ISO 9126 promotes a three-level model of quality (factors, criteria, and metrics) which allows one to assess quality at the top level of factors and criteria. However, it is difficult to use this model as a tool to increase software quality. In the Squale model, we add practices as an intermediate level between metrics and criteria. Practices abstract away from raw information (metrics, tool reports, audits) and provide technical guidelines to abide by. Moreover, practice marks may be adjusted using formulae to suit a company's development habits or exigences: for example bad marks are stressed to point to places which need more attention. The Squale model has been developed and validated over the last couple of years in an industrial setting with Air France-KLM and PSA Peugeot-Citroën [78][23], [33].

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 [78]. 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 [34].

5.4. Tools Infrastructure

Participants: Stéphane Ducasse, Simon Denier, Nicolas Anquetil, Jannik Laval.

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

Supporting multiple source models efficiently. When reengineering large systems, software developers would like to assess and compare the impact of multiple change scenarios without actually performing these changes. A change can be effected by applying a tool to the source code, or by a manual refactoring. In addition, tools run over a model are costly to redevelop. It raises an interesting challenge for tools implementors: how to support modification of large source code models to enable comparison of multiple versions. One naive approach is to copy the entire model after each modification. However, such an approach is too expensive in memory and execution time. In [21] we explore different implementations that source code metamodels support multiple versions of a system. We propose a solution based on dynamic binding of entities between multiple versions, providing good access performance while minimizing memory consumption.

Metamodeling at work. Object-oriented modelling languages such as EMOF are often used to specify domain specific meta-models. However, these modelling languages lack the ability to describe behavior or operational semantics. Several approaches have used a subset of Java mixed with OCL as executable meta-languages. In [11] we show how we use Smalltalk as an executable meta-language in the context of the Moose reengineering environment. We present how we implemented EMOF and its behavioral aspects. Over the last decade we validated this approach through incrementally building a meta-described reengineering environment. Such an approach bridges the gap between a code-oriented view and a meta-model driven one. It avoids the creation of yet another language and reuses the infrastructure and run-time of the underlying implementation language. It offers an uniform way of letting developers focus on their tasks while, at the same time, allowing them to

meta-describe their domain model. The advantage of our approach is that developers use the same tools and environment they use for their regular tasks. Still the approach is not Smalltalk specific but can be applied to any language offering an introspective API such as Ruby, Python, CLOS, Java and C#.

5.5. IDE for Reengineers

Participants: Stéphane Ducasse, David Rothlisberger.

Navigating large software systems is difficult as they are composed of numerous artifacts distributed in a huge space, with relationships between artifacts that often remain hidden and obscure. As a consequence, developers using a modern interactive development environment (IDE) are forced to open views on numerous source artifacts to reveal these hidden relationships, leading to a crowded workspace with many opened windows or tabs. Developers often lose the high level view in such a cluttered workspace as IDEs provide little support to get rid of unused windows. AutumnLeaves automatically selects windows unlikely to be needed in the future to be closed or grayed out while other important ones are displayed more prominently. This reduces the number of windows opened at a time and adds structure to the developer's workspace. We validated AutumnLeaves with a benchmark evaluation using recorded navigation data of various developers to determine the prediction quality of our algorithms [28], [26]. Mainstream IDEs generally rely on the static structure of a software project to support browsing and navigating it. We propose HeatMaps, a simple but highly configurable technique to enrich the way an IDE displays the static structure of a software system with additional kinds of information. A heatmap highlights software artifacts according to various metric values, such as bright red or pale blue, to indicate their potential degree of interest. We present a prototype system that implements heatmaps, and we describe an initial study that assesses the degree to which different heatmaps effectively guide developers in navigating software [27].

5.6. Traits

Participants: Stéphane Ducasse, Marcus Denker, Damien Pollet.

Traits are groups of methods that can be used to compose classes. They do not have a runtime existence and are conceptually folded into the classes that use them. Traits have been implemented in different languages.

Lexical Scoping and Traits. Traits are reusable building blocks that can be composed to share methods across unrelated class hierarchies. Original traits are stateless and cannot express visibility control for methods. Two extensions, stateful traits and freezable traits, have been proposed to overcome these limitations. However, these extensions introduce complexity and have not yet been combined to simultaneously add both state and visibility control to traits. In [29] we revisit the addition of state and visibility control to traits. Rather than extending the original traits model with additional operations, we allow traits to be lexically nested within other modules. Traits can then have (shared) state and visibility control by hiding variables or methods in their lexical scope. Although the Traits' "flattening property" has to be revisited, the combination of traits with lexical nesting results in a simple and expressive trait model. We discuss an implementation of the model in AmbientTalk and specify its operational semantics.

Reusing and Composing Tests with Traits. Single inheritance often forces developers to duplicate code and logic. This widely recognized situation affects both business code and tests. In a large and complex application whose classes implement many groups of methods (protocols), duplication may also follow the application's idiosyncrasies, making it difficult to specify, maintain, and reuse tests. The research questions we faced are (i) how can we reuse test specifications across and within complex inheritance hierarchies, especially in presence of orthogonal protocols; (ii) how can we test interface behavior in a modular way; (iii) how far can we reuse and parametrize composable tests. In this paper, we compose tests out of separately specified behavioral units of reuse —traits. We propose test traits, where: (i) specific test cases are composed from independent specifications; (ii) executable behavior specifications may be reused orthogonally to the class hierarchy under test; (iii) test fixtures are external to the test specifications, thus are easier to specialize. Traits have been successfully applied to test two large and critical class libraries in Pharo, a new Smalltalk dialect based on Squeak, but are applicable to other languages with traits [18].

5.7. Constructs for Reflective Secure Languages

Participants: Stéphane Ducasse, Marcus Denker, Damien Pollet, Jean-Baptiste Arnaud, Gwenael Casaccio.

Evolving a Reflective Language. While implementing traits in Smalltalk, our first reflex was to take advantage of the fact that they are not run-time entities: we optimized the implementation for space and hence shared methods between traits and classes. However, by doing so we broke the introspective API of Smalltalk. This illustrates a more general problem seen in all reflective systems: the implementation serves both as a model for execution and as the model that is exposed to the programmer. There is a conflict of interests between the information necessary for execution and the information the programmer is interested in. In addition, as soon as the implementation is exposed via reflection, we are no longer free to optimize it. As the complete implementation is visible reflectively, there is no way to hide the optimizations. Few papers report errors and this is one of them. We report our experience facing the initial API mismatch, which has a significant impact on the system because the language is reflective (i.e., written in itself and causally connected). We present the new introspective API we put in place [17].

Dynamic Read-only objects. Supporting read-only and side effect free execution has been the focus of a large body of work in the area of statically typed programming languages. So far, dynamically typed languages have been poorly addressed despite their increasing presence in web pages and multi-language applications. Read-onlyness in dynamically typed languages is difficult to achieve because of the absence of a type checking phase and the support of an open-world assumption in which code can be constantly added.

We propose Dynamic Read-Only objects (DRO) to address this issue by providing a read-only execution and its dynamic propagation over an object graph. Graph and subgraph of referencing objects may be dynamically set as read-only based on client invocations and can be dynamically adjusted. To enable dynamic read-only execution, we introduce smart object proxies that dynamically propagate the read-only property to the underlying object graph. We have implemented dynamic read-only execution in Pharo, an open-source Smalltalk, and applied them to realize side-effect free assertions. This work is under publication.

Confining objects. Long-lived systems rely on reflective self-modification to evolve. Unfortunately, since the system is at both ends of a causal loop, this means modifications that impact the reflective layer itself can be overly difficult to apply. In [14] we introduce ObjectSpaces, a reification of the familiar Smalltalk image as a first-class entity. By confining the system inside an ObjectSpace, we isolate the evolution tools from it, while still giving them reflective access to the confined system. We describe the ObjectSpaces idea, the interface to communicate, inspect, and debug objects contained inside and ObjectSpace, based on a prototype implementation in GNU Smalltalk.

5.8. Dynamic Web Development

Participants: Stéphane Ducasse, Lukas Renggli.

Developing web applications is difficult since (1) the client-server relationship is asymmetric: the server cannot update clients but only responds to client requests and (2) the navigation facilities of web browsers lead to a situation where servers cannot control the state of the clients. Page-centric web application frameworks fail to offer adequate solutions to model control flow at a high-level of abstraction. Developers have to work manually around the shortcomings of the HTTP protocol. Some approaches offer better abstractions by composing an application out of components, however they still fail to offer modeling control flow at a high level. Continuation-based approaches solve this problem by providing the facilities to model a control flow over several pages with one piece of code. However combining multiple flows inside the same page is difficult. This article presents Seaside. Seaside is a framework which combines an object-oriented approach with a continuation-based one. A Seaside application is built out of components (i.e., objects) and the logic of the application benefits from the continuation-based program flow infrastructure. Seaside offers a unique way to have multiple control flows on a page, one for each component. This enables the developer to write components that are highly reusable and that can be used to compose complex web applications with higher quality in less time. We wrote a book [39]: Dynamic web development with Seaside http://book.seaside.st/ with Lukas Renggli which is a member of the associated team REMOOSE and also one of the main developer of Seaside.

Despite the technological evolution of the web, there is still no standard mechanism to send data or events from the server to the client without an explicit request from the later, thus forcing the web browser to constantly poll the server for updates. To solve this problem a set of techniques under the name of Comet were proposed, allowing to send information from the server to the web browser without an explicit client request. In [19] we introduce Meteoroid, a Comet approach to make "live" Seaside applications. Our framework exploits the Model-View-Controller (MVC) paradigm for building simple yet scalable web applications, requiring very little programming effort.

6. Contracts and Grants with Industry

6.1. MITI-QualitéSys

Participants: Simon Denier, Stéphane Ducasse.

MITI-QualitéSys is a software company developing a bug detection platform especially in the context of concurrent programming. The goal of this collaboration is the design of dedicated visualizations for presenting the results of MITI-QualitéSys software analyses.

7. Other Grants and Activities

7.1. National Initiatives

7.1.1. Squale FUI Project

Participants: Stéphane Ducasse [correspondant], Nicolas Anquetil, Simon Denier, Jannik Laval, Hani Abdeen.

Squale (Software QUALity Enhancement) is a project funded by the Pole of Compétitivité System@tic. It is led by Qualixo and composed by LIASD Université Paris 8, INRIA Lille-Nord Europe, Air France, PSA (Peugeot Citroen) and Paqtigo.

The aim of the Squale project is to define quality models and to implement an open-source application, in order to offer a solution that: knows how to aggregate raw quality information (like metrics for instance) - given by third party technologies, into high level factors, offers dashboards which present those factors and allow to dig deeply into the code quality, shows the evolution of the quality over time, and gives economical indicators about the return of investment of quality efforts.

7.2. Exterior research visitors

In the context of the REMOOSE associated Team with the University of Bern and Montréal we got a large number of visitors over a period of one week (L. Renggli, O. Nierstrasz, D. Röthlisberger, D. Girba, J. Ressia). This leads to the production of two books and several research articles (see below).

7.3. European Initiatives

Participants: Stéphane Ducasse [correspondant], Veronica Uquillaz-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.

7.3.3. Associated Team with University of Bern and Montréal (REMOOSE)

REMOOSE is a collaboration between INRIA Lille, SCG University of Bern/Switzerland and Université de Montréal, Canada. The theme of the collaboration is remodularisation of object oriented systems. The collaboration started in 2008 and is now in its second year.

Results.

- Stabilisation of a new version of the FAMIX source code metamodel. FAMIX3.0 has been designed and specified in the FAME meta-metamodel and implemented in the Pharo environment.
- We released an first official version of Moose based on FAMIX30 and reimplemented on top of Pharo during the holiday 2009. The University of Bern and the RMoD research groups are still actively working on this port.
- We evaluated explanation-based algorithms to see how we could express package structures. However contrary to what we expected we gave up this approach because the first results were clearly not encouraging. Instead we conducted a number of experiments on Dependency Structure Matrix. We developed on Moose a new DSM and enhanced the traditional DSM to offer a wider range of information. We got one paper at WCRE 2009, an international conference on reverse engineering. We basically enhanced this well known technique with local and qualitative information to help remodularizing a system.

Workshops and Seminars. In the context of REMOOSE, we organized one international workshop and two seminars:

- International workshop: FAMOOSr. We organized International 3rd Workshop on FAMIX and Moose in Reengineering held in conjunction with the International Working Conference on Reverse Engineering (WCRE 2009). The official website for this event is http://moose.unibe.ch/events/famoosr2009 (14 October 2009, Lille).
- International seminars: SATTOSE and PHARO. SATTOSE was seminar organized at Cap-Hornu from the 11 to 15 of May 2009. 9 researchers attended the seminar. With the support of INRIA Lille–Nord Europe and the associated team, we conducted a coding seminar around the pharo environment. This event took place at the University of Lille from Saturday 17 to Tuesday 20 October 2009. It is worth mentioning that this event attracted people that are not part of an institution partner. This demonstrates the relevance of the effort for companies.

Publication production. This year saw the public release of two books by authors of the REMOOSE associated team.

- Seaside Book [39]. *Dynamic Web Development with Seaside*. It is a free open-source book and will be later proposed as a physical book with a on-demand mechanism.

- Pharo Book [38]. *Pharo by Example*. It is the Pharo version of the Squeak by example book that was released as a free online book, and a paperback version last year. Squeak by Example is available at Squeak by Example and was downloaded around 90000 times for the english and french versions. Several translations of Pharo by example in spanish and french are under way.

In addition we collaborated on several joint publications. Here is the list of joint papers so far:

- David Röthlisberger and Oscar Nierstrasz and Stephane Ducasse and Damien Pollet and Romain Robbes, Supporting Task-oriented Navigation in IDEs with Configurable HeatMaps, Proceedings of the 17th International Conference on Program Comprehension (ICPC 2009), IEEE Computer Society, Los Alamitos, CA, USA, 253-257, 2009.
- David Röthlisberger and Oscar Nierstrasz and Stephane Ducasse and Alexandre Bergel, Tackling Software Navigation Issues of the Smalltalk IDE, Proceedings of International Workshop on Smalltalk Technologies (IWST 2009), ACM Digital Library, 2009, to appear.
- Stephane Ducasse and Tudor Girba and Adrian Kuhn and Lukas Renggli, Meta-Environment and Executable Meta-Language using Smalltalk: an Experience Report, Journal of Software and Systems Modeling (SOSYM), Springer Verlag, 2009.
- Stephane Ducasse, Marcus Denker and Adrian Lienhard, Evolving a Reflective Language: Lessons Learned from Implementing Traits, Proceedings of International Workshop on Smalltalk Technologies (IWST 2009), ACM Digital Library, 2009, to appear.
- Gwenael Casaccio, Damien Pollet, Marcus Denker and Stephane Ducasse, Object Spaces for Safe Image Surgery, Proceedings of International Workshop on Smalltalk Technologies (IWST 2009), ACM Digital Library, 2009, to appear.
- Jannik Laval and Simon Denier and Stéphane Ducasse and Andy Kellens, Supporting Incremental Changes in Large Models, Proceedings of International Workshop on Smalltalk Technologies (IWST 2009), ACM Digital Library, 2009, to appear.
- Alexandre Bergel and Stephane Ducasse and Lukas Renggli, Seaside – Advanced Composition and Control Flow for Dynamic Web Applications, ERCIM News, 72, 2008.
- S. Denier and H. Sahraoui, Understanding the Use of Inheritance with Visual Patterns, International Symposium on Empirical Software Engineering and Measurement (ESEM), 2009.
- Lukas Renggli, Marcus Denker and Oscar Nierstrasz, Language Boxes. Bending the Host Language with Modular Language Changes, the 2nd International Conference on Software Language Engineering (SLE 2009), LNCS, 2009, to appear.
- Hani Abdeen and Stephane Ducasse and Houari Sahraoui and Ilham Alloui: Automatic Package Coupling and Cycle Minimization, International Working Conference on Reverse Engineering (WCRE), IEEE Computer Society Press, 2009.

7.4. International Initiatives

Participants: Stéphane Ducasse [correspondant], Simon Denier, Marcus Denker.

7.4.1. STICAmsud

This project focuses on software remodularization. Aspects, Traits and Classboxes are proven software mechanisms to provide modules in software applications. However, reenginering-based methodologies using these mechanisms have not yet been explored so far. This project intents 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 also use these mechanisms. 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).

8. Dissemination

8.1. Scientific Community Animation

8.1.1. Examination Committees

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

- *Visualizing, Assessing and Re-Modularizing Object-Oriented Architectural Elements*, Hani Abdeen, Université de Lille, France. 24/11/09 (advisor).
- *Contributions à l'IDM: reconstruction et alignement de modèles de classes*, Jean-Rémi Falléri, Université de Montpellier, France. 28/10/09 (referee).
- *Programmer Friendly Refactoring Tools*, Emerson Murphy-Hill, Portland State University, USA. 26/2/09 (referee).
- *Conception d'un language dédié à l'analyse et la transformation de programmes*, Emilie Balland, Université de Nancy, France. 11/03/08 (referee).
- *Of Change and Software*, Romain Robbes, Université de Lugano, Suisse. 1/12/08 (referee).

8.1.2. Scientific Community Animation

The team has organized two international conferences:

- WCRE'2009, Lille (International Working Conference on Reverse engineering) - 80 participants
- ESUG'2009, Brest (International Smalltalk conference) - 150 participants

Stéphane Ducasse has been/is :

- Member of the following committees:
  - European Conference on Object-Oriented Programming (ECOOP 2010).
  - International Conference on Software Maintenance (ICSM 08, 2010).
  - International Conference on the Unified Modeling Language (Models 09).
  - International Conference on Objects, Models, Components, Patterns (TOOLS 2009, TOOLS 2010).
  - International Conference on Software Composition (SC 09).
  - Program Chair of the International Smalltalk Conference (2008, 2009).
- 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)
- Keynote speaker:
  - Software Composition 2009 (Switzerland)
  - Smalltalks 2009 (Argentina)

Nicolas Anquetil has been/is :

- Member of the following program committees:
  - Working Conference on Reverse Engineering (WCRE 2009).
  - International Conference on Software Maintenance (ICSM 2010).
  - European Conference on Software Maintenance and Re-engineering (CSMR 2010)
- Recipient of the WCRE'09 most influential paper award:
  - Presented at WCRE'09

Marcus Denker has been/is :

- Member of the following committees:
  - International Conference on Software Composition (SC 09).
  - ECOOP Workshop on Reflection, AOP, and Meta-Data for Software Evolution (RAM-SE 2009).
  - ESUG International Smalltalk Workshop (IWST09).
- Reviewer for the following journals:
  - Elsevier Science of Computer Programming (SCP), ISSN: 0167-6423.
  - IEEE Transactions on Software Engineering (TSE), ISSN 0098-5589.

Damien Pollet has been/is :

- Co-chair for the workshops of WCRE'09.
- Speaker at Tools'09 for [18].
- Reviewer for the following conferences:
  - MajecSTIC'09 (francophone).
  - Models 09.
  - International Conference on Software Composition (SC 2009).
  - International Conference on Objects, Models, Components, Patterns (Tools 2009).
  - Working Conference on Reverse Engineering (WCRE 2009).

Simon Denier has been/is :

- Member of the following committees:
  - Working Conference on Reverse Engineering (WCRE 2009).
  - International Conference on Software Composition (SC 2010).
- Co-organiser of the FAMOOSr workshop (2009, 2010).
- Reviewer for the IEEE Software journal.
- Reviewer for the following conferences: LMO 2009, MODELS 2009.

8.2. Teaching

Stéphane Ducasse teaches a course on advanced OO design in the Master of Computer Science of the University of Lille 1.

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.

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, T. LETHBRIDGE. *Extracting Concepts from File Names; a New File Clustering Criterion*, in "International Conference on Software Engineering (ICSE 1998)", 1998, p. 84–93.

[2] N. ANQUETIL, T. LETHBRIDGE. *Experiments with Clustering as a Software Remodularization Method*, in "Proceedings of WCRE '99 (6th Working Conference on Reverse Engineering)", 1999, p. 235–255.

[3] M. DENKER, S. DUCASSE, É. TANTER. *Runtime Bytecode Transformation for Smalltalk*, in "Journal of Computer Languages, Systems and Structures", vol. 32, n^{o} 2-3, July 2006, p. 125–139, 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)", vol. 8, n^{o} 1, February 2009, p. 5–19, 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)", vol. 31, n^{o} 1, January 2005, p. 75–90, 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", vol. 24, n^{o} 5, 2007, p. 56–63, http://www.computer.org/portal/cms_docs_software/software/homepage/2007/S507/s5056.pdf.

[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)", vol. 28, n^{o} 2, March 2006, p. 331–388, 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", vol. 35, n^{o} 4, July 2009, p. 573-591, 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, http://scg.unibe.ch/archive/papers/Scha04bOOEncapsulation.pdf.

Year Publications

Articles in International Peer-Reviewed Journal

[11] 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)", vol. 8, n° 1, February 2009, p. 5–19, http://scg.unibe.ch/archive/drafts/Duca08a-Sosym-ExecutableMetaLanguage.pdfCH.

[12] S. DUCASSE, D. POLLET. *Software Architecture Reconstruction: A Process-Oriented Taxonomy*, in "IEEE Transactions on Software Engineering", vol. 35, n° 4, July 2009, p. 573-591, http://rmod.lille.inria.fr/archives/papers/Duca09c-TSE-SOAArchitectureExtraction.pdf.

International Peer-Reviewed Conference/Proceedings

[13] H. ABDEEN, S. DUCASSE, H. A. SAHRAOUI, I. ALLOUI. *Automatic Package Coupling and Cycle Minimization*, in "International Working Conference on Reverse Engineering (WCRE), Washington, DC, USA", IEEE Computer Society Press, 2009, p. 103–112, http://rmod.lille.inria.fr/archives/papers/Abde09b-WCRE2009-AutomaticPackageCoupling.pdf CA .

[14] G. CASACCIO, D. POLLET, M. DENKER, S. DUCASSE. *Object Spaces for Safe Image Surgery*, in "Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2009), New York, USA", ACM digital library, 2009, http://rmod.lille.inria.fr/archives/workshops/Casa09a-IWST09-ObjectSpaces.pdf CL .

[15] S. DENIER, D. POLLET, S. DUCASSE. *Proposals for the Reborn Pharo Developer*, in "Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2009), Brest, France", 2009, http://rmod.lille.inria.fr/archives/workshops/Deni09b-IWST09-PharoBrowsers.pdf.

[16] S. DENIER, H. A. SAHRAOUI. *Understanding the Use of Inheritance with Visual Patterns*, in "Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009), Lake Buena Vista, FL, USA", J. MILLER, R. SELBY (editors), IEEE Computer Society Press, 2009, http://rmod.lille.inria.fr/archives/papers/Deni09a-ESEM09-VisuInheritance.pdf CA .

[17] S. DUCASSE, M. DENKER, A. LIENHARD. *Evolving a Reflective Language*, in "Proceedings of the International Workshop on Smalltalk Technologies (IWST 2009), Brest, France", aug 2009, http://rmod.lille.inria.fr/archives/workshops/Duca09b-IWST09-TraitMop.pdf CL CH .

[18] S. DUCASSE, D. POLLET, A. BERGEL, D. CASSOU. *Reusing and Composing Tests with Traits*, in "Proceedings of the 47th International Conference Objects, Models, Components, Patterns (TOOLS-Europe'09), Zurich, Switzerland", jun 2009, p. 252–271, http://rmod.lille.inria.fr/archives/papers/Duca09a-Tools2009-TraitTests.pdf.

[19] J. L. FERNÁNDEZ, S. ROBLES, A. FORTIER, S. DUCASSE, G. ROSSI, S. GORDILLO. *Meteoroid Towards a real MVC for the Web*, in "Proceedings of International Workshop on Smalltalk Technologies (IWST 2009)", ACM Digital Library, 2009, http://rmod.lille.inria.fr/archives/workshops/Laut09a-IWST09-Meteroid.pdf, To appear AR .

[20] J. LAVAL, S. DENIER, S. DUCASSE, A. BERGEL. *Identifying cycle causes with Enriched Dependency Structural Matrix*, in "WCRE '09: Proceedings of the 2009 16th Working Conference on Reverse Engineering, Lille, France", 2009, http://rmod.lille.inria.fr/archives/papers/Lava09c-WCRE2009-eDSM.pdf.

[21] J. LAVAL, S. DENIER, S. DUCASSE, A. KELLENS. *Supporting Incremental Changes in Large Models*, in "Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2009), Brest, France", 2009, http://rmod.lille.inria.fr/archives/workshops/Lava09b-ESUG2009-incrementalChange.pdf BE .

[22] J. LAVAL, S. DENIER, S. DUCASSE. *Identifying cycle causes with CycleTable*, in "FAMOOSr 2009: 3rd Workshop on FAMIX and MOOSE in Software Reengineering, Brest, France", 2009, http://rmod.lille.inria.fr/archives/workshops/Lava09d-Famoosr2009-CycleTable.pdf.

[23] K. MORDAL-MANET, F. BALMAS, S. DENIER, S. DUCASSE, H. WERTZ, J. LAVAL, F. BELLINGARD, P. VAILLERGUES. *The Squale Model – A Practice-based Industrial Quality Model*, in "ICSM '09: Proceedings of the IEEE International Conference on Software Maintenance, Edmonton, Canada", 2009, p. 94–103, http://rmod.lille.inria.fr/archives/papers/Mord09a-ICSM2009-SqualeModel.pdf.

[24] A. RAZAVIZADEH, S. CÎMPAN, H. VERJUS, S. DUCASSE. *Multiple Viewpoints Architecture Extraction*, in "Proceedings of the 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architectures", 2009, p. 329–332, http://rmod.lille.inria.fr/archives/papers/Raza09a-WASECSA-MultipleViewPoints.pdf.

[25] A. RAZAVIZADEH, S. CÎMPAN, H. VERJUS, S. DUCASSE. *Software System Understanding via Architectural Views Extraction According to Multiple Viewpoints*, in "8th International Workshop on System/Software Architectures", 2009, http://rmod.lille.inria.fr/archives/workshops/Raza09b-iwssa-.pdf.

[26] D. RÖTHLISBERGER, O. NIERSTRASZ, S. DUCASSE, A. BERGEL. *Tackling Software Navigation Issues of the Smalltalk IDE*, in "Proceedings of International Workshop on Smalltalk Technologies (IWST 2009)", ACM Digital Library, 2009, http://rmod.lille.inria.fr/archives/workshops/Roet09e-IWST2009-obEnhancements.pdf CH .

[27] D. RÖTHLISBERGER, O. NIERSTRASZ, S. DUCASSE, D. POLLET, R. ROBBES. *Supporting Task-oriented Navigation in IDEs with Configurable HeatMaps*, in "Proceedings of the 17th International Conference on Program Comprehension (ICPC 2009), Los Alamitos, CA, USA", IEEE Computer Society, 2009, p. 253–257, http://rmod.lille.inria.fr/archives/papers/Roet09a-ICPC2009-HeatMaps.pdfCH.

[28] D. RÖTHLISBERGER, O. NIERSTRASZ, S. DUCASSE. *Autumn Leaves: Curing the Window Plague in IDEs*, in "Proceedings of the 16th Working Conference on Reverse Engineering (WCRE 2009), Los Alamitos, CA, USA", IEEE Computer Society, 2009, http://rmod.lille.inria.fr/archives/papers/Roet09f-WCRE2009-AutumnLeaves-ieee.pdf CH .

[29] T. VAN CUTSEM, A. BERGEL, S. DUCASSE, W. DE MEUTER. *Adding State and Visibility Control to Traits using Lexical Nesting*, in "Proceedings of ECOOP 2009, London, UK", S. DROSSOPOULOU (editor), Lecture Notes in Computer Science, Springer, 2009, http://rmod.lille.inria.fr/archives/papers/Cuts09a-ECOOP09-Traits.pdf BE .

National Peer-Reviewed Conference/Proceedings

[30] A. BERGEL. *Contrôler la visibilité des aspects avec Aspectboxes*, in "Proceedings of LMO'09, Toulouse, France", Cépaduès, 2009.

[31] J. LAVAL, A. BERGEL, S. DUCASSE. *Matrice de dépendances enrichie*, in "Proceedings of Languages et Modèles à Objets (LMO 2009), Nancy, France", 2009, http://rmod.lille.inria.fr/archives/papers/Lava09a-LMO2009-DSM.pdf.

Books or Proceedings Editing

[32] S. DENIER, T. GÎRBA (editors). *Proceedings of the 3rd Workshop on FAMIX and MOOSE in Software Reengineering (FAMOOSr 2009)*, 2009, 27, http://rmod.lille.inria.fr/archives/workshops/Deni09c-famoosr09-proceedings.pdf CH .

Research Reports

[33] F. BALMAS, F. BELLINGARD, S. DENIER, S. DUCASSE, J. LAVAL, K. MORDAL-MANET. *Practices in the Squale Quality Model (Squale Deliverable 1.3)*, INRIA, 2009, Technical report.

[34] F. BALMAS, A. BERGEL, S. DENIER, S. DUCASSE, J. LAVAL, K. MORDAL-MANET, H. ABDEEN, F. BELLINGARD. *Software metric for Java and C++ practices*, INRIA Lille Nord Europe, 2009, Technical report.

[35] S. DUCASSE, S. DENIER, F. BALMAS, A. BERGEL, J. LAVAL, K. MORDAL-MANET, F. BELLINGARD. *Software metric for Java and C++ practices*, INRIA Lille Nord Europe, 2009, Technical report.

[36] S. DUCASSE, S. DENIER, F. BALMAS, A. BERGEL, J. LAVAL, K. MORDAL-MANET, F. BELLINGARD. *Visualization of Practices and Metrics (Squale Deliverable 1.2)*, INRIA, 2009, Technical report.

[37] K. MORDAL-MANET, F. BALMAS, S. DENIER, S. DUCASSE, H. WERTZ, J. LAVAL, F. BELLINGARD, P. VAILLERGUES. *The Squale Model – A Practice-based Industrial Quality Model*, INRIA Lille Nord Europe, 2009, Technical report.

Scientific Popularization

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

[39] S. DUCASSE, L. RENGGLI, D. C. SHAFFER, R. ZACCONE, M. DAVIES. *Dynamic Web Development with Seaside*, Square Bracket Associates, 2009, http://book.seaside.st/ US CH .

Other Publications

[40] A. BERGEL, S. DENIER, S. DUCASSE, J. LAVAL, F. BELLINGARD, P. VAILLERGUES, F. BALMAS, K. MORDAL-MANET. *SQUALE – Software QUALity Enhancement*, in "13th European Conference on Software Maintenance and Reengineering (CSMR)", 2009, Presentation.

References in notes

[41] E. ARISHOLM, L. C. BRIAND, A. FOYEN. *Dynamic Coupling Measurement for Object-Oriented Software*, in "IEEE Transactions on Software Engineering", vol. 30, n^{o} 8, 2004, p. 491–506, http://csdl.computer.org/comp/trans/ts/2004/08/e0491abs.htmhttp://www.sce.carleton.ca/faculty/briand/pubs/simula_tr_20035.pdf.

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

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

[44] 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, http://scg.unibe.ch/archive/papers/Berg05bclassboxjOOPSLA.pdf.

[45] A. BERGEL, S. DUCASSE, O. NIERSTRASZ, R. WUYTS. *Stateful Traits*, in "Advances in Smalltalk — Proceedings of 14th International Smalltalk Conference (ISC 2006)", LNCS, vol. 4406, Springer, August 2007, p. 66–90, http://scg.unibe.ch/archive/papers/Berg07aStatefulTraits.pdf.

[46] A. BERGEL, S. DUCASSE, O. NIERSTRASZ, R. WUYTS. *Stateful Traits and their Formalization*, in "Journal of Computer Languages, Systems and Structures", vol. 34, n^{o} 2-3, 2008, p. 83–108, http://scg.unibe.ch/archive/papers/Berg07eStatefulTraits.pdf.

[47] 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)", vol. 38, October 2003, p. 47–64, http://scg.unibe.ch/archive/papers/Blac03aTraitsHierarchy.pdf.

[48] 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.

[49] 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", vol. 25, n^{o} 1, 1999, p. 91–121, http://dx.doi.org/10.1109/32.748920.

[50] 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", vol. 3, n^{o} 1, 1998, p. 65–117.

[51] 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.

[52] D. CAROMEL, J. VAYSSIÈRE. *A security framework for reflective Java applications*, in "Software: Practice and Experience", vol. 33, n^{o} 9, 2003, p. 821–846.

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

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

[55] 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.

[56] 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, http://www.emn.fr/x-info/obasco/events/jfdlpa04/.

[57] 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, vol. 4199, Springer-Verlag, 2006, p. 604–618, http://scg.unibe.ch/archive/papers/Duca06dMOOSEMODELS2006.pdf.

[58] 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, http://scg.unibe.ch/archive/papers/Duca06cDistributionMap.pdf.

[59] 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, 2005, p. 55–71, http://scg.unibe.ch/archive/papers/Duca05aMooseBookChapter.pdf.

[60] 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)", vol. 28, n^{o} 2, March 2006, p. 331–388, http://scg.unibe.ch/archive/papers/Duca06bTOPLASTraits.pdf.

[61] 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, http://scg.unibe.ch/archive/papers/Duca07b-FreezableTrait.pdf.

[62] 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.

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

[64] N. FENTON, S. L. PFLEEGER. *Software Metrics: A Rigorous and Practical Approach*, Second, International Thomson Computer Press, London, UK, 1996.

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

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

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

[68] L. GONG. *New security architectural directions for Java*, in "compcon", vol. 00, 1997, 97, http://dx.doi.org/10.1109/CMPCON.1997.584679.

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

[70] 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.

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

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

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

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

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

[76] 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, Ph. D. Thesis.

[77] 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.

[78] 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.

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

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

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

[82] 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.

[83] 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.

[84] 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.

[85] 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.

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

[87] 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.

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

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

[90] J. MCCALL, P. RICHARDS, G. WALTERS. *Factors in Software Quality*, NTIS Springfield, 1976.

[91] 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.

[92] M. S. MILLER. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control*, Johns Hopkins University, Baltimore, Maryland, USA, May 2006, Ph. D. Thesis.

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

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

[95] 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.

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

[97] D. POLLET, S. DUCASSE, L. POYET, I. ALLOUI, S. CÎMPAN, H. VERJUS. *Towards A Process-Oriented Software Architecture Reconstruction Taxonomy*, in "Proceedings of 11th European Conference on Software Maintenance and Reengineering (CSMR'07)", R. KRIKHAAR, C. VERHOEF, G. DI LUCCA (editors), IEEE

Computer Society, March 2007, http://scg.unibe.ch/archive/papers/Poll07a-CSMRSARTaxonomy.pdf, Best Paper Award.

[98] 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.

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

[100] 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.

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

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

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

[104] 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.

[105] 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, http://scg.unibe.ch/archive/papers/Scha04bOOEncapsulation.pdf.

[106] 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, vol. 2743, Springer Verlag, July 2003, p. 248–274, http://scg.unibe.ch/archive/papers/Scha03aTraits.pdf.

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

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

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

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

[111] 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.

[112] 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.

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