Project-Team RMOD 2022 Activity Report

Marcus Denker, Nicolas Anquetil, Steven Costiou, Stéphane Ducasse, Anne Etien, Guillermo Polito

▶ **To cite this version:**

Marcus Denker, Nicolas Anquetil, Steven Costiou, Stéphane Ducasse, Anne Etien, et al.. Project-Team RMOD 2022 Activity Report. Inria Lille - Nord Europe. 2022. hal-04112164

HAL Id: hal-04112164
https://inria.hal.science/hal-04112164

Submitted on 31 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

RESEARCH CENTRE

Inria Center at the University of Lille

IN PARTNERSHIP WITH:

Université de Lille

2022

ACTIVITY REPORT

Project-Team

RMOD

Analyses and Languages Constructs for Object-Oriented Application Evolution

IN COLLABORATION WITH: Centre de Recherche en Informatique, Signal et Automatique de Lille

DOMAIN

Networks, Systems and Services, Distributed Computing

THEME

Distributed programming and Software engineering

Inria

Contents

Project-Team RMOD

Creation of the Project-Team: 2009 July 01

Keywords

Computer sciences and digital sciences

A1.3.3. – Blockchain

A2. – Software

A2.1. – Programming Languages

A2.1.3. – Object-oriented programming

A2.1.8. – Aspect-oriented programming

A2.1.10. – Domain-specific languages

A2.1.12. – Dynamic languages

A2.3.1. – Embedded systems

A2.5. – Software engineering

A2.5.1. – Software Architecture & Design

A2.5.3. – Empirical Software Engineering

A2.5.4. – Software Maintenance & Evolution

A2.6. – Infrastructure software

A2.6.3. – Virtual machines

Other research topics and application domains

B2. – Health

B2.7. – Medical devices

B5. – Industry of the future

B5.9. – Industrial maintenance

B6.5. – Information systems

B7. – Transport and logistics

1 Team members, visitors, external collaborators

Research Scientists

- Stephane Ducasse [Team leader, INRIA, Senior Researcher, HDR]
- Steven Costiou [INRIA, Researcher]
- Marcus Denker [INRIA, Researcher]
- Guillermo Polito [INRIA, Researcher, from Oct 2022]

Faculty Members

- Nicolas Anquetil [UNIV LILLE, Associate Professor, HDR]
- Vincent Aranega [UNIV LILLE, Associate Professor, On Leave since 08/22]
- Anne Etien [UNIV LILLE, Professor, HDR]

Post-Doctoral Fellow

- Larisa Safina [INRIA, from Dec 2022]

PhD Students

- Nour Jihene Agouf [Arolla SAS, CIFRE]
- Valentin Bourcier [INRIA, from Oct 2022]
- Santiago Bragagnolo [BERGER-LEVRAULT, CIFRE]
- Gabriel Darbord [INRIA, from Oct 2022]
- Mahugnon Honore Houekpetodji [Cim, until Jun 2022, CIFRE]
- Aless Hosry [INRIA]
- Pierre Misse-Chanabier [INRIA]
- Nahuel Palumbo [INRIA, from Dec 2022]
- Théo Rogliano [INRIA, until Sep 2022]
- Iona Thomas [INRIA]
- Maximilian Ignacio Willembrinck Santander [INRIA]
- Oleksandr Zaitsev [Arolla SAS]

Technical Staff

- Christophe Demarey [INRIA, Engineer, 60%]
- Sebastian Jordan Montano [INRIA, Engineer, until Sep 2022]
- Soufyane Labsari [INRIA, Engineer]
- Esteban Lorenzano [INRIA, Engineer, Pharo Consortium]
- Milton Mamani Torres [INRIA, Engineer]
- Hernan Federico Morales [INRIA, Engineer, from Dec 2022]

- Nahuel Palumbo [INRIA, Engineer, until Nov 2022]
- Guillermo Polito [CNRS, Engineer, until Sep 2022]
- Abderrahmane Seriai [BERGER-LEVRAULT, Engineer]
- Pablo Tesone [INRIA, Engineer, Pharo Consortium]
- Clotilde Toullec [INRIA, Engineer]
- Benoît Verhaeghe [BERGER-LEVRAULT, Engineer]

Interns and Apprentices

- Abir Bezzazi [UNIV LILLE, Intern, from Apr 2022 until Aug 2022, L3]
- Daniel Camacho Santa Cruz [UNIV of Cochabamba, Bolivia, Intern, from Apr 2022]
- Romain Degrave [UNIV Lille, Intern, from Apr 2022 until Aug 2022, L3]
- Antoine Delaby [UNIV LILLE, Intern, from Apr 2022 until Jul 2022, L3]
- Gaylord Delporte [UNIV LILLE, Intern, from Apr 2022 until Aug 2022]
- Ibrahima Diallo [UNIV LILLE, Intern, from Apr 2022 until Jun 2022]
- Aboubacar Diawara [UNIV LILLE, Intern, from Apr 2022 until Jun 2022]
- Remi Dufloer [UNIV LILLE, Intern, from Apr 2022 until Aug 2022]
- Mohamed Jedny [INRIA LILLE, Intern, from Apr 2022 until Jul 2022, Master]
- Sebastian Jordan Montano [INRIA, Apprentice, from Sep 2022]
- Nicolas Rainhart [UNIV LILLE, Intern, from Apr 2022 until Aug 2022]
- Younoussa Sow [INRIA, until Jan 2022, relai these]
- Adrien Vanegue [INRIA, Apprentice, from Apr 2022, Intern before starting Aprenticeship]

Administrative Assistant

- Aurore Dalle [INRIA]

Visiting Scientists

- Gordana Rakic [Université de Novi Sad - Serbie, from Dec 2022]
- Gabriel Cavalheiro Ullmann [Concordia University, from Oct 2022 until Nov 2022]
- Domenico Cipriani [Independent Artist, from Mar 2022 until Mar 2022]
- Quentin Ducasse [École Nationale Supérieure de Techniques Avancées, from Sep 2022 until Sep 2022]
- Michele Lanza [USI, Switzerland, from Mar 2022 until Mar 2022]
- Kasper Osterbye [Independent, from Feb 2022 until Feb 2022]
- Jean Privat [Université du Québec à Montréal, from Sep 2022 until Nov 2022, Sabatical]
- Balsa Sarenac [University of Novi Sad. Serbia, from Sep 2022 until Oct 2022]
- Quentin Stievenart [Université Libre de Bruxelles, from Sep 2022 until Sep 2022]
- Max Zurbriggen [University of Zürich, from Oct 2022 until Nov 2022]

2 Overall objectives

2.1 Introduction

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

2.2 Reengineering and remodularization

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

How to help remodularize existing software applications?

We are developing analyses and algorithms to remodularize object-oriented applications. This is why we started studying and building tools to support the *understanding of applications* at the level of packages and modules. This allows us to understand the results of the *analyses* that we are building.

We seek to create tools to help developers perform large refactoring. How can they keep track of changes in various locations in a system while ensuring *integrity of current and new code* by *uniformally applying new design choices*.

2.3 Constructs for modular and isolating programming languages

Dynamically-typed programming languages such as JavaScript are getting new attention as illustrated by the large investment of Google in the development of the Chrome V8 JavaScript engine and the development of a new dynamic language DART. This new trend is correlated to the increased adoption of dynamic programming languages for web-application development, as illustrated by Ruby on Rails, PHP and JavaScript. With web applications, users expect applications to be always available and getting updated on the fly. This continuous evolution of application is a real challenge [55]. Hot software evolution often requires *reflective* behavior and features. For instance in CLOS and Smalltalk each class modification automatically migrates existing instances on the fly.

At the same time, there is a need for *software isolation*, i.e., applications should reliably run co-located with other applications in the same virtual machine with neither confidential information leaks nor vulnerabilities. Indeed, often for economical reasons, web servers run multiple applications on the same virtual machine. Users need confined applications. It is important that (1) an application does not access information of other applications running on the same virtual machine and (2) an application authorized to manipulate data cannot pass such authorization or information to other parts of the application that should not get access to it.

Static analysis tools have always been confronted to reflection [52]. Without a full treatment of reflection, static analysis tools are both incomplete and unsound. Incomplete because some parts of the program may not be included in the application call graph, and unsound because the static analysis does not take into account reflective features [61]. In reflective languages such as F-Script, Ruby, Python, Lua, JavaScript, Smalltalk and Java (to a certain extent), it is possible to nearly change any aspect of an application: change objects, change classes dynamically, migrate instances, and even load untrusted code.

Reflection and isolation concerns are a priori antagonistic, pulling language design in two opposite directions. Isolation, on the one hand, pulls towards more static elements and types (e.g., ownership types). Reflection, on the other hand, pulls towards fully dynamic behavior. This tension is what makes this a real challenge: As experts in reflective programming, dynamic languages and modular systems, we believe that by working on this important tension we can make a breakthrough and propose innovative

solutions in resolving or mitigating this tension. With this endeavor, we believe that we are working on a key challenge that can have an impact on future programming languages. The language construct challenge tackled by RMoD is formulated as follows:

What are the language modularity constructs to support isolation?

In parallel we are continuing 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, we dedicate efforts to remodularizing a meta-level architecture in the context of the design of an isolating dynamic language. Indeed at the extreme, modules and structural control of reflective features are the first steps towards flexible, dynamic, yet isolating, 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.

3 Research program

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 programming 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 be selected for a given remodularization?

To help us answer these questions, we work on enriching Moose, our reengineering environment, with a new set of analyses [45, 46]. We decompose our approach in three main and potentially overlapping steps:

1. Tools for understanding applications
2. Remodularization analyses
3. Software Quality

3.1.1 Tools for understanding applications

Context and Problems We are studying the problems raised by the understanding of applications at a larger level of granularity such as packages or modules. We want to develop a set of conceptual tools to support this understanding.

Some approaches based on Formal Concept Analysis (FCA) [74] show that such an analysis can be used to identify modules. However the presented examples are too small and not representative of real code.

Research Agenda FCA provides an important approach in software reengineering for software understanding, design anomalies detection and correction, but it suffers from two problems: (i) it produces lattices that must be interpreted by the user according to his/her understanding of the technique and different elements of the graph; and, (ii) the lattice can rapidly become so big that one is overwhelmed by the mass of information and possibilities [35]. We look for solutions to help people putting FCA to real use.

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

3.1.2 Remodularization analyses

Context and Problems It is a well-known practice to layer applications with bottom layers being more stable than top layers [62]. Until now, few works have attempted to identify layers in practice: Mudpie [76] is a first cut at identifying cycles between packages as well as package groups potentially representing layers. DSM (dependency structure matrix) [70, 75] 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 [58]. 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 [49, 77].

As we are designing and evaluating algorithms and analyses to remodularize applications, we also need a way to understand and assess the results we are obtaining.

Research Agenda We work on the following items:

- **Layer identification:** We propose an approach to identify layers based on a semi-automatic classification of package and class interrelationships that they contain. However, taking into account the wish or knowledge of the designer or maintainer should be supported.
- **Cohesion Metric Assessment:** We are building a validation framework for cohesion/coupling metrics to determine whether they actually measure what they promise to. We are also compiling a number of traditional metrics for cohesion and coupling quality metrics to evaluate their relevance in a software quality setting.

3.1.3 Software Quality

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

- **Quality models:** We studied existing quality models and the different options to combine indicators — often, software quality models happily combine metrics, but at the price of losing the explicit relationships between the indicator contributions. There is a need to combine the results of one metric over all the software components of a system, and there is also the need to combine different metric results for any software component. Different combination methods are possible that can give very different results. It is therefore important to understand the characteristics of each method.
- **Bug prevention:** Another aspect of software quality is validating or monitoring the source code to avoid the emergence of well known sources of errors and bugs. We work on how to best identify such common errors, by trying to identify earlier markers of possible errors, or by helping identifying common errors that programmers did in the past.

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 [47, 72] and classboxes [36] but also start to work on new areas such as isolation in dynamic languages. We will work on the following points: (1) Traits and (2) Modularization as a support for isolation.

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 [47, 72]. 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 [37, 44, 48, 66] and several type systems were defined [50, 60, 67, 73].

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 both to use for software developers and to implement for language designers.

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 [39]. 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 [47], stateful [38], and freezable [48]) and their operators. We will compare these models by (1) implementing a trait-based collection hierarchy, (2) analyzing several existing applications that exhibit the need for traits. Traits may be flattened [65]. This is a fundamental property that confers to traits their simplicity and expressiveness over Eiffel's multiple inheritance. Keeping these aspects is one of our priority in forthcoming enhancements of traits.

- Alternative trait models. This work revisits the problem of adding state and visibility control to traits. Rather than extending the original trait model with additional operations, we use a fundamentally different approach by allowing traits to be lexically nested within other modules. This enables traits to express (shared) state and visibility control by hiding variables or methods in their lexical scope. Although the traits' "flattening property" no longer holds when they can be lexically nested, the combination of traits with lexical nesting results in a simple and more expressive trait model. We formally specify the operational semantics of this combination. Lexically nested traits are fully implemented in AmbientTalk, where they are used among others in the development of a Morphic-like UI framework.

- We want to evaluate how inheritance can be replaced by traits to form a new object model. For this purpose we will design a minimal reflective kernel, inspired first from ObjVlisp [43] then from Smalltalk [53].

3.2.2 Reconciling Dynamic Languages and Isolation

Context and Problems More and more applications require dynamic behavior such as modification of their own execution (often implemented using reflective features [57]). 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 or broken code, which may be problematic for the system if the application is not properly isolated. Bytecode checking and static code analysis are used to enable isolation, but such approaches do not really work in presence of dynamic languages and reflective features. Therefore there is a tension between the need for flexibility and isolation.

Research Agenda: Isolation in dynamic and reflective languages To solve this tension, we will work on *Sure*, a language where isolation 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 [40], as well as controlling the access to reflective features [41, 42] are important challenges. We plan to:

- Study the isolation abstractions available in erights (www.erights.org) [63, 64], and Java's class loader strategies [54, 59].
- Categorize the different reflective features of languages such as CLOS [56], Python and Smalltalk [68] and identify suitable isolation mechanisms and infrastructure [51].
- Assess different isolation models (access rights, capabilities [69] etc.) 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 [40],
  - the use of encapsulation policies as a basis to restrict the interfaces of the controlled objects [71],
  - the definition of method modifiers to support controlling encapsulation in the context of dynamic languages.

An open question is whether, instead of providing restricted interfaces, we could use traits to grant additional behavior to specific instances: without trait application, the instances would only exhibit default public behavior, but with additional traits applied, the instances would get extra behavior. We will develop *Sure*, a modular extension of the reflective kernel of Smalltalk (since it is one of the languages offering the largest set of reflective features such as pointer swapping, class changing, class definition etc.) [68].

4 Application domains

4.1 Programming Languages and Tools

Many of the results of RMoD are improving programming languages or development tools for such languages. As such the application domain of these results is as varied as the use of programming languages in general. Pharo, the language that RMoD develops, is used for a very broad range of applications. From pure research experiments to real world industrial use (the Pharo Consortium has more than 25 company members).

Examples are web applications, server backends for mobile applications or even graphical tools and embedded applications

4.2 Software Reengineering

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. As such Moose is used for analyzing software systems to support understanding and continuous development as well as software quality analysis.

5 Social and environmental responsibility

5.1 Footprint of research activities

The main environmental footprint of RMoD is related to international travel. Meeting researchers in person is indispensable for research.

We try to reduce travel by using online meetings as much as possible. The team tries to reduce impact of daily local travel by the use of local transport and biking to work.

5.2 Impact of research results

Our work on language runtimes has potential impact to reduce energy consumption.

Reengineering can be understood as a kind of *"recycling"*. Our tools allow companies to use systems for a longer time, reducing environmental impact of software that is created as a new project.

All software we develop as part of our research is released as Open Source, all our publications are available in the HAL archive.

6 Highlights of the year

6.1 Awards

- Best paper award at VISSOFT 2022: *A New Generation of Class Blueprint* [9] (Nour Jihene Agouf)
- ESUG 2022 Innovation Award. (Maximilian Ignacio Willembrinck Santander)
- Best poster award at Journées Nationales du GDR-GPL 2022 for the poster *Time-Traveling Queries for Faster Debugging and Program Comprehension* [34](Maximilian Ignacio Willembrinck Santander)
- Second place for the IWST 22 best paper awards: *How Fast is AI in Pharo? Benchmarking Linear Regression* [23] (Sebastian Jordan Montano and Oleksandr Zaitsev)

6.2 Highlights

- Recruitment of Guillermo Polito as CR.
- We released Pharo10: Pharo.org
- The paper *Interpreter-guided Differential JIT Compiler Unit Testing* [17] was published at PLDI'22 (rank A*)
- The paper "Heap Fuzzing: Automatic Garbage Collection Testing with Expert-Guided Random Events" was accepted in ICST (rank A, to be published in 2023)

7 New software and platforms

7.1 New software

7.1.1 Moose

Name: Moose: Software and Data Analysis Platform

Keywords: Software engineering, Meta model, Software visualisation

Functional Description: Moose is an extensive platform for software and data analysis. It offers multiple services ranging from importing and parsing data, to modeling, to measuring, querying, mining, and building interactive and visual analysis tools. The development of Moose has been evaluated to 200 person-years.

URL: `https://modularmoose.org`

Contact: Stephane Ducasse

Participants: Anne Etien, Nicolas Anquetil, Stephane Ducasse

Partners: Université de Berne, Sensus, Pleiad, USI, Vrije Universiteit Brussel

7.1.2 Pharo

Keywords: Live programmation objet, Reflective system, Web Application

Functional Description: Pharo is a pure object reflective and dynamic language inspired by Smalltalk. In addition, Pharo comes with a full advanced programming environment developed under the MIT License. It provides a platform for innovative development both in industry and research. By providing a stable and small core system, excellent developer tools, and maintained releases, Pharo's goal is to be a platform to build and deploy mission critical applications, while at the same time continue to evolve.

URL: http://www.pharo.org

Contact: Marcus Denker

Participants: Christophe Demarey, Damien Pollet, Esteban Lorenzano, Marcus Denker, Stephane Ducasse, Guillermo Polito, Pablo Tesone

Partners: BetaNine, Reveal, Inceptive, Netstyle, Feenk, ObjectProfile, GemTalk Systems, Greyc Université de Caen - Basse-Normandie, Université de Berne, Yesplan, RMod, Sensus, Université de Bretagne Occidentale, École des Mines de Douai, ENSTA, Uqbar foundation Argentina, ZWEIDENKER, LifeWare, JPMorgan Chase, KnowRoaming, ENIT, Spesenfuchs, FINWorks, Esug, FAST, Ingenieubüro Schmidt, Projector Software, HRWorks, Inspired.org, Palantir Solutions, High Octane, Soops, Osoco, Ta Mère SCRL, University of Yaounde 1, Software Quality Laboratory, University of Novi Sad, Software Institute Università della Svizzera italiana, Universdad Nacional de Quilmes, UMMISCO IRD, Université technique de Prague

7.1.3 Pillar

Keywords: HTML, LaTeX, HTML5

Functional Description: Pillar is a markup syntax and associated tools to write and generate documentation and books. Pillar is currently used to write several books and other documentation. It is used in the tools developed by Feenk.com.

URL: https://github.com/Pillar-markup/pillar

Contact: Stephane Ducasse

8 New results

8.1 Dynamic Languages

Participants: Steven Costiou, Marcus Denker, Théo Rogliano, Guillermo Polito, Stéphane Ducasse, Vincent Aranega, Pablo Tesone, Oleksandr Zaitsev, Sebastian Jordan Montano.

Unanticipated Object Synchronisation for Dynamically-Typed Languages Developing concurrent programs requires the use of threadsafe abstractions to avoid race conditions. Nevertheless, many libraries are not thread-safe either because it was not a concern for the developer or the developer purposely traded thread-safety for performance. All calls to such a library require to be synchronised in the client program. While different synchronisation mechanisms exist, only lock-based solutions and transaction solutions allow one to add synchronisation without anticipation. But they are both about reasoning in terms of execution. It forces developers to explore the whole execution to identify when synchronisation is required. A developer accidentally missing a synchronisation point circumvents the mechanism. Instead by ensuring exclusive access to an object for a thread prevents other thread to access

it and thus allows developers to miss synchronisation points without circumventing the mechanism. We propose the Atomic Samurai, a synchronisation model that is object-based, unanticipated, and while the object is synchronised the rest of the program normally runs concurrently. It relies on 3 mechanisms: pointer swapping to ensure reference unicity, proxies to control object accesses and late binding to intercept and redirect messages sent. We validate our approach by showing how Atomic Samurai helps both solving examples extracted from the literature and a real-world scenario from the Pharo community on font rendering. We also measure the impact in performance of our solution by comparing it with a semaphore-based solution. In term of performance, our solution is 5 orders of magnitude slower than the semaphore-based solution, but scales better with the number of processes. [30]

How Fast is AI in Pharo? Benchmarking Linear Regression As many other modern programming languages, Pharo spreads its applications into computationally demanding fields such as machine learning, big data, cryptocurrency, etc. This raises a need for fast numerical computation libraries. In this work, we propose to speed up the low-level computations by calling the routines from highly optimized external libraries, e.g., LAPACK or BLAS through the foreign function interface (FFI). As a proof of concept, we build a prototype implementation of linear regression based on the DGELSD routine of LAPACK. Using three benchmark datasets of different sizes, we compare the execution time of our algorithm agains pure Pharo implementation and scikit-learn - a popular Python library for machine learning. We show that LAPACK/Pharo is up to 2103 times faster than pure Pharo. We also show that scikit-learn is 8-5 times faster than our prototype, depending on the size of the data. Finally, we demonstrate that pure Pharo is up to 15 times faster than the equivalent implementation in pure Python. Those findings can lay the foundation for the future work in building fast numerical libraries for Pharo and further using them in higher-level libraries such as pharo-ai. [23]

What do developers consider magic literals? A smalltalk perspective Literals are constant values (numbers, strings, etc.) used in the source code. Magic literals are such values used without an explicit explanation of their meaning. Such undocumented values may hinder sourcecode comprehension, negatively impacting maintenance. Relatively little literature can be found on the subject beyond the usual (and very old) recommendation of avoiding literals and preferring named constants. Yet, magic literals are still routinely found in source code. We studied literal values in source code to understand when they should be considered magic or not (i.e., acceptable). First, we perform a qualitative study of magic literals, to establish why and under which conditions they are considered harmful. We formalize hypotheses about the reasoning behind how literals are considered magic. Second, we perform a quantitative study on seven real systems ranging from small (a few classes) to large (thousands of classes). We report the literals' types (number, string, Boolean,. . .), their grammatical function (e.g., argument in a call, operand in an expression, value assigned,. . .), or the purpose of the code in which they appear (test methods, regular code). Third, we report on another study involving 26 programmers who analyzed about 24,000 literals, to understand which ones they consider magic. Finally, we evaluate the hypotheses defining specific conditions under which literals are acceptable. We show that (1) literals still exist and are relatively frequent (found in close to 50% of the methods considered); (2) they are more frequent in test methods (in 80% of test methods); (3) to a large extent, they were considered acceptable (only 25% considered magic); and (4) the hypotheses concerning acceptable literals are valid to various degrees. Conclusion: We thus pave the way to future research on magic literals, for example, with tools that could help developers deciding if a literal is acceptable. [7]

8.2 Virtual Machines

Participants: Guillermo Polito, Stéphane Ducasse, Pablo Tesone, Théo Rogliano, Pierre Misse-Chanabier, Nahuel Palumbo, Soufyane Labsari.

Differential Testing of Simulation-Based Virtual Machine Generators Automatic Detection of VM Generator Semantic Gaps Between Simulation and Generated VMs Testing and debugging language Virtual Machines (VMs) is a laborious task without the proper tooling. This complexity is aggravated when

the VM targets multiple architectures. To solve this problem, simulation-based VM generator frameworks allow one to write test cases on the simulation, but those test cases do not ensure the correctness of the generated artifact due to the semantic gaps between the simulated VM and generated VMs. We propose Test Transmutation to extend simulation-based VM generator frameworks to support test case generation. It validates the generated VM by also running test cases generated from existing simulation test cases. Results of the generated test cases are compared against the simulation test cases using differential testing. Moreover, test cases are automatically mutated with non-semanticpreserving mutations. Test Transmutation detects bugs that are representative of typical VM modifications. We demonstrate its effectiveness by applying it to a set of real test cases of the Pharo VM. It allowed us to find several issues that were unknown to the VM development team. Our approach shows promising results to test simulationbased VM generator frameworks. [13]

Differential testing of simulation-based VM generators Testing and debugging language Virtual Machines (VMs) is a laborious task without the proper tooling. This complexity is aggravated when the VM targets multiple architectures. Simulation-based VM generator frameworks allow one to write test cases on the simulation, however they do not ensure the correctness of the generated artifact due to the semantic gap between the environments. In this article we propose Test Transmutation. It extends simulation-based VM generator frameworks to also generate simulation test cases and execute them on the generated VMs. It extends such frameworks to translate test cases and applies differential testing and non-semantic-preserving mutations. Test Transmutation detects bugs that are representative of typical VM modifications. Moreover, we apply it to a set of real test cases of the Pharo VM and find several issues. Our approach shows promising results to test simulation-based VM generator frameworks. [14]

Ease Virtual Machine Level Tooling with Language Level Ordinary Object Pointers Virtual Machines (VMs) are critical language execution engines. When tooling the VM level, developers face an important abstraction gap. For instance, a VM supporting an Object-oriented Programming language often manipulates its memory using addresses whereas these addresses are hidden in the language this VM supports. This discourages tooling at the VM level. We propose to use language level object ordinary pointer (LLOOP) to reduce the abstraction gap. LLOOP combine VM level and language level knowledge at the VM level to ease VM tooling. We present our implementation on the Pharo language, which is supported by the Pharo VM. Moreover, we created two tools solving two real-world major bugs in the Pharo environment. These tools required VM level support. First, we investigate how to fix a meta error that was breaking a Pharo environment, preventing it to open. We repair the broken environment by tracking and fixing the language level method responsible for the error at the VM level. Second, we investigate a corrupted Pharo image. A few objects in the Pharo memory space were corrupted i.e., the VM was not able to read and manipulate them. We are able to identify and remove the corrupted objects, fixing the Pharo environment. [15]

Interpreter Register Autolocalisation: Improving the performance of efficient interpreters Interpreter performance has been a hot topic for a long time, where several solutions have been proposed with different ranges of complexity and portability. On the one hand, some work proposes to optimize language-specific features in interpreters such as type dispatches using static type predictions, quickening or type specializations. On the other hand, many solutions focus on improving general interpreter behavior by minimizing branch miss-predictions of interpreter dispatches and stack caching. Some solutions aim for minimizing branch miss-predictions by modifying the intermediate code (e.g., bytecode) design with super-instructions and register-based instructions. Stack caching proposes to optimize the access of operands by caching the top of the stack. interpreter registers are also related to stack caching: interpreter variables that are critical to the efficient execution of the interpreter loop. Examples of such variables are the instruction pointer (IP), the stack pointer (SP), and the frame pointer (FP). Interpreter registers put pressure on the overall design and implementation of the interpreter: Req1: Value access outside the interpreter loop. VM routines outside of the interpreter loop may require access to interpreter registers. For example, this is the case of garbage collectors that need to traverse the stack to find root objects, routines that unwind or reify the stack, or give access to stack values to native methods. Req2: Efficiency. Interpreter registers are used on each instruction to manipulate the instruction stream and

the stack. Under-efficient implementations have negative impacts on performance. [16]

Porting a JIT Compiler to RISC-V: Challenges and Opportunities The RISC-V Instruction Set Architecture (ISA) is an open-source, modular and extensible ISA. The ability to add new instructions into a dedicated core opens up perspectives to accelerate VM components or provide dedicated hardware IPs to applications running on top. However, the RISC-V ISA design is clashing on several aspects with other ISAs and therefore software historically built around them. Among them, the lack of condition codes and instruction expansion through simple instruction combination. We present the challenges of porting Cogit, the Pharo's JIT compiler tightly linked to the x86 ISA, on RISC-V. We present concrete examples of them and the rationale behind their inclusion in the RISC-V ISA. We show how those mismatches are solved through design choices of the compilation process or through tools helping development: a VM simulation framework to keep the development in a high-level environment for the most part, an ISA-agnostic test harness covering main VM functionalities and a machine code debugger to explore and execute generated machine code. We also present a way to prototype custom instructions and execute them in the Pharo environment. [12]

Interpreter-guided Differential JIT Compiler Unit Testing Modern language implementations using Virtual Machines feature diverse execution engines such as byte-code interpreters and machine-code dynamic translators, a.k.a. JIT compilers. Validating such engines requires not only validating each in isolation, but also that they are functionally equivalent. Tests should be duplicated for each execution engine exercising the same execution paths on each of them. We present a novel automated testing approach for virtual machines featuring byte-code interpreters. Our solution uses concolic meta-interpretation: it applies concolic testing to a byte-code interpreter to explore all possible execution interpreter paths and obtain a list of concrete values that explore such paths. We then use such values to apply differential testing on the VM interpreter and JIT compiler. This solution is based on two insights: (1) both the interpreter and compiler implement the same language semantics and (2) interpreters are simple executable specifications of those semantics and thus promising targets to (meta-) interpretation using concolic testing. We validated it on 4 different compilers of the open-source Pharo Virtual Machine and found 468 differences between them, produced by 91 different causes, organized in 6 different categories. [17]

8.3 Debugging

Participants: Steven Costiou, Vincent Aranega, Marcus Denker, Maximilian Willembrinck, Anne Etien.

Reflection as a Tool to Debug Objects We share our experience with using reflection as a systematic tool to build advanced debuggers. We illustrate the usage and combination of reflection techniques for the implementation of object-centric debugging. Object-centric debugging is a technique for object-oriented systems that scopes debugging operations to specific objects. The implementation of this technique is not straightforward, as there are, to the best of our knowledge, no description in the literature about how to build such debugger. We describe an implementation of object-centric breakpoints. We built these breakpoints with Pharo, a highly reflective system, based on the combination of different classical reflection techniques: proxy, anonymous subclasses, and sub-method partial behavioral reflection. Because this implementation is based on common reflective techniques, it is applicable to other reflective languages and systems for which a set of identified primitives are available. [11]

Towards Object-Centric Time-Traveling Debuggers Object-centric debugging aims at facilitating the debugging of object-oriented programs by focusing debugging operations on specific objects. This technique is tedious to use because developers have to manually find objects to debug, which is not straightforward. Time-traveling debuggers allows developers to explore executions back and forth in time. It has been shown that time-traveling features effectively facilitate debugging and program understanding. We propose to combine these techniques to benefit from both of them to debug objectoriented programs.

Time-travel navigation could help finding and remembering objects by providing means to explore executions back and forth. Object-centric debugging could extend time-traveling debugging with object-centric exploration features. These techniques have never been combined, and the challenges and benefits of such combination have never been explored. We present SeekerOC, a time-traveling debugger prototype which provides object-centric debugging support. To combine both techniques, we use Time-Traveling Queries, a query system to automatically explore executions. We discuss the expected benefits of this combination, and we argue that exploring object-centric time-traveling debugging will open new research perspectives towards more effective debugging techniques and tools for object-oriented systems. [20]

8.4 Software Reengineering

Participants: Stéphane Ducasse, Anne Etien, Steven Costiou, Nicolas Anquetil, Santiago Bragagnolo, Oleksandr Zaitsev, Benoit Verhaeghe, Abderrahmane Seriai, Nour Jihene Agouf.

Understanding Class Name Regularity: A Simple Heuristic and Supportive Visualization. Studies have shown that more than 50% of software maintenance time is spent reading code to understand it. This puts a strong emphasis on the understandability of source code. Class names constitute one of the first pieces of information developers have access to. To assist developers in understanding the logic and regularity of class names, we present a new and simple visualization, called ClassName Distribution. It brings together package and inheritance as structural perspectives on class names. ClassName Distribution allows one to spot naming irregularities in large hierarchies scattered over multiple packages. We show (1) how this visualization helps capture recurrent patterns relative to concept reference in class names and (2) that this visualization supports the evolution of software systems by monitoring and guiding class renamings over multiple versions. To evaluate our approach we did a consequent assessment with real practitioners and open-source software structured in two different setups: in the first one, we asked domain experts to use the visualization: three groups of engineers applied our tool to the the software they develop or maintain. They proposed and performed respectively 91, 68, and 24 class renamings. In the second setup, as authors of the visualization and the tool (visualization experts), we applied our tool to a new UI framework for Pharo. We sent 34 pull requests for renaming classes and 32 were accepted. Finally, we applied our visualizations to 50 Java projects and identified visual patterns in most of them. Consequently, it shows that the proposed visualization is effective for spotting class name inconsistencies, and this by both developers of the system and external persons. [6]

Deprewriter: On the fly rewriting method deprecations Deprecations are a common way to indicate that a given feature or API will not be available in subsequent versions of a library or framework. While raising deprecation warnings lets developers of libraries evolve their APIs, the developers of client applications often have to manually rewrite their applications to adapt to the deprecation (removal, new APIs...). Some may use static analysis tools to support the rewriting. However, dynamically-typed languages or the use of reflective features often produce incorrect rewrite candidates. This is a costly activity that can lead to bug introductions. In this article, we present a method deprecation approach and a tool called DEPREWRITER that can automatically rewrite the callers of deprecated methods during program execution. Clients of a deprecated API execute their program and associated tests, and DEPREWRITER dynamically rewrites the source code of methods that called a deprecated API to use the new API. The implementation of DEPREWRITER is based on dynamic program transformation: when a deprecated method is executed, a program transformation engine rewrites and recompiles the caller's code before continuing the execution. The approach presented in this article has been developed by the Pharo consortium. Since 2016, DEPREWRITER is used in production in multiple distributions of the Pharo programming language: Pharo 6, 7, 8, and 9 alpha. This article presents and validates this approach. The validation is done in two steps: first with an analysis of deprecations available in Pharo 8 and second with an open survey of software developers about DEPREWRITER. We studied 367 Pharo 8 deprecations, among which we analyzed the 218 rewriting deprecations that use transformation

rules. We identified the validity conditions and reported defects to the community. We also proposed 33 transformation rules to be added to the non-rewriting deprecations. Both contributions were accepted into Pharo 9 alpha. We classified the rules and identified possible points of improvement. In addition, we performed a user survey and collected information from 46 software developers: some of them used existing DEPREWRITER' rules and executed them on their code, others used DEPREWRITER to create rewriting deprecations, and finally, some were not aware of DEPREWRITER. 28 of 46 developers (60%) reported that the rewriting deprecations helped them, while 10 stated the inverse and 8 were uncertain. After discussing the current implementation, we sketch possible implementations for other languages than Pharo, showing that the approach is general enough to be applied to other languages. [8]

A New Generation of Class Blueprint In object-oriented programming, classes are the primary abstraction mechanism used by and exposed to developers. Understanding classes is key for the development and evolution of object-oriented applications. The fundamental problem faced by developers is that while classes are intrinsically structured entities, in IDEs they are represented as a blob of text. The idea behind the original CLASS BLUEPRINT visualization was to represent the internal structure of classes in terms of fields, their accesses, and the method call flow. Additional information was depicted using colors. The thus created visualization proved to be an effective means to support program comprehension. However, a number of omissions rendered it only partially useful. We propose CLASS BLUEPRINT V2 (in short BLUEPRINTV2), which in addition to the information depicted by CLASS BLUEPRINT also supports dead code identification, methods under tests, and calling relationships between class and instance level methods. In addition, BLUEPRINTV2 enhances the understanding of fields by showing how fields of super/subclasses are accessed. We present the enhanced visualization and report on a first validation with 26 developers and 18 projects. [9]

Transformation-based Refactorings: a First Analysis Refactorings are behavior preserving transformations. Little work exists on the analysis of their implementation and in particular how refactorings could be composed from smaller, reusable, parts (being simple transformations or other refactorings) and how (non behavior preserving) transformations could be used in isolation or to compose new refactoring operators. In this article we study the seminal implementation and evolution of Refactorings as proposed in the PhD of D. Roberts. Such an implementation is available as the Refactoring Browser package in Pharo. In particular we focus on the possibilities to reuse transformations independently from the behavior preserving aspect of a refactoring. The long term question we want to answer is: Is it possible to have more atomic transformations and refactorings composed out of such transformations? We study preconditions of existing refactorings and identify several families. We identify missed opportunities of reuse in the case of implicit composite refactorings. We analyze the refactorings that are explicitly composed out of other refactorings to understand whether the composition could be expressed at another level of abstraction. This analysis should be the basis for a more systematic expression of composable refactorings as well as the reuse of logic between transformations and refactorings. [10]

DepMiner: Automatic Recommendation of Transformation Rules for Method Deprecation Software applications often depend on external libraries and must be updated when one of those libraries releases a new version. To make this process easier, library developers try to reduce the negative effect of breaking changes by deprecating the API elements before removing them and suggesting replacements to the clients. Modern programming languages and IDEs provide powerful tools for deprecations that can reference the replacement or incorporate the rules written by library developers and use them to automatically update the client code. However, in practice library developers often miss the deprecation opportunities and fail to document the deprecations. In this work, we propose to help library developers support their clients with better deprecations. We rely on the transforming deprecations offered by Pharo and use data mining to detect the missing deprecation opportunities and generate the transformation rules. We implemented our approach for Pharo in a prototype tool called DepMiner. We have applied our tool to five open-source projects and proposed the generated deprecations to core developers of those projects. 63 recommended deprecations were accepted as pull requests. [21]

A Hybrid Architecture for the Incremental Migration of a Web Front-end Nowadays, software migration is an effective solution to adopt new technologies while reusing the business value of existing applications. Among other challenges, the size and complexity of large applications are obstacles that increase the risks of migration projects. Moreover, the migration can imply a switch of programming languages. This is the case when migrating from Java to TypeScript. Thus, it is hard to migrate large and complex applications in one straightforward step. Incremental approaches have been designed to counter this problem. These approaches are based on hybrid architecture usages. However, none of the approaches use a hybrid architecture for GUI defined with different programming languages. We propose a new hybrid architecture that enables the incremental migration of web applications. Our architecture is based on Web Components that allow legacy technology artifacts to work with modern ones. We implement the architecture and use it in the case of migrating GWT applications to Angular. Then, we validate its usability in a real context by migrating an industrial web application. [19]

How Libraries Evolve: A Survey of Two Industrial Companies and an Open-Source Community The evolution of software libraries is a process that requires a joint effort of two groups of developers: the library developers who prepare the release and client developers who need to update their applications to the new versions. To build better tools that support both library and client developers throughout the evolution, we need to understand what problems they face and how they react to those problems. We present the result of two surveys: one for library developers and one for client developers. Our surveys involved developers from two industrial companies and an open-source community. We assess (1) how they perceive the impact of library evolution and (2) what is the support that library developers can provide to their clients. By approaching those questions from the perspectives of library and client developers, we try to assess how challenging library update is for each of those groups and how motivated they are to overcome those challenges. [22]

9 Bilateral contracts and grants with industry

Thales DMS, Brest, France
Participants: Pharo Consortium, from 2023.
Industrial R&D collaboration with Dr. Eric Le Pors, lead prototyping architect at Thales DMS (Brest). We work on the Pharo core graphics library.

Participants: Steven Costiou, from 2020, ongoing.
Industrial R&D collaboration with Dr. Eric Le Pors, lead prototyping architect at Thales DMS (Brest). We work on 1) unanticipated object-centric debugging of HMI prototypes 2) we study the practices of Thales with software component reuse and its impact on their development process.

Berger Levrault, France
Participants: Nicolas Anquetil, Santiago Bragagnolo, Stéphane Ducasse, Anne Etien, Benoît Verhaeghe
From 2017, ongoing.
Collaboration with the software editor Berger-Levrault about software architecture remodularization. The collaboration started with an end study project exploring the architecture used in the company in order to later migrate from GWT to Angular since GWT will not be backward supported anymore in the next versions. A PhD CIFRE thesis finished in 2021. S. Bragagnolo started a CIFRE in 2020

Siemens AG, Germany
Participants: Stéphane Ducasse, Anne Etien, Nicolas Anquetil
The Siemens Digital Industry Division approached our team to help them restructure a large legacy systems.

CIFRE Arolla, France
Participants: Nicolas Anquetil, Stéphane Ducasse, Anne Etien, Oleksandr Zaitsev, Nour Jihene Agouf.
We are collaborating with the council company, Arolla, about software evolution. Arolla has daily problems with identifying architecture, design, and deviations from those artefacts. The goal of Oleksandr's

CIFRE (finished in 2022) experiments with different machine learning techniques that can help us automate the process of library migration. A new CIFRE PhD (from 2021) is based around the study of visualisation techniques that can help us understand legacy systems.

CIM, France
Participants: Honore Mahugnon Houekpetodji, Stéphane Ducasse, Nicolas Anquetil, from 2019.
A PhD started in 2019. We work on the analysis of PowerBuilder applications. PhD finished in 2022: Honore Mahugnon Houekpetodji *Analyse multi-facettes et operationnelle pour la transformation des systèmes d'information.*

Pharo Consortium
From 2012, ongoing.
The Pharo Consortium was founded in 2012 and is growing constantly. consortium.pharo.org.

Lifeware AG, Switzerland
Participants: Pablo Tessone, Esteban Lorenzano, Marcus Denker, Stéphane Ducasse, ongoing.
In collaboration with the Pharo Consortium, we improve Pharo. The goal is to be able to work with very large systems (>100K classes).

Ingenieurbüro für Bauwesen Schmidt GmbH
Participants: Esteban Lorenzano, Marcus Denker, Stéphane Ducasse, Pablo Tessone, ongoing.
In collaboration with the Pharo Consortium, we improve Pharo. One focus is the use of Pharo to build user interfaces on the windows platform.

Dedalus
Participant: Nicolas Anquetil, Stéphane Ducasse, Soufyane Labsari, Anne Etien, from 2021
A collaboration started in 2021. It includes a 6 month engineer position. The goals are (1) The development of a software prototype for the identification of unused functionalities within an application developed by Inovelan. (2) Analysis of the source code of the software using the open-source software platform Moose. (3) Identification of a CIFRE thesis subject on software maintenance and development.

10 Partnerships and cooperations

10.1 International initiatives

10.1.1 Inria associate team not involved in an IIL or an international program

SADPC

Title: Systems Analyses and Debugging for Program Comprehension

Duration: 2020-2023 (no visits possible due to COVID in 2020 and 2021).

Coordinator: Yann-Gaël Guéhéneuc (Concordia University)

Partners:
- IDepartment of Electrical Engineering, Concordia University (Canada)
- Christopher Fuhrman: Ecole de Technologie Supérieure (Montreal)
- Fabio Petrillo, UQAC, Université du Québec à Chicoutimi
- Foutse Khomh,Polytechnique Montréal.

Inria contact: Stéphane Ducasse

Summary: Systemic changes in the past decades have pushed software systems into all aspects of our lives, from our homes to our cars to our factories. These systems, both legacy (e.g., handling contracts for the Dept of Defense of the USA since 1958) and very recent (e.g., running the latest smart factory in France in 2019), are difficult to understand by software engineers because of their

intrinsic complexity. These engineers need help understanding the systems they must adapt to the new requirements of our time. The proposed associate team considers three research directions to support the software engineers maintaining and evolving large software systems: (a) system analyses and (b) debugging for (c) program comprehension. (a) Complex algorithms often act or are perceived by software engineers as black boxes because of their intrinsic and accidental complexity, both in architecture, design, and implementation. We will develop new software analyses to support algorithm understanding. (b) Previous debugging techniques assume a unique software engineer performing a solitary debugging session. We will work on a language allowing software engineers to build their own debuggers to fit their collaborative debugging strategies. (c) Previous work on program comprehension proposed views to address one single problem at a given moment of the comprehension process. They only provide a subset of the information required by software engineers. We want to propose an approach to adapt and combine views using meta-data.

10.1.2 Participation in other International Programs

University of Novi Sad, Serbia
Participants: Stéphane Ducasse, Anne Etien, Nicolas Anquetil, Vincent Aranega.

We started to collaborate with two groups of the University of Novi Sad (G. Rakic and G. Milosavljevic). We hope post COVID will let us restart our efforts.

Vrije Universiteit Brussel (VUB), Belgium – SOFT
Participants: Guillermo Polito, Matteo Marra.

We collaborate since several years with the Soft team (previously PROG) of the Vrije Universiteit Brussels (Prof E. Gonzalez Boix). We got a large number of exchanges between our two teams - this was slowed down because of COVID. G. Polito co-supervised the PhD of M. Marra with E. Gonzales Boix on debugging map reduce applications.

Université de Chicoutimi au Quebec, Canada
Participants: Steven Costiou, Stéphane Ducasse, from 2020.
We collaborate with F. Petrillo, who builds cloud infrastructures for large-scale evaluation of debuggers. We use these infrastructures for the empirical evaluation of our debugging tools.

University of Zurich, Switzerland
Participants: Steven Costiou, Stéphane Ducasse, Marcus Denker from 2020.

We collaborate with A. Bachelli on large-scale evaluations of debugging tools. This collaboration involves 3 researchers and 2 PhD students.

Instituto Federal de Educação Ciência e Tecnologia do Ceará, Brazil
Parcipants: Vincent Aranega, from 2020.

Collaboration with Pr. Antonio Wendell de Oliveira Rodrigues on smart language and DSL for live coding semi-autonomous physical systems (drones).

Open University, UK
Participants: Marcus Denker and Pablo Tessone, from 2020.

With Prof. Simon Holland and Günter Khyo (Vienna/Austria) we are working on Direct Combination using the new traits model of Pharo.

University of Prague, Czech Republic
Participants: Stéphane Ducasse. From 2015, ongoing.

We are working with Dr. Robert Pergl from the University of Prague. Stéphane Ducasse gave a lecture at the University of Prague in 2018, the next lecture is planned for 2021.

10.2 International research visitors

10.2.1 Visits of international scientists

Jean Privat

Status Professor

Institution of origin: Université du Québec à Montréal

Country: Canada

Dates: Sep 2022 to Nov 2022

Context of the visit: Pharo/VM/Research

Mobility program/type of mobility: sabbatical

Kasper Osterbye

Status Professor

Country: Denmark

Dates: Februar 2022 (2 weeks)

Context of the visit: Microdown design

Mobility program/type of mobility: Research stay

Michele Lanza

Status Professor

Country: Switzerland

Dates: March 2022

Context of the visit: Class Blueprint

Mobility program/type of mobility: Research stay

Short Visits:

- Balsa Sarenac, University of Novi Sad. Serbia, from Sep 2022 until Oct 2022 (1.5 months)
- Quentin Ducasse, École Nationale Supérieure de Techniques Avancées, Sep 2022 (3 weeks)
- Max Zurbriggen, University of Zürich, from Oct 2022 until Nov 2022 (3 weeks)
- Gabriel Cavalheiro Ullmann, Concordia University, from Oct 2022 until Nov 2022
- Gordana Rakic, Researcher, University of Novi Sad, Dec 2022 (10 days)
- Quentin Stievenart, Université Libre de Bruxelles, Sep 2022
- Richard Uttner, Projector Software GmbH and Ingenieurbüro für Bauwesen Schmidt GmbH, (Germany), Sept 2022

- Domenico Cipriani. Independent Artist, March 2022
- Coen De Roover, Université Libre de Bruxelles, Oct 2022
- Chouki Tibermacine, Université de Montpellier, Jun 2022
- Romain Robbes, CNRS, Oct 2022
- Mireille Blay Fornarino, Université Côte d'Azur, Jun 2022
- Mathieu Bacou, TelecomParis Sud, Jun 2022

10.2.2 Visits to international teams

Research stays abroad Stéphane Ducasse and Nicolas Anquetil visited several groups/universities in Montreal, Canada in June 2022 (in no particular order):

- Ecole Polytechnique of Montreal
- UQAM – University of Quebec in Montreal
- UMontreal, University of Montreal
- ETS, École de technologie supérieure
- Concordia University
- UOttawa, University of Ottawa

10.3 European initiatives

10.3.1 H2020 projects

RMOD is part of the COST project CERCIRAS: Connecting Education and Research Communities for an Innovative Resource Aware Society. www.cost.eu/actions/CA19135

10.3.2 Digital Europe

Steven Costiou: **Work Package Leader : "Skills and Formation"** for Inria Lille Nord Europe, European Digital Innovation Hub - GreenPowerIT (Hauts de France). 2022. EDIH

10.4 National initiatives

ARCAD, Lab-STICC, Brest, France

Participants: Guillermo Polito, Pablo Tesone, Stéphane Ducasse.

We collaborate since the beginning of 2021 with the ARCAD team of the Lab-STICC in Bretagne (Prof L. Lagadec.) We started at the beginning of the year with a common workshop between the two teams looking for collaboration points. G. Polito and P. Tesone are now collaborating with the PhD of Q. Ducasse on Just-In-Time compiler technology for extensible ISA processors such as RISC-V.

École nationale d'Ingénieurs de Tarbes

Participants: Marcus Denker.

With Cédrik Béler (LGP/ICE) we are exploring the life-cycle (contextual time relation) of data, information, and knowledge in the context of Object-Oriented data models.

ANR JCJC OCRE
Partners: RMoD, SmArtSE (UCAQ, Quebec), UX Prototyping (Thales DMS, Brest) (From 2022 to 2024).

Participants: Steven Costiou, Maximilian Ignacio Willembrinck Santander, Marcus Denker.

The objectives of the OCRE project are to study the fundamental and practical limits that hinder the implementation, the evaluation, and the adoption of object-centric debugging. We propose to build the first generation of object-centric debuggers, in order to identify and evaluate its real benefits to OOP debugging. We argue that these debuggers have the potential to drastically lower the cost (time and effort) of tracking and understanding hard bugs in OOP.

Action Exploratoire Inria: AlaMVic

Participants: Guillermo Polito, Pablo Tesone, Nahuel Palumbo.

Summary: Language Virtual Machines (VMs) are pervasive in every laptop, server, and smartphone. Industry-level VMs use highly-engineered optimization techniques, often handcrafted by experts, difficult to reproduce, replicate and change. Such optimization techniques target mostly speed improvements and are incompatible with constraints such as space and energy efficiency important in the fields of IoT or robotics. In AlaMVic1 we propose to approach VM construction using a holistic generative approach, in contrast with existing approaches that focus on speed and single VM components such as the JIT compiler. We explore how to transform handcrafted optimizations into generation heuristics, how they are applied and combined in fields such as IoT and robotics, and new methods and metrics to evaluate VMs in such fields.

10.5 Regional initiatives

IMT Douai
Collaboration with Prof L. Fabresse and Prof. N. Bouraqadi. The PhDs of P. Tesone, P. Misse, and C. Hernandez are joint PhD with the team of IMT Douai.

11 Dissemination

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

- ESUG 2022 Novi Sad Serbia.
- Pharo Days March 3-4 2022 Lille, France.

General chair, scientific chair

- Vincent Aranega was chair of IWST'22

11.1.2 Scientific events: selection

Member of the conference program committees

- Anne Etien: ICSME 2022, SANER (ERA track) 2022, SANER (ERA Track 2023), ICSR 22 (Doctoral Consortium), ICSE 22 (SRC track).
- Steven Costiou: SANER ERA track 2023
- Stéphane Ducasse: ECOOP22, APSEC 22
- Guillermo Polito: MPLR'22
- Marcus Denker: 22RAW, 22Smaltalks

11.1.3 Journal

Reviewer - reviewing activities

- Anne Etien: Empirical Software Engineering Journal

11.1.4 Invited talks

- Milton Mamani-Torres: Open Source Experience. Talk about About Visualization with Roassal (09/11/2022)
- Anne Etien: Software Maintenance: Around Test and Program Comprehension at LS2N (12/12/22)
- Guillermo Polito gave an invited talk in Argentina (LAFHIS group, Universidad de Buenos Aires): Máquinas Virtuales Performantes y Robustas: Desafíos Actuales y Pharo como Motor de Research (8/11/22) (Performant and Robust Virtual Machines: Challenges and Pharo as Research Engine)
- Guillermo Polito gave an invited talk in Argentina (Uqbar Foundation, Universidad Tecnológica Nacional Buenos Aires): Implementación de lenguajes y Open Source: mi experiencia en Pharo (Language implementation and Open Source: my Pharo Experience) (12/11/22)

11.1.5 Leadership within the scientific community

GT GLIA working group of the CNRS GDR GPL
Anne Etien is co-leader of the GT GLIA working group of the CNRS GDR-GPL (Software Engineering and artificial intelligence) from 2020

GT Debugging working group of the CNRS GDR GPL
Steven Costiou is leader of the GT Debugging working group of the CNRS GDR GPL. This working group aims to gather any researcher, engineer or GDR team interested in software debugging problems. (from 2020)

11.1.6 Scientific expertise

Stéphane Ducasse: Evaluation CIR et JEI (Jeune entreprise innovante)

11.1.7 Research administration

- Anne Etien animates the thematic group of Software Engineering and is member of scientific council of CRIStAL lab.
- Anne Etien: Directrice des Études de la L3 MIAGE, Université de Lille

11.2 Teaching - Supervision - Juries

11.2.1 Teaching

- Licence: Nicolas Anquetil, Conception OO Avancée, 48h, L2, IUT- A, Université de Lille, France
- Licence: Nicolas Anquetil, Programation Mobile, 30h, L2, IUT-A, Université de Lille, France
- Licence: Nicolas Anquetil, Programation Web, 16h, L2, IUT-A, Université de Lille, France
- Licence: Nicolas Anquetil, Projets Agiles, 12h, L2, IUT-A, Université de Lille, France
- Master: Santiago Bragagnolo, Navigation autonome , 28h, CM, M1, ISEN
- Master: Santiago Bragagnolo, Initiation aux technologies Blockchain, 6h, CM, M2, Universite de lille (MFCA)
- Master: Steven Costiou, Conception et modélisation objet, Polytech Lille, 12hCM 12hTD
- Master: Steven Costiou, Fondamentaux du debugging, Université de Lille, 4hCM 8hTD
- Master: Steven Costiou, Debugging: bases and implementation, Université de Brest, 4hCM 8hTD
- Master: Stéphane Ducasse, Programmation orientée objet, Centrale Lille, 5h CM 5hTD
- Master: Stéphane Ducasse, Meta, Université de Lille, 12hTD
- Master: Stéphane Ducasse, Conception avancée, Université de Lille, 60hTD
- Licence: Anne Etien, Introduction à la programmation, 40h, L1, Université de Lille
- Licence: Anne Etien, projet, 24h, L2, Université de Lille
- Licence: Anne Etien, Conception orientée objet, 18, L3, Université de Lille
- Licence: Anne Etien, Bases de données relationnelles, 16h, L3, Université de Lille
- Licence: Anne Etien, Génie Logiciel, 27h, L3, Université de Lille
- Master: Anne Etien, Metamodelisation, 30h, M2, Université de Lille
- Licence: Carolina Hernandez, Algorithmique et Programmation, 12h, L3, IMT Nord Europe, France
- Licence: Carolina Hernandez, Introduction à la Programmation par Objets avec Java, 20h, L3, IMT Nord Europe, France
- Master: Guillermo Polito, Analyse et Verification de Logiciel, Université de Lille, 16h CM
- Master: Guillermo Polito, Conception et Paradigmes de Programmations par la Pratique, Université de Lille, 48hTD
- Licence: Guillermo Polito, Meta, Université de Lille, 13hTD
- Licence: Pablo Tesone, Project Programation orientée objet, 20, L3, Polytech Lille

11.2.2 Supervision

- PhD: Pierre Misse-Chanabier, Modular, green, versatile Virtual Machines, started Oct 2019, Stéphane Ducasse, Noury Bouraqadi, finished 16/12/2022
- PhD: Oleksandr Zaitsev, Machine Learning-Based Tools to Support Software Evolution, started Jul 2019, Stéphane Ducasse, Nicolas Anquetil, finished 28/10/2022
- PhD: Mahugnon Honoré Houekpetodji, Multi-Facet Actionable for Information System Rejuvenation, SPI Lille, France, Stéphane Ducasse, Nicolas Anquetil, Nicolas Dias, Jérome Sudich, finished 24/06/2022
- PhD in progress: Santiago Bragagnolo Migration de programmes légataires vers des architectures Web: le cas de de la migration de programmes Microsoft Access vers Angular / Microservices, CIFRE Berger-Levrault, Stéphane Ducasse, Nicolas Anquetil.
- PhD in progress: Maximilian Ignacio Willembrinck Santander, Scriptable Time-Traveling Debuggers, since october 2020, Inria, Anne Etien, Steven Costiou
- PhD in progress: Iona Thomas, Elements of Language Strenghtening, since 2021 Stéphane Ducasse, Pablo Tesone, Guillermo Polito
- PhD in progress: Nour-Dijene Agouf, Visualisations for Real software systeme, since 2021, CIFRE Arolla, Stéphane Ducasse, Anne Etien
- PhD in progress: Aless Hosry, Model transformations for automatic source code modification, Nicolas Anquetil.
- PhD in progress: Gabriel Darbord, Automatic Test Generation, since october 2022, Inria through the EPC with BL, Anne Etien, Nicolas Anquetil.
- PhD in progress: Valentin Bourcier, Reducing the cost of debugging with the first generation of object-centric debuggers, since october 2022, Inria, Steven Costiou
- PhD in progress: Nahuel Palumbo, Virtual Machine Generation Techniques, since november 2022, Inria, Stéphane Ducasse, Guillermo Polito

11.2.3 Juries

- Anne Etien: Faezeh Khorram, *A Testing Framework for Executable Domain Specific Languages*, IMT Atlantic, 12 December 2023, Anne Etien, reviewer.
- Nicolas Anquetil: "confirmation jury" (end of phd 2nd year) of Céline Deknop, supervised by Kim Mens at UC Louvain

11.3 Popularization

11.3.1 Internal or external Inria responsibilities

- Anne Etien is elected member of the center committee of Inria Lille Nord Europe center.
- Marcus Denker is a member of the AGOS board (Section culture) of Inria Lille
- Guillermo Polito is a member of the Argentinian Uqbar Foundation
- Guillermo Polito and Marcus Denker are members of the Pharo Board

11.3.2 Articles and contents

- Pharo with Style got published by BOD [24]
- Pharo by Example 9 got published by BOD [25]
- Article on Inria website: *The importance of debugging*

11.3.3 Interventions

- We organized Pharo Days March 3-4.
- Pharo Workshop at Inria March 2.
- Multiple public Pharo Sprints in Lille/Online.

12 Scientific production

12.1 Major publications

[1] S. Costiou, V. Aranega and M. Denker. 'Sub-method, partial behavioral reflection with Reflectivity: Looking back on 10 years of use'. In: *The Art, Science, and Engineering of Programming* 4.3 (Feb. 2020). DOI: 10.22152/programming-journal.org/2020/4/5. URL: https://hal.inria.fr/hal-02480136.

[2] J. Delplanque, A. Etien, N. Anquetil and S. Ducasse. 'Recommendations for Evolving Relational Databases'. In: *CAiSE 2020 - 32nd International Conference on Advanced Information Systems Engineering*. Grenoble, France, June 2020. URL: https://hal.inria.fr/hal-02511466.

[3] G. Polito, P. Tesone and S. Ducasse. 'Interpreter-guided Differential JIT Compiler Unit Testing'. In: Programming Language Design and Implementation - PLDI 2022. San Diego, United States, 13th June 2022. URL: https://hal.archives-ouvertes.fr/hal-03607939.

[4] P. Tesone, S. Ducasse, G. Polito, L. Fabresse and N. Bouraqadi. 'A new modular implementation for Stateful Traits'. In: *Science of Computer Programming* 195 (Apr. 2020). DOI: 10.1016/j.scico.2020.102470. URL: https://hal.inria.fr/hal-02541842.

[5] P. Tesone, G. Polito, L. Fabresse, N. Bouraqadi and S. Ducasse. 'Preserving Instance State during Refactorings in Live Environments'. In: *Future Generation Computer Systems* (2020). DOI: 10.1016/j.future.2020.04.010. URL: https://hal.archives-ouvertes.fr/hal-02541754.

12.2 Publications of the year

International journals

[6] N. J. Agouf, S. Ducasse, A. Etien, A. Alidra and A. Thiefaine. 'Understanding Class Name Regularity: A Simple Heuristic and Supportive Visualization.' In: *The Journal of Object Technology* 21 (2022). DOI: 10.5381/jot.2022.21.1.a2. URL: https://hal.inria.fr/hal-03706041.

[7] N. Anquetil, J. Delplanque, S. Ducasse, O. Zaitsev, C. Fuhrman and Y.-G. Guéhéneuc. 'What do developers consider magic literals? A smalltalk perspective'. In: *Information and Software Technology* 149 (Sept. 2022). DOI: 10.1016/j.infsof.2022.106942. URL: https://hal.inria.fr/hal-03679130.

[8] S. Ducasse, G. Polito, O. Zaitsev, M. Denker and P. Tesone. 'Deprewriter: On the fly rewriting method deprecations.' In: *The Journal of Object Technology* 21.1 (2022), pp. 1–23. DOI: 10.5381/jot.2022.21.1.a1. URL: https://hal.inria.fr/hal-03563605.

International peer-reviewed conferences

[9] N. J. Agouf, S. Ducasse, A. Etien and M. Lanza. 'A New Generation of Class Blueprint'. In: VISSOFT 2022 - IEEE Working Conference on Software Visualization. Limassol, Cyprus, 2nd Oct. 2022. DOI: 10.1109/VISSOFT55257.2022.00012. URL: https://hal.inria.fr/hal-03752237.

[10] N. Anquetil, M. Campero, S. Ducasse, J.-P. Sandoval and P. Tesone. 'Transformation-based Refactorings: a First Analysis'. In: IWST 22 - International Workshop of Smalltalk Technologies. Novisad, Serbia, 24th Aug. 2022. URL: https://hal.inria.fr/hal-03752247.

[11] S. Costiou, V. Aranega and M. Denker. 'Reflection as a Tool to Debug Objects'. In: SLE 2022 - 15th ACM SIGPLAN International Conference on Software Language Engineering. Auckland, New Zealand, 5th Dec. 2022. DOI: 10.1145/3567512.3567517. URL: https://hal.inria.fr/hal-03846015.

[12] Q. Ducasse, G. Polito, P. Tesone, P. Cotret and L. Lagadec. 'Porting a JIT Compiler to RISC-V: Challenges and Opportunities'. In: Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes (MPLR '22). Brussels, Belgium, 14th Sept. 2022. URL: https://hal.science/hal-03725841.

[13] P. Misse-Chanabier, G. Polito, N. Bouraqadi, S. Ducasse, L. Fabresse and P. Tesone. 'Differential Testing of Simulation-Based Virtual Machine Generators Automatic Detection of VM Generator Semantic Gaps Between Simulation and Generated VMs'. In: International Conference on Software and Software Reuse. Montpellier, France, June 2022. URL: https://hal.inria.fr/hal-03783354.

[14] P. Misse-Chanabier, G. Polito, S. Ducasse, N. Bouraqadi, L. Fabresse and P. Tesone. 'Differential testing of simulation-based VM generators'. In: SAC '22: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. Virtual Event, France, 25th Apr. 2022. DOI: 10.1145/3477314.3507171. URL: https://hal.inria.fr/hal-03783301.

[15] P. Misse-Chanabier and T. Rogliano. 'Ease Virtual Machine Level Tooling with Language Level Ordinary Object Pointers'. In: Proceedings of the 14th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages (VMIL '22). Auckland, New Zealand, 5th Dec. 2022. DOI: 10.1145/3563838.3567676. URL: https://hal.inria.fr/hal-03827632.

[16] G. Polito, N. Palumbo, P. Tesone, S. Labsari and S. Ducasse. 'Interpreter Register Autolocalisation: Improving the performance of efficient interpreters'. In: MoreVMs 2022. Porto, Portugal, 21st Mar. 2022. URL: https://hal.inria.fr/hal-03594766.

[17] G. Polito, P. Tesone and S. Ducasse. 'Interpreter-guided Differential JIT Compiler Unit Testing'. In: Programming Language Design and Implementation - PLDI 2022. San Diego, United States, 13th June 2022. DOI: 10.1145/3519939.3523457. URL: https://hal.science/hal-03607939.

[18] G. Polito, P. Tesone, J. Privat, N. Palumbo and S. Ducasse. 'Heap Fuzzing: Automatic Garbage Collection Testing with Expert-Guided Random Events'. In: International Conference on Software Testing. Dublin, Ireland, 16th Apr. 2023. URL: https://hal.inria.fr/hal-03962007.

[19] B. Verhaeghe, A. Shatnawi, A. Seriai, A. Etien, N. Anquetil, M. Derras and S. Ducasse. 'A Hybrid Architecture for the Incremental Migration of a Web Front-end'. In: ICSOFT 2022 - 17th International Conference on Software Technologies. Proceedings of the 17th International Conference on Software Technologies - ICSOFT. Lisbon, France: SCITEPRESS - Science and Technology Publications, 11th July 2022, pp. 101–110. DOI: 10.5220/0011338900003266. URL: https://hal.science/hal-03772864.

[20] M. Willembrinck, S. Costiou, A. Vanègue and A. Etien. 'Towards Object-Centric Time-Traveling Debuggers'. In: International Workshop on Smalltalk Technologies : IWST 22. Novi Sad, Serbia: ACM Digital Libraries, 23rd Aug. 2022. URL: https://hal.inria.fr/hal-03825736.

[21] O. Zaitsev, S. Ducasse, N. Anquetil and A. Thiefaine. 'DepMiner: Automatic Recommendation of Transformation Rules for Method Deprecation'. In: ICSR 2022 - 20th International Conference on Software and System Reuse. Montpellier, France, 15th June 2022. URL: https://hal.inria.fr/hal-03647706.

[22] O. Zaitsev, S. Ducasse, N. Anquetil and A. Thiefaine. 'How Libraries Evolve: A Survey of Two Industrial Companies and an Open-Source Community'. In: 29th Asia-Pacific Software Engineering Conference (APSEC 2022). Virtual, Japan, 6th Dec. 2022. URL: https://hal.science/hal-03853493.

[23] O. Zaitsev, S. Jordan Montaño and S. Ducasse. 'How Fast is AI in Pharo? Benchmarking Linear Regression'. In: IWST22 - International Workshop on Smalltalk Technologies. Novi Sad, Serbia, 24th Aug. 2022. URL: https://hal.science/hal-03768601.

Scientific books

[24] S. Ducasse. *Pharo with Style*. Books on Demand, Mar. 2022. URL: https://hal.inria.fr/hal-03687860.

[25] S. Ducasse, G. Rakic, S. Kaplar and Q. Ducasse. *Pharo 9 by Example*. Books on Demand Collection / Série : The Pharo Technology Collection, 4th Mar. 2022. URL: https://hal.inria.fr/hal-03687932.

Reports & preprints

[26] S. Bragagnolo, S. Ducasse, N. Anquetil, A. Seriai and M. Derras. *Alce: Predicting Software Migration*. 2022. URL: https://hal.inria.fr/hal-03814782.

[27] M. Denker, N. Anquetil, V. Aranega, S. Costiou, S. Ducasse and A. Etien. *Project-Team RMOD 2021 Activity Report*. INRIA Lille - Nord Europe, 1st Apr. 2022. URL: https://hal.inria.fr/hal-03629450.

[28] G. Polito, S. Ducasse, P. Tesone, L. Fabresse, G. Thomas, M. Bacou, L. Lagadec and P. Cotret. *Remarkable Challenges of High-Performance Language Virtual Machines*. Inria Lille - Nord Europe, 6th Sept. 2022. URL: https://hal.inria.fr/hal-03770065.

[29] T. Rogliano, G. Polito, P. Tesone, L. Fabresse and S. Ducasse. *Technical report: Object-centric Access Control Mechanisms in Dynamic Languages*. Inria Lille Nord Europe - Laboratoire CRIStAL - Université de Lille, 22nd Sept. 2022. URL: https://hal.science/hal-03784027.

[30] T. Rogliano, G. Polito, P. Tesone, L. Fabresse and S. Ducasse. *Technical Report: Unanticipated Object Synchronisation for Dynamically-Typed Languages*. INRIA Lille - Nord Europe, 20th Sept. 2022. URL: https://hal.science/hal-03781743.

Other scientific publications

[31] S. Ducasse, G. Polito, P. Tesone, G. Thomas and L. Lagadec. *High-performance language virtual machines: an analysis and challenges*. 2nd Mar. 2022. URL: https://hal.inria.fr/hal-03770053.

[32] N. Palumbo, P. Tesone, G. Polito and S. Ducasse. 'Selecting Semi-Permanent Object Candidates in Dynamically-Typed Reflective Languages'. In: MPLR 2022 - Managed Programming Languages and Runtimes. Brussels, Belgium, 14th Sept. 2022. DOI: 10.1145/3546918.3560806. URL: https://hal.inria.fr/hal-03785536.

[33] N. M. Rainhart, G. Polito, P. Tesone and S. Ducasse. 'Analyzing the cost of safety for Vectorized bytecode in dynamically-typed languages'. In: MPLR 2022 - Managed Programming Languages and Runtimes. Brussels, Belgium, 14th Sept. 2022. DOI: 10.1145/3546918.3560803. URL: https://hal.inria.fr/hal-03784758.

[34] M. Willembrinck, S. Costiou, A. Etien and S. Ducasse. 'Time-Traveling Queries for Faster Debugging and Program Comprehension'. In: Journées Nationales du Génie de la Programmation et du Logiciel 2022. Vannes, France, 7th June 2022. URL: https://hal.inria.fr/hal-03738585.

12.3 Cited publications

[35] N. Anquetil. 'A Comparison of Graphs of Concept for Reverse Engineering'. In: *Proceedings of the 8th International Workshop on Program Comprehension*. IWPC'00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 231–240. URL: http://rmod.lille.inria.fr/archives/papers/Anqu00b-ICSM-GraphsConcepts.pdf.

[36] A. Bergel, S. Ducasse and 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, pp. 177–189. DOI: 10.1145/1094811.1094826. URL: http://scg.unibe.ch/archive/papers/Berg05bclassboxjOOPSLA.pdf.

[37] A. Bergel, S. Ducasse, O. Nierstrasz and R. Wuyts. 'Stateful Traits'. In: *Advances in Smalltalk — Proceedings of 14th International Smalltalk Conference (ISC 2006)*. Vol. 4406. LNCS. Springer, Aug. 2007, pp. 66–90. DOI: 10.1007/978-3-540-71836-9_3. URL: http://dx.doi.org/10.1007/978-3-540-71836-9%5C_3.

[38] A. Bergel, S. Ducasse, O. Nierstrasz and R. Wuyts. 'Stateful Traits and their Formalization'. In: *Journal of Computer Languages, Systems and Structures* 34.2-3 (2008), pp. 83–108. DOI: 10.1016/j.cl.2007.05.003. URL: http://dx.doi.org/10.1016/j.cl.2007.05.003.

[39] A. P. Black, N. Schärli and 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. Oct. 2003, pp. 47–64. DOI: 10.1145/949305.949311. URL: http://scg.unibe.ch/archive/papers/Blac03aTraitsHierarchy.pdf.

[40] G. Bracha and 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, pp. 331–344. URL: http://bracha.org/mirrors.pdf.

[41] D. Caromel and J. Vayssière. 'A security framework for reflective Java applications'. In: *Software: Practice and Experience* 33.9 (2003), pp. 821–846. DOI: 10.1002/spe.528. URL: http://dx.doi.org/10.1002/spe.528.

[42] D. Caromel and 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, pp. 256–274.

[43] P. Cointe. 'Metaclasses are First Class: the ObjVlisp Model'. In: *Proceedings OOPSLA '87, ACM SIGPLAN Notices*. Vol. 22. Dec. 1987, pp. 156–167.

[44] S. Denier. 'Traits Programming with AspectJ'. In: *Actes de la Première Journée Francophone sur le Développement du Logiciel par Aspects (JFDLPA'04)*. Ed. by P. Cointe. Paris, France, Sept. 2004, pp. 62–78.

[45] S. Ducasse and T. Gîrba. 'Using Smalltalk as a Reflective Executable Meta-Language'. In: *International Conference on Model Driven Engineering Languages and Systems (Models/UML 2006)*. Vol. 4199. LNCS. Berlin, Germany: Springer-Verlag, 2006, pp. 604–618. DOI: 10.1007/11880240_42. URL: http://scg.unibe.ch/archive/papers/Duca06dMOOSEMODELS2006.pdf.

[46] S. Ducasse, T. Gîrba, M. Lanza and S. Demeyer. 'Moose: a Collaborative and Extensible Reengineering Environment'. In: *Tools for Software Maintenance and Reengineering*. RCOST / Software Technology Series. Milano: Franco Angeli, 2005, pp. 55–71. URL: http://scg.unibe.ch/archive/papers/Duca05aMooseBookChapter.pdf.

[47] S. Ducasse, O. Nierstrasz, N. Schärli, R. Wuyts and A. P. Black. 'Traits: A Mechanism for fine-grained Reuse'. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 28.2 (Mar. 2006), pp. 331–388. DOI: 10.1145/1119479.1119483. URL: http://scg.unibe.ch/archive/papers/Duca06bTOPLASTraits.pdf.

[48] S. Ducasse, R. Wuyts, A. Bergel and 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)*. Montreal, Quebec, Canada: ACM Press, Oct. 2007, pp. 171–190. DOI: 10.1145/1297027.1297040. URL: http://scg.unibe.ch/archive/papers/Duca07b-FreezableTrait.pdf.

[49] A. Dunsmore, M. Roper and M. Wood. 'Object-Oriented Inspection in the Face of Delocalisation'. In: *Proceedings of ICSE '00 (22nd International Conference on Software Engineering)*. Limerick, Ireland: ACM Press, 2000, pp. 467–476.

[50] K. Fisher and J. Reppy. *Statically typed traits*. Technical Report TR-2003-13. University of Chicago, Department of Computer Science, Dec. 2003.

[51] P. W. L. Fong and C. Zhang. *Capabilities as alias control: Secure cooperation in dynamically extensible systems*. Tech. rep. Department of Computer Science, University of Regina, 2004.

[52] M. Furr, J.-h. An and J. S. Foster. 'Profile-guided static typing for dynamic scripting languages'. In: *OOPSLA'09*. 2009.

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

[54] L. Gong. 'New security architectural directions for Java'. In: *Proceedings IEEE COMPCON 97. Digest of Papers*. Los Alamitos, CA, USA: IEEE Computer Society, 1997, pp. 97–102. DOI: 10.1109/CMPCON.1997.584679. URL: http://dx.doi.org/10.1109/CMPCON.1997.584679.

[55] M. Hicks and S. Nettles. 'Dynamic software updating'. In: *ACM Transactions on Programming Languages and Systems* 27.6 (Nov. 2005), pp. 1049–1096. DOI: 10.1145/1108970.1108971. URL: http://dx.doi.org/10.1145/1108970.1108971.

[56] G. Kiczales, J. des Rivières and D. G. Bobrow. *The Art of the Metaobject Protocol*. MIT Press, 1991.

[57] G. Kiczales and L. Rodriguez. 'Efficient Method Dispatch in PCL'. In: *Proceedings of ACM conference on Lisp and Functional Programming*. Nice, 1990, pp. 99–105.

[58] R. Koschke. 'Atomic Architectural Component Recovery for Program Understanding and Evolution'. PhD thesis. Universität Stuttgart, 2000. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIS-2000-05&mod=0&engl=0&inst=PS.

[59] S. Liang and G. Bracha. 'Dynamic Class Loading in the Java Virtual Machine'. In: *Proceedings of OOPSLA '98, ACM SIGPLAN Notices*. 1998, pp. 36–44.

[60] L. Liquori and A. Spiwack. 'FeatherTrait: A Modest Extension of Featherweight Java'. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 30.2 (2008), pp. 1–32. DOI: 10.1145/1330017.1330022. URL: http://www-sop.inria.fr/members/Luigi.Liquori/PAPERS/toplas-07.pdf.

[61] B. Livshits and T. Zimmermann. 'DynaMine: finding common error patterns by mining software revision histories'. In: *SIGSOFT Software Engineering Notes* 30.5 (Sept. 2005), pp. 296–305.

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

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

[64] M. S. Miller, C. Morningstar and B. Frantz. 'Capability-based Financial Instruments'. In: *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*. Vol. 1962. Springer-Verlag, 2001, pp. 349–378.

[65] O. Nierstrasz, S. Ducasse and N. Schärli. 'Flattening Traits'. In: *Journal of Object Technology* 5.4 (May 2006), pp. 129–148. URL: http://www.jot.fm/issues/issue_2006_05/article4.

[66] P. J. Quitslund. *Java Traits — Improving Opportunities for Reuse*. Technical Report CSE-04-005. Beaverton, Oregon, USA: OGI School of Science & Engineering, Sept. 2004.

[67] J. Reppy and A. Turon. 'A Foundation for Trait-based Metaprogramming'. In: *International Workshop on Foundations and Developments of Object-Oriented Languages*. 2006.

[68] F. Rivard. 'Pour un lien d'instanciation dynamique dans les langages à classes'. In: *JFLA96*. INRIA — collection didactique, Jan. 1996.

[69] J. H. Saltzer and M. D. Schoroeder. 'The Protection of Information in Computer Systems'. In: *Fourth ACM Symposium on Operating System Principles*. Vol. 63. IEEE, Sept. 1975, pp. 1278–1308.

[70] N. Sangal, E. Jordan, V. Sinha and D. Jackson. 'Using Dependency Models to Manage Complex Software Architecture'. In: *Proceedings of OOPSLA'05*. 2005, pp. 167–176.

[71] N. Schärli, A. P. Black and 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)*. Oct. 2004, pp. 130–149. DOI: 10.1145/1028976.1028988. URL: http://scg.unibe.ch/archive/papers/Scha04bOOEncapsulation.pdf.

[72] N. Schärli, S. Ducasse, O. Nierstrasz and A. P. Black. 'Traits: Composable Units of Behavior'. In: *Proceedings of European Conference on Object-Oriented Programming (ECOOP'03)*. Vol. 2743. LNCS. Springer Verlag, July 2003, pp. 248–274. DOI: 10.1007/b11832. URL: http://scg.unibe.ch/archive/papers/Scha03aTraits.pdf.

[73] C. Smith and S. Drossopoulou. ‘Chai: Typed Traits in Java’. In: *Proceedings ECOOP 2005*. 2005.

[74] G. Snelting and F. Tip. ‘Reengineering Class Hierarchies using Concept Analysis’. In: *ACM Trans. Programming Languages and Systems*. 1998.

[75] K. J. Sullivan, W. G. Griswold, Y. Cai and B. Hallen. ‘The Structure and Value of Modularity in SOftware Design’. In: *ESEC/FSE 2001*. 2001.

[76] D. Vainsencher. ‘MudPie: layers in the ball of mud’. In: *Computer Languages, Systems & Structures* 30.1-2 (2004), pp. 5–19.

[77] N. Wilde and R. Huitt. ‘Maintenance Support for Object-Oriented Programs’. In: *IEEE Transactions on Software Engineering* SE-18.12 (Dec. 1992), pp. 1038–1044.