INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team rmod

Analyses and Languages Constructs for Object-Oriented Application Evolution

Lille - Nord Europe

Theme : Distributed Systems and Services

Table of contents

1. Team

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

Cyrille Delaunay [Associate Engineer from INRIA]

PhD Students

Jean-Baptiste Arnaud [Ministère de l'éducation et de la recherche scientifique]
Gwenael Casaccio [Bourse Région]
Aaron Jimenez Govea [Universidad de Guadalajara]
Jannik Laval [INRIA]
Nick Papoylias [Ecole des Mines de Douai]
Mariano Martinez Peck [Ecole des Mines de Douai]
Veronica Uquillas Gomez [Vrije Universiteit Brussels]

Post-Doctoral Fellows

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

Administrative Assistant

Marie-Bénédicte Dernoncourt [Secretary (SAR) Inria]

Others

Gabriel Hernán Barbuto [Internship (3 months)]
Adrien Barreau [Internship (3 months)]
Tristan Bourgois [Internship (3 months)]
Nicolas Paes [Internship (3 months)]
Martin Dias [Internship (3 months)]
Benjamin Van Ryseghem [Internship (3 months)]

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 4.1, our open-source reengineering platform, was released (http://moose.unibe.ch/)
- Pharo 1.1, a new open-source Smalltalk, was released (http://www.pharo-project.org) with accompanying book [37] (http://www.pharobyexample.org).
- CASCON High Impact Paper. A 1997 paper by Nicolas Anquetil (INRIA/RMod), Janice Singer (NSERC), Norman Vinson(NSERC), and Timothy Lethbridge (University of Canada) was selected as one of the "CASCON First Decade High Impact Papers" for the 20th edition of the CASCON international conference. Fourteen (14) papers, published between 1991 and 2000, were selected out of the 425 published in 10 years (acceptance rate=3,3%) on criteria including industrial impact and academic merit.
- Our new book “Dynamic web development with Seaside” http://book.seaside.st won the ESUG member 2010 best book award.

3. Scientific Foundations

[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.

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 [50], [48]. 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 [49]. The problem is that understanding large systems is already difficult [101], [73], [75], [68], 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 [68], [102]. 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 [49] 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 [69]. 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 [79]. Until now, few works have attempted to identify layers in practice: Mudpie [100] is a first cut at identifying cycles between packages as well as package groups potentially representing layers. DSM (dependency structure matrix) [99], [94] 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 [67]. 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 [103], [53].

Some approaches based on Formal Concept Analysis [98] 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 [63], [64] have been proposed to identify modules [74], [85]. 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 [72], [55], [60] such as the McCabe complexity metrics [80]. 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 [44] has been highly criticized [54], [62], [61], [31], [40], [41]. Other approaches have been proposed such as RFC and CBO [44] 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) [33], or some of metrics are not clearly specified (MCX, CCO, CCP, CRE) [72]. New cohesion measures were proposed [76], [88] 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 [54], [62], [61], [41]. 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 [31], [40]. 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 [67].

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 [81] 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 Standardization Organization) 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 [82], [90], [86] 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 [32] while other work focus on linking different high level criteria with software code metrics [77], [78].

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 [96], [51] and classboxes [34] 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 [96], [51]. 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 [47], [89], [35], [52] and several type systems where defined [56], [97], [91], [71].

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 [38]. 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 [51], stateful [36], and freezable [52]) 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 [87]. 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 [45], then from Smalltalk [58].

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 [66]). 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 [39], as well as controlling the access to reflective features [42], [43] 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] [84], [83], Java as well as classLoader strategies [70], [59].
- Categorize the different reflective features of languages such as CLOS [65], Python and Smalltalk [92] and identify suitable security mechanisms and infrastructure [57].
- Assess different security models (access rights, capabilities [93]...) 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 [39],
  - the use encapsulation policies as a basis to restrict the interfaces of the controlled objects [95],
  - 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...) [92].

4. Software

4.1. Moose

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

See also the web page http://www.moosetechnology.org/.

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 have been 150 publications (journal, international conferences, PhD theses) based on the Moose environment.

The first version running on top of Pharo (Moose 4.0) was released in June 2010, with a minor update (version 4.1) following end of September 2010.

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.

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

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*, was released in May 2010. A minor update release followed in June, and the 1.2 development branch has seen already over 230 incremental releases as of mid November 2010.

RMoD is the main maintainer and coordinator of Pharo. It is used widely in both research and industry.

4.3. VerveineJ

Participant: Nicolas Anquetil [correspondant].

See also the INRIA project https://gforge.inria.fr/projects/verveinej/.

VerveineJ is a tool to export Java project 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. A other similar tool, Infusion, already existed to fulfill the same needs, but it was closed sources and presented some errors that tainted the results we could obtain.

5. New Results

5.1. Package understanding and Assessing

Participants: Stéphane Ducasse, Damien Pollet, Nicolas Anquetil, Jannik Laval, Hani Abdeen.

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.

Package Fingerprint: a visual summary of package interfaces and relationships. Object-oriented languages such as Java, Smalltalk, and C++ structure their programs using packages. Maintainers of large systems need to understand how packages relate to each other, but this task is complex because packages often have multiple clients and play different roles (class container, code ownership...). Several approaches have been proposed, among which the use of cohesion and coupling metrics. Such metrics help identify candidate packages for restructuring; however, they do not help maintainers actually understand the structure and interrelationships between packages [12].

Objectives: Use pre-attentive processing as the basis for package visualization and see to what extent it could be used in package understanding.

Method: We present the package fingerprint, a 2D visualization of the references made to and from a package. The proposed visualization offers a semantically rich, but compact and zoomable views centered on packages. We focus on two views (incoming and outgoing references) that help users understand how the package under analysis is used by the system and how it uses the system.

Results: We applied these views on four large systems : Squeak, JBoss, Azureus, and ArgoUML. We obtained several interesting results, among which, the identification of a set of recurring visual patterns that help maintainers: (a) more easily identify the role of and the way a package is used within the system (e.g., the package under analysis provides a set of layered services), and, (b) detect either problematic situations (e.g., a single package that groups together a large number of basic services) or opportunities for better package restructuring (e.g., removing cyclic dependencies among packages). The visualization generally scaled well and the detection of different patterns was always possible.

Conclusion: The proposed visualizations and patterns proved to be useful in understanding and maintaining the different systems we addressed. To generalize to other contexts and systems, a real user study is required.

5.2. Tools Infrastructure

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

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

Supporting multiple source models efficiently. When reengineering software systems, maintainers should be able to assess and compare multiple change scenarios for a given goal, so as to choose the most pertinent one. Because they implicitly consider one single working copy, revision control systems do not scale up well to perform simultaneous analyses of multiple versions of systems. We designed Orion, an interactive prototyping tool for reengineering, to simulate changes and compare their impact on multiple versions of software source code models. Our approach offers an interactive simulation of changes, reuses existing assessment tools, and has the ability to hold multiple and branching versions simultaneously in memory. Specifically, we devise an infrastructure which optimizes memory usage of multiple versions for large models. This infrastructure uses an extension of the FAMIX source code meta-model but it is not limited to source code analysis tools since it can be applied to models in general. In the paper[27], we validate our approach by running benchmarks on memory usage and computation time of model queries on large models. Our benchmarks show that the Orion approach scales up well in terms of memory usage, while the current implementation could be optimized to lower its computation time. We also report on two large case studies on which we applied Orion.

Modeling Features at Runtime. A feature represents a functional requirement fulfilled by a system. Since many maintenance tasks are expressed in terms of features, it is important to establish the correspondence between a feature and its implementation in source code. Traditional approaches to establish this correspondence exercise features to generate a trace of runtime events, which is then processed by post-mortem analysis. These approaches typically generate large amounts of data to analyze. Due to their static nature, these approaches do not support incremental and interactive analysis of features. We propose a radically different approach called live feature analysis, which provides a model at runtime of features. Our approach analyzes features on a running system and also makes it possible to grow feature representations by exercising different scenarios of the same feature, and identifies execution elements even to the sub-method level. We describe how live feature analysis is implemented effectively by annotating structural representations of code based on abstract syntax trees. We illustrate our live analysis with a case study where we achieve a more complete feature representation by exercising and merging variants of feature behavior and demonstrate the efficiency or our technique with benchmarks. [17]

Domain-Specific Program Checking. Lint-like program checkers are popular tools that ensure code quality by verifying compliance with best practices for a particular programming language. The proliferation of internal domain-specific languages and models, however, poses new challenges for such tools. Traditional program checkers produce many false positives and fail to accurately check constraints, best practices, common errors, possible optimizations and portability issues particular to domain-specific languages. We advocate the use of dedicated rules to check domain-specific practices. We demonstrate the implementation of domain-specific rules, the automatic fixing of violations, and their application to two case-studies: (1) Seaside defines several internal DSLs through a creative use of the syntax of the host language; and (2) Magritte adds meta-descriptions to existing code by means of special methods. Our empirical validation demonstrates that domain-specific program checking significantly improves code quality when compared with general purpose program checking. [20]

Practical Dynamic Grammars for Dynamic Languages. Grammars for programming languages are traditionally specified statically. They are hard to compose and reuse due to ambiguities that inevitably arise. PetitParser combines ideas from scannerless parsing, parser combinators, parsing expression grammars and packrat parsers to model grammars and parsers as objects that can be reconfigured dynamically. Through examples and benchmarks we demonstrate that dynamic grammars are not only flexible but highly practical. [24]

5.3. Change support in the IDE

Participants: Stéphane Ducasse, Damien Pollet, Veronica Uquillas Gomez.

Replaying IDE Interactions. Change prediction helps developers by recommending program entities that will have to be changed alongside the entities currently being changed. To evaluate their accuracy, current change prediction approaches use data from versioning systems such as CVS or SVN. These data sources provide a coarse-grained view of the development history that flattens the sequence of changes in a single commit. They are thus not a valid basis for evaluation in the case of developmentstyle prediction, where the order of the predictions has to match the order of the changes a developer makes. We propose a benchmark for the evaluation of change prediction approaches based on fine-grained change data recorded from IDE usage. Moreover, the change prediction approaches themselves can use the more accurate data to fine-tune their prediction. We present an evaluation procedure and use it on several change prediction approaches, both novel and from the literature, and report on the results. [21]

Visually Supporting Source Code Changes Integration. Automatic and advanced merging algorithms help programmers to merge their modifications in main development repositories. However, there is little support to help release masters (integrators) to take decisions about the integration of published merged changes into the system release. Most of the time, the release master has to read all the changed code, check the diffs to build an idea of a change, and read unchanged code to understand the context of some changes. Such a task can be overwhelming. We present a dashboard to support integrators getting an overview of proposed changes in the context of object-oriented programming [23]. Our approach named Torch characterizes changes based on structural information, authors and symbolic information. It mixes text-based diff information with visual representation and metrics characterizing the changes. We describe our experiment applying it to Pharo, a large open-source system, and report on the evaluation of our approach by release masters of several open-source projects.

5.4. Traits

Participants: Stéphane Ducasse, Damien Pollet, Jannik Laval, Tristan Bourgois.

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.

Since traits offer an alternative to traditional inheritance-based code reuse, a couple of questions arise. For example, what is a good granularity for a Trait enabling reuse as well as plug ease? How much reuse can we expect on large existing inheritance-based hierarchies? We take as case study the Smalltalk Collection hierarchy and we start rewriting it from scratch using traits from the beginning [16]. We show how such library can be built using traits and we report such a preliminary experience. Since the Collection library is large, we focused and built the main classes of the library with Traits and report problems we encountered and how we solved them. Results of this experience are positive and show that we can build new collections based on the traits used to define the new library kernel.

5.5. Constructs for Reflective Secure Languages

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

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. 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 and modified. To address this issue, we propose Dynamic Read-Only references (DRO) that provide a view on an object where this object and its object graph are protected from modification. The read- only view dynamically propagates to aggregated objects, without changing the object graph itself; it acts as a read-only view of complex data structures, without making them read-only globally. We implement dynamic read-only references by using smart object proxies that lazily propagate the read-only view, following the object graph and driven by control flow and applied them to realize side-effect free assertions.

This work has been published at Tools 2010 [15].

5.6. Memory

Participants: Stéphane Ducasse, Marcus Denker, Mariano Martinez Peck.

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.

Visualizing Objects and Memory Usage. Most of the current garbage collector implementations work by reachability. This means they only take care of the objects that nobody else points to. As a consequence, there are objects which are not really used but are not garbage collected because they are still referenced. Such unused but reachable objects create memory leaks. This is a problem because applications use much more memory than what is actually needed. In addition, they may get slower and crash. It is important to understand which parts of the system are instantiated but also which are used or unused. There is a plethora of work on runtime information or class instantiation visualizations but none of them show whether instances are actually used. Such information is important to identify memory leaks. In this paper, we present some visualizations that show used/unused objects in object-oriented applications. For this, we use Distribution Map which is a visualization showing spread and focus of properties across systems. We extend Distribution Maps to represent the way classes are used or not [19], since we distinguish between a class that just has instances from one that has used instances. To identify unused objects, we modified the Pharo Virtual Machine.

Experiments with a Fast Object Swapper. In object-oriented systems, runtime memory is composed of an object graph in which objects refer to other objects. This graph of objects evolves while the system is running. Graph exporting and swapping are two important object graph operations. Exporting refers to copying the graph to some other memory so that it can be loaded by another system. Swapping refers to moving the graph to a secondary memory (e.g., a hard disk) to temporary release part of the primary memory. While exporting and swapping are achieved in different ways, each of them faces a common and central problem which is the speed of the approach in presence of large object graphs. Nevertheless, most of the existing solutions do not address well this issue. Another challenge is to deal with extremely common situations where objects outside the exported/swapped graph point to objects inside the graph. To correctly load back an exported subgraph, it is necessary to compute and export extra information that is not explicit in the object subgraph. This extra information is needed because certain objects may require to be reinitialized or recreated, to run specific code before or after the loading, to be updated to a new class definition, etc. In a worksho paper, we discuss most of the general problems of object exporting and swapping [18]. As a case of study, we present an analysis of ImageSegment, a fast solution to export and swap object graphs, developed by Dan Ingalls. ImageSegment addresses the speed problems in an efficient way, as shown by the results of several benchmarks we have conducted using Pharo Smalltalk. However, ImageSegment is not a panacea since it still has other problems that hampers its general use.

6. Other Grants and Activities

6.1. National Initiatives

6.1.1. 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.

6.1.2. Cutter ANR Project

Participants: Stéphane Ducasse [correspondant], Nicolas Anquetil, Damien Pollet.

Cutter was just accepted, as of December 2010. It is funded by ANR; participants are RMoDand the D'Oc–APR group 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.

6.2. European Initiatives

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

6.2.1. 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.

6.2.2. 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.

6.2.3. 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 remodularization of object oriented systems. The collaboration started in 2008 and is now in its third and final year.

Results 2010.

- We released a second version of a Moose based on FAMIX30 and reimplemented on top of Pharo during 2010. The University of Bern and the RMoD research groups continue to develop Moose.

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

- International workshop: FAMOOSr. We organized International 4th Workshop on FAMIX and Moose in Reengineering held in conjunction with the International Working Conference on Reverse Engineering (WCRE 2010). The official website for this event is http://www.moosetechnology.org/events/famoosr2010 (17 September 2010, Timisoara, Romania).
- International seminars: SATTOSE. This year, SATTOSE was held from 26th to 29th of April in Clapiers, a small village next to Montpellier. 22 researchers attended the seminar.

Publication production. This year saw the release of one book by authors of the REMOOSE associated team via on-demand printing.

- Seaside Book [26]. *Dynamic Web Development with Seaside*. A free online version remains to be available.

In addition we collaborated on several joint publications. Here is the list of joint papers of 2010:

- Marcus Denker, Jorge Ressia, Orla Greevy, and Oscar Nierstrasz. Modeling Features at Runtime. In Proceedings of MODELS 2010 Part II, LNCS 6395 p. 138–152, Springer-Verlag, 2010.
- Lukas Renggli, Stéphane Ducasse, Tudor Gîrba, and Oscar Nierstrasz. Practical Dynamic Grammars for Dynamic Languages. In 4th Workshop on Dynamic Languages and Applications (DYLA 2010), Malaga, Spain, 2010.
- Lukas Renggli, Stéphane Ducasse, Tudor Gîrba, and Oscar Nierstrasz. Domain-Specific Program Checking. In Jan Vitek (Ed.), Proceedings of the 48th International Conference on Objects, Models, Components and Patterns (TOOLS'10), LNCS 6141 p. 213–232, Springer-Verlag, 2010.
- Adrian Kuhn, David Erni, and Marcus Denker. Empowering Collections with Swarm Behavior. Technical report arXiv:1007.0159, Arxiv, 2010.

6.3. International Initiatives

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

6.3.1. 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).

Publications 2010:

- Gabriela Arévalo, Stéphane Ducasse, Silvia Gordillo, and Oscar Nierstrasz. Generating a catalog of unanticipated schemas in class hierarchies using Formal Concept Analysis. In Information and Software Technology 52 p. 1167–1187, 2010.
- Jean-Baptiste Arnaud, Marcus Denker, Stéphane Ducasse, Damien Pollet, Alexandre Bergel, and Mathieu Suen. Read-Only Execution for Dynamic Languages. In Proceedings of the 48th International Conference Objects, Models, Components, Patterns (TOOLS-Europe'10), Malaga, Spain, 2010.

- Romain Robbes, Damien Pollet, and Michele Lanza. Replaying IDE Interactions to Evaluate and Improve Change Prediction Approaches. In Jim Whitehead and Thomas Zimmermann (Ed.), 7th IEEE Working Conference on Mining Software Repositories (MSR), p. 161–170, IEEE Computer Society, 2010.

Marcus Denker visited Argentina November 3 to November 5th, 2010.

6.4. 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, D. Girba, J. Ressia, T. Verwaest and C. Bruni). This leads to the production of several research articles (see below).

For STICAmsud, the RMOD team hosted A. Bergel and J. Fabry from Chile and G. Arevalo from Argentina.

7. Dissemination

7.1. Animation of the scientific community

7.1.1. 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-Rémy Falleri, 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*, Émilie Balland, Université de Nancy, France. 11/03/08 (referee).
- *Of Change and Software*, Romain Robbes, Université de Lugano, Suisse. 1/12/08 (referee).

7.1.2. Scientific Community Animation

Stéphane Ducasse has been/is :

- Member of the following committees:
  - European Conference on Object-Oriented Programming (ECOOP).
  - International Conference on Software Maintenance (ICSM).
  - International Conference on the Unified Modeling Language (Models).
  - International Conference on Objects, Models, Components, Patterns (TOOLS).
  - International Conference on Software Composition (SC).
  - Program Chair of the International Smalltalk Conference.
- 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 :

- Reviewer for journal: IEEE Transactions on software Engineering (2010).
- Reviewer for conferences: WCRE, ICSM, CSMR
- Recipient of the CASCON First Decade High Impact paper award:
  - 14 papers were selected out of the 425 published at CASCON (IBM's Center for Advanced Studies Conference) between 1991 and 2000 (acceptance rate=3,3%).
- Co-organizer for the *Journées Ingénierie dirigée par les modèles* (Dec 2010).
- Co-organizer for the Benevol 2010 workshop.

Marcus Denker has been/is :

- PC Chair Smalltalks 2010
- Member of the following program committees:
  - ECOOP Workshop on Reflection, AOP, and Meta-Data for Software Evolution (RAM-SE 2010).
  - ESUG International Smalltalk Workshop (IWST10).
  - 2nd Workshop on Self-sustaining Systems (S3 2010).
  - 9th Annual Aspect-Oriented Software Development Conference (AOSD 2011).
- 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 :

- PC member for Smalltalks 2010.
- Reviewer for conferences: ICPC, ICSM, Models, Software Composition, Tools.
- Co-organizer for the *Journées Ingénierie dirigée par les modèles* (Dec 2010).
- Co-organizer for the Benevol 2010 workshop.
- Maintainer of the LaTeX style files for submissions to the Journal of Object Technology.

Simon Denier has been/is :

- Reviewer for journal: Elsevier Science of Computer Programming (2010).
- PC member for the Tool Demonstration track at the International Conference on Software Maintenance (ICSM 2010).
- PC member for Software Composition (SC 2010).
- PC member for the 5th International Workshop on Program Comprehension through Dynamic Analysis (PCODA 2010).
- PC member for Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2010).
- External reviewer for journals: IEEE Software (special issue on Software Evolution, July/August 2010).
- External reviewer for conferences: ECOOP 2010.
- Co-organiser for FAMOOSr 2010 and DYLA 2010.

7.2. Teaching

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.

8. 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^{o} 5, p. 515–529 [*DOI :* 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^{o} 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^{o} 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^{o} 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^{o} 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^{o} 5, p. 56–63, http://www.computer.org/portal/web/csdl/doi/10.1109/MS.2007.144.

[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^{o} 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] H. FERNANDES. *iStoa, modèle notionnel de guidage macroscopique de l'apprentissage*, Université de Lille, 2010, http://rmod.lille.inria.fr/archives/phd/PhD-2010-Fernandes.pdf.

Articles in International Peer-Reviewed Journal

[12] H. ABDEEN, S. DUCASSE, D. POLLET, I. ALLOUI. *Package Fingerprint: a visual summary of package interfaces and relationships*, in "Information and Software Technology Journal", 2010, vol. 52, p. 1312-1330 [*DOI :* 10.1016/J.INFSOF.2010.07.005], http://rmod.lille.inria.fr/archives/papers/Abde10a-IST-Official-packageFingerprints.pdf.

[13] G. ARÉVALO, S. DUCASSE, S. GORDILLO, O. NIERSTRASZ. *Generating a catalog of unanticipated schemas in class hierarchies using Formal Concept Analysis*, in "Information and Software Technology", December 2010, vol. 52, p. 1167-1187 [*DOI :* 10.1016/J.INFSOF.2010.05.010], http://rmod.lille.inria.fr/archives/papers/Arev10a-IST-generating_a_catalog.pdf.

[14] S. DUCASSE, S. DENIER, D. POLLET, I. ALLOUI, H. ABDEEN, J.-R. FALLERI. *Understanding Packages: The Package Blueprint*, in "IEEE Transactions on Software Engineering (TSE) (Accepted with major revisions)", August 2010.

International Peer-Reviewed Conference/Proceedings

[15] J.-B. ARNAUD, M. DENKER, S. DUCASSE, D. POLLET, A. BERGEL, M. SUEN. *Read-Only Execution for Dynamic Languages*, in "Proceedings of the 48th International Conference Objects, Models, Components, Patterns (TOOLS-Europe'10)", Malaga, Spain, June 2010, http://rmod.lille.inria.fr/archives/papers/Arna10a-Tools2010-ReadOnlyExecutionForDynamicLanguages.pdf.

[16] T. BOURGOIS, J. LAVAL, S. DUCASSE, D. POLLET. *BLOC: a Trait-Based Collections Library - a Preliminary Experience Report*, in "Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST 2010)", Barcelona, Spain, 2010, http://rmod.lille.inria.fr/archives/workshops/Bour10a-IWST10-bloc.pdf.

[17] M. DENKER, J. RESSIA, O. GREEVY, O. NIERSTRASZ. *Modeling Features at Runtime*, in "Proceedings of MODELS 2010 Part II", LNCS, Springer-Verlag, October 2010, vol. 6395, p. 138–152 [*DOI :* 10.1007/978-3-642-16129-2_11], http://rmod.lille.inria.fr/archives/papers/Denk10a-Models10-FeatureModels.pdf.

[18] M. MARTINEZ PECK, N. BOURAQADI, M. DENKER, S. DUCASSE, L. FABRESSE. *Experiments with a Fast Object Swapper*, in "Smalltalks 2010", 2010, http://rmod.lille.inria.fr/archives/workshops/Mart10b-Smalltalks2010-Swapper-ImageSegments.pdf.

[19] M. MARTINEZ PECK, N. BOURAQADI, M. DENKER, S. DUCASSE, L. FABRESSE. *Visualizing Objects and Memory Usage*, in "Smalltalks 2010", 2010, http://rmod.lille.inria.fr/archives/workshops/Mart10a-Smalltalks2010-VisualizingUnusedObjects.pdf.

[20] L. RENGGLI, S. DUCASSE, T. GÎRBA, O. NIERSTRASZ. *Domain-Specific Program Checking*, in "Proceedings of the 48th International Conference on Objects, Models, Components and Patterns (TOOLS'10)", J. VITEK (editor), LNCS, Springer-Verlag, 2010, vol. 6141, p. 213–232 [*DOI :* 10.1007/978-3-642-13953-6_12], http://rmod.lille.inria.fr/archives/papers/Reng10b-DomainSpecificProgramChecking.pdf.

[21] R. ROBBES, D. POLLET, M. LANZA. *Replaying IDE Interactions to Evaluate and Improve Change Prediction Approaches*, in "7th IEEE Working Conference on Mining Software Repositories (MSR)", J. WHITEHEAD, T. ZIMMERMANN (editors), IEEE Computer Society, May 2010, p. 161–170 [*DOI : 10.1109/MSR.2010.5463278*], http://rmod.lille.inria.fr/archives/papers/Robb10a-MSR10-ChangePrediction.pdf.

[22] V. UQUILLAS GÓMEZ, S. DUCASSE, T. D'HONDT. *Meta-models and Infrastructure for Smalltalk Omnipresent History*, in "Smalltalks'2010", 2010, http://rmod.lille.inria.fr/archives/workshops/Uqui10b-Smalltalk2010-Metamodels.pdf.

[23] V. UQUILLAS GÓMEZ, S. DUCASSE, T. D'HONDT. *Visually Supporting Source Code Changes Integration: the Torch Dashboard*, in "Working Conference on Reverse Engineering (WCRE 2010)", October 2010, http://rmod.lille.inria.fr/archives/papers/Uqui10a-Torch-WCRE10.pdf.

Workshops without Proceedings

[24] L. RENGGLI, S. DUCASSE, T. GÎRBA, O. NIERSTRASZ. *Practical Dynamic Grammars for Dynamic Languages*, in "4th Workshop on Dynamic Languages and Applications (DYLA 2010)", Malaga, Spain, June 2010, http://scg.unibe.ch/archive/papers/Reng10cDynamicGrammars.pdf.

Research Reports

[25] A. KUHN, D. ERNI, M. DENKER. *Empowering Collections with Swarm Behavior*, Arxiv, July 2010, n° arXiv:1007.0159, http://rmod.lille.inria.fr/archives/reports/Kuhn10a-ArXiv-SwarmBehavior.pdf.

Scientific Popularization

[26] S. DUCASSE, L. RENGGLI, C. D. SHAFFER, R. ZACCONE, M. DAVIES. *Dynamic Web Development with Seaside*, Square Bracket Associates, 2010, http://book.seaside.st/book.

[27] J. LAVAL, S. DENIER, S. DUCASSE, J.-R. FALLERI. *Supporting Simultaneous Versions for Software Evolution Assessment*, in "Journal of Science of Computer Programming (SCP)", May 2010, http://dx.doi.org/10.1016/j.scico.2010.11.014.

[28] J. LAVAL, S. DUCASSE. *Optimisation d'applications en Pharo*, January 2010, vol. 1, n° 123.

[29] J. LAVAL, S. DUCASSE. *Package et gestion de versions en Pharo*, November 2010, vol. 1, n° 132.

[30] J. LAVAL, S. DUCASSE. *Pharo: un nouveau Smalltalk open source*, September 2010, vol. 1, n° 130.

References in notes

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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 [*DOI :* 10.1007/978-3-540-71836-9_3], http://scg.unibe.ch/archive/papers/Berg07aStatefulTraits.pdf.

[36] 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 [*DOI :* 10.1016/J.CL.2007.05.003], http://scg.unibe.ch/archive/papers/Berg07eStatefulTraits.pdf.

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

[38] 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.

[39] 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.

[40] 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° 1, p. 91–121, http://dx.doi.org/10.1109/32.748920.

[41] 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° 1, p. 65–117.

[42] 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.

[43] D. CAROMEL, J. VAYSSIÈRE. *A security framework for reflective Java applications*, in "Software: Practice and Experience", 2003, vol. 33, n° 9, p. 821–846 [*DOI :* 10.1002/SPE.528].

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

[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. 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

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

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

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

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

[61] 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.

[62] 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.

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

[64] 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.

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

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

[67] 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.

[68] 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.

[69] 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.

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

[71] 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.

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

[73] 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.

[74] 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.

[75] 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.

[76] 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.

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

[78] 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.

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

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

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

[82] 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.

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

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

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

[90] 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.

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

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

[93] 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.

[94] 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.

[95] 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.

[96] 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.

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

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

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

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

[101] 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.

[102] 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 [*DOI :* 10.1109/VISSOF.2007.4290706].

[103] 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.