Project-Team RMOD 2021 Activity Report

Marcus Denker, Nicolas Anquetil, Vincent Aranega, Steven Costiou, Stéphane Ducasse, Anne Etien

RESEARCH CENTRE

Lille - Nord Europe

IN PARTNERSHIP WITH:

Université de Lille

2021
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

Contents

Project-Team RMOD 1

1 Team members, visitors, external collaborators 2

2 Overall objectives 3
2.1 Introduction . . . 3
2.2 Reengineering and remodularization . . . 4
2.3 Constructs for modular and isolating programming languages . . . 4

3 Research program 5
3.1 Software Reengineering . . . 5
3.1.1 Tools for understanding applications . . . 5
3.1.2 Remodularization analyses . . . 5
3.1.3 Software Quality . . . 6
3.2 Language Constructs for Modular Design . . . 6
3.2.1 Traits-based program reuse . . . 6
3.2.2 Reconciling Dynamic Languages and Isolation . . . 7

4 Application domains 8
4.1 Programming Languages and Tools . . . 8
4.2 Software Reengineering . . . 8

5 Social and environmental responsibility 8
5.1 Footprint of research activities . . . 8
5.2 Impact of research results . . . 8

6 Highlights of the year 9
6.1 Awards . . . 9
6.2 Highlights . . . 9

7 New software and platforms 9
7.1 New software . . . 9
7.1.1 Moose . . . 9
7.1.2 Pharo . . . 9
7.1.3 Pillar . . . 10

8 New results 10
8.1 Dynamic Languages: Language Features . . . 10
8.2 Dynamic Languages: Virtual Machines . . . 11
8.3 Dynamic Languages: Debugging . . . 11
8.4 Software Reengineering . . . 13
8.5 BlockChain and Smart Data . . . 15

9 Bilateral contracts and grants with industry 16

10 Partnerships and cooperations 17
10.1 International initiatives . . . 17
10.1.1 Inria associate team not involved in an IIL or an international program . . . 17
10.1.2 Inria international partners . . . 17
10.2 International research visitors . . . 18
10.2.1 Visits to international teams . . . 18
10.3 European initiatives . . . 18
10.3.1 FP7 & H2020 projects . . . 18
10.4 National initiatives . . . 19
10.5 Regional initiatives . . . 19

11 Dissemination **19**
11.1 Promoting scientific activities . . . 19
11.1.1 Scientific events: organisation . . . 19
11.1.2 Journal . . . 20
11.1.3 Leadership within the scientific community . . . 20
11.1.4 Scientific expertise . . . 20
11.1.5 Research administration . . . 20
11.2 Teaching - Supervision - Juries . . . 21
11.2.1 Teaching . . . 21
11.2.2 Supervision . . . 21
11.2.3 Juries . . . 22
11.3 Popularization . . . 22
11.3.1 Internal or external Inria responsibilities . . . 22
11.3.2 Articles and contents . . . 22
11.3.3 Education . . . 22
11.3.4 Interventions . . . 22

12 Scientific production **22**
12.1 Publications of the year . . . 22
12.2 Cited publications . . . 24

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

- Stéphane Ducasse [Team leader, Inria, Senior Researcher, HDR]
- Steven Costiou [Inria, Researcher]
- Marcus Denker [Inria, Researcher]

Faculty Members

- Nicolas Anquetil [Université de Lille, Associate Professor, HDR]
- Vincent Aranega [Université de Lille, Associate Professor]
- Anne Etien [Université de Lille, Professor, HDR]
- Damien Pollet [Université de Lille, Associate Professor, until Sep 2021]

PhD Students

- Nour Jihene Agouf [Arolla SAS, CIFRE]
- Santiago Bragagnolo [Berger-Levrault, CIFRE]
- Thomas Dupriez [Université de Lille, until Sep 2021]
- Carolina Hernandez Phillips [Inria, until Sep 2021]
- Aless Hosry [Inria, from Oct 2021]
- Mahugnon Honore Houekpetodji [Cim]
- Pierre Misse-Chanabier [Inria]
- Theo Rogliano [Inria]
- Iona Thomas [Inria, from Nov 2021]
- Benoit Verhaeghe [Berger-Levrault, CIFRE]
- Maximilian Ignacio Willembrinck Santander [Inria]
- Oleksandr Zaitsev [Arolla SAS]

Technical Staff

- Christophe Demarey [Inria, Engineer, 60%]
- Sebastian Jordan Montano [Inria, Engineer, from Jul 2021]
- Sebastijan Kaplar [Berger-Levrault, Engineer, from Oct 2021]
- Soufyane Labsari [Inria, Engineer, from Oct 2021]
- Esteban Lorenzano [Inria (Pharo Consortium), Engineer]
- Hernan Morales [Inria, Engineer, until Feb 2021]
- Nahuel Palumbo [Inria, Engineer, from Dec 2021]
- Abderrahmane Seriai [Berger-Levrault, Engineer, from Oct 2021]
- Pablo Tesone [Inria (Pharo Consortium), Engineer]
- Clotilde Toullec [Inria, Engineer]

Interns and Apprentices

- Luc Bigand [Inria, from May 2021 until Aug 2021]
- Bastien Degardins [Université de Lille, from Apr 2021 until Aug 2021]
- Maxime Desmont [Inria, from Oct 2021]
- Clement Dutriez [Inria, from Feb 2021 until Jul 2021]
- Leo Frere [Université de Lille, from Apr 2021 until Aug 2021]
- Lyna Grangaud [Inria, from Oct 2021]
- Reda Idtaleb [Université de Lille, from Apr 2021 until Aug 2021]
- Mohamed Jedny [Inria, from Oct 2021]
- Soufyane Labsari [Université de Lille, from Apr 2021 until Sep 2021]
- Theo Lanord [Inria, from Apr 2021 until Aug 2021]
- Axel Marlard [Inria, from May 2021 until Aug 2021]
- Ilyas Ouardi [Inria, until Feb 2021]
- Younoussa Sow [Inria, from Oct 2021]
- Iona Thomas [Inria, from Jun 2021 until Sep 2021]

Administrative Assistant

- Aurore Dalle [Inria]

Visiting Scientists

- Quentin Ducasse [École Nationale Supérieure de Techniques Avancées, Jun 2021]
- Carolina Hernandez Phillips [NC, from Oct 2021 until Nov 2021]
- Giuseppe Pierro [Université de Cagliari Sardaigne - Italie, until Mar 2021]
- Gordana Rakic [Université de Novi Sad - Serbie, Oct 2021]

External Collaborators

- Luc Fabresse [École des Mines de Douai, until Jun 2021]
- Guillermo Polito [CNRS]

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 [47]. 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 [44]. 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 [53]. 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

[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.

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 [37, 38]. 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) [66] 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 [27]. We look for solutions to help people putting FCA to real use.

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 [54]. Until now, few works have attempted to identify layers in practice: Mudpie [68] is a first cut at identifying cycles between packages as well as package groups potentially representing layers. DSM (dependency structure matrix) [62, 67] 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 [50]. 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 [41, 69].

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 [39, 64] and classboxes [28] 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 [39, 64]. 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 [29, 36, 40, 58] and several type systems were defined [42, 52, 59, 65].

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 [31]. 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 [39], stateful [30], and freezable [40]) 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 [57]. 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 [35] then from Smalltalk [45].

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 [49]). 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 [32], as well as controlling the access to reflective features [33, 34] are important challenges. We plan to:

- Study the isolation abstractions available in erights () [55, 56], and Java's class loader strategies [46, 51].

- Categorize the different reflective features of languages such as CLOS [48], Python and Smalltalk [60] and identify suitable isolation mechanisms and infrastructure [43].

- Assess different isolation models (access rights, capabilities [61] 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 [32],
  - the use of encapsulation policies as a basis to restrict the interfaces of the controlled objects [63],
  - 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.) [60].

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

CHOICE Outstanding Academic Title 2020 for the book: *Blockchain and Web 3.0. Social, Economic and Technological Challenges.* RMoD contributed the chapter *SmartAnvil: Open-Source Tool Suite for Smart Contract Analysis.*

6.2 Highlights

- We obtained an Inria Action Exploratoire funding
- ANR JCJC OCRE was accepted
- We released Pharo 9. pharo.org
- We released Moose 90. www.moosetechnology.org

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: http://www.moosetechnology.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

Partner: Feenk

8 New results

8.1 Dynamic Languages: Language Features

Participants Steven Costiou, Marcus Denker, Théo Rogliano, Guillermo Polito, Stéphane Ducasse, Vincent Aranega.

Analyzing Permission Transfer Channels for Dynamically Typed Languages. Communicating Sequential Process (CSP) is nowadays a popular concurrency model in which threads/processes communicate by exchanging data through channels. Channels help in orchestrating concurrent processes but do not solve perse data races. To prevent data races in the channel model, many programming languages rely on type systems to express ownership and behavioural restrictions such as immutability. However, dynamically-typed languages require run-time mechanisms because of the lack of type information at compile-time. We propose to augment channels with four different permission transfer semantics. We explore two mechanisms to implement such permission transfers at run time: write barriers and partial-read barriers. To validate our approach we implemented a channel framework in Pharo, and we extended it with different permission transfer semantics. We report on performance measurements of both (a) the transfer overhead on a single object and on a graph of objects, and (b) the per-object access overhead incurred by ownership checks. This work stands as a cornerstone of future work on adaptive optimizations for permission transfer channels. [14]

Tool demo: fine-grained run-time reflection in Python with Reflectivipy. Reflectivipy is a Python implementation of sub-method, partial behavioral reflection (SPBR). SPBR provides selective reflection operations applicable to sub-elements of methods (e.g., sub-expressions). SPBR helps in run-time code instrumentation with various application, from advanced debugging to hot patching of running programs. We briefly describe SPBR and its Reflectivipy API and implementation. We illustrate Reflectivipy through two examples: first we build and demonstrate a basic object-centric debugger and describe how SPBR favors its implementation and second, we hot patch a running REST server. [21]

8.2 Dynamic Languages: Virtual Machines

Participants Guillermo Polito, Stéphane Ducasse, Pablo Tesone, Théo Rogliano, Pierre Misse-Chanabier, Carolina Phillips.

Static Basic Block Reordering Heuristics for Implicit Control Flow in Baseline JITs. Baseline JIT compilers in dynamically-typed languages often use techniques such as static type predictions to optimize common execution paths using static heuristics. Such compilations exhibit implicit slow paths, defined by the language implementation and not by a developer, representing uncommon execution paths e.g., automatic type coercions, type validations and operation reifications. At run time implicit slow paths need to be jumped over and penalize overall execution. Removing implicit slow paths from the main execution path requires code reordering techniques. However, such heuristics are generally designed to work with profiling information. Based on the insight that implicit slow paths are known at compile-time, and thus do not require runtime-profiles, we experimented with two different code reordering algorithms: Pettis-Hansen Bottom-Up augmented with static code layout heuristics, and a slow-to-end heuristic.Our results show that many micro-benchmarks improve their run time by 1.2x. Benchmarks governed by more expensive computations such as message sends or garbage collections show in general no visible performance improvement nor degradations, while very few cases show degradations of up to 1.2x. We show that such static heuristics have low performance impact at compile-time and have great potential when static type predictions are present in the JIT compiler. [24]

Profiling Code Cache Behaviour via Events Virtual machine performance tuning for a given application is an arduous and challenging task. For example, parametrizing the behaviour of the JIT compiler machine code caches affects the overall performance of applications while being rather obscure for final users not knowledgeable about VM internals. Moreover, VM components are often heavily coupled and changes in some parameters may affect several seemingly unrelated components and may have unclear performance impacts. Therefore, choosing the best parametrization requires to have precise information. We present Vicoca, a tool that allows VM users and developers to obtain detailed information about the behaviour of the code caches and their interactions with other virtual machine components. We present a complex optimization problem due to the heavy interaction of components in the Pharo VM, and we explain it using Vicoca. The information produced by the tool allows developers to produce an optimized configuration for the VM. Vicoca is based on event recording that are manipulated during offline analysis. Vicoca not only allows us to understand this given problem, but it opens the door to future work such as automatic detection of application characteristics, identification of performance issues, and automatic hinting. [15]

Cross-ISA Testing of the Pharo VM: Lessons Learned While Porting to ARMv8. Testing and debugging a Virtual Machine is a laborious task without the proper tooling. This is particularly true for VMs with JIT compilation and dynamic code patching for techniques such as inline caching. In addition, this situation is getting worse when the VM builds and runs on multiple target architectures. We report on several lessons we learned while testing the Pharo VM, particularly during the port of its Cogit JIT compiler to the AArch64 architecture. The Pharo VM presented already a simulation environment that is very handy to simulate full executions and live-develop the VM. However, this full simulation environment makes it difficult to reproduce short and simple testing scenarios. We extended the pre-existing simulation environment with a testing infrastructure and a methodology that allow us to have fine-grained control of testing scenarios, making tests small, fast, reproducible, and cross-ISA. We report on how this testing infrastructure allowed us to cope with two different development scenarios: (1) porting the Cogit JIT compiler to AArch64 without early access to real hardware and (2) debugging memory corruptions due to GC bugs. [13]

8.3 Dynamic Languages: Debugging

Participants Guillermo Polito, Matteo Marra, Maximilian Willembrinck, Steven Costiou, Anne Etien, Stéphane Ducasse, Vincent Aranega, Marcus Denker.

A debugging approach for live Big Data applications. Many frameworks exist for programmers to develop and deploy Big Data applications such as Hadoop Map/Reduce and Apache Spark. However, very little debugging support is currently provided in those frameworks. When an error occurs, developers are lost in trying to understand what has happened from the information provided in log files. Recently, new solutions allow developers to record& replay the application execution, but replaying is not always affordable when hours of computation need to be re-executed. We present an online approach that allows developers to debug Big Data applications in isolation by moving the debugging session to an external process when a halting point is reached. We introduce IDRA MR , our prototype implementation in Pharo. IDRA MR centralizes the debugging of parallel applications by introducing novel debugging concepts, such as composite debugging events, and the ability to dynamically update both the code of the debugged application and the same configuration of the running framework. We validate our approach by debugging both application and configuration failures for two driving scenarios. The scenarios are implemented and executed using Port, our Map/Reduce framework for Pharo, also introduced. [3]

Time-Traveling Debugging Queries: Faster Program Exploration. Efficiently debugging a program requires program comprehension. To acquire it, developers explore the program execution, a task often performed using interactive debuggers. Unfortunately, exploring a program execution through standard interactive debuggers is a tedious and costly task. We propose Time-Traveling Queries (TTQs) to ease program exploration. TTQs is a mechanism that automatically explores program executions to collect execution data. This data is used to time-travel through execution states, facilitating the exploration of program executions. We built a set of key TTQs based on typical questions developers ask when trying to understand programs. We conducted a user study with 34 participants to evaluate the impact of our queries on program comprehension activities. Results show that, compared to traditional debugging tools, TTQs significantly improve developers' precision, while reducing required time and efforts when performing program comprehension tasks. [17]

Practical Online Debugging of Spark-like applications. Apache Spark is a framework widely used for writing Big Data analytics applications that offers a scalable and fault-tolerant model based on rescheduling failing tasks on other nodes. While this is well-suited for hardware and infrastructure errors, it is not for application errors as they will reappear in the rescheduled tasks. As a result, applications are killed, losing all the progress and forcing developers to restart them from scratch. Despite the popularity of such a failure-recovery model, understanding and debugging Sparklike applications remain challenging. When an error occurs, developers need to analyze huge log files or undergo timeconsuming replays to find the bug. To address these concerns, we present an online debugging approach tailored to Big Data analytics applications. Our approach includes local debugging of remote parallel exceptions through dynamic local checkpoints, extended with domain-specific debugging operations and live code updating functionality. To deal with data-cleaning errors, we extend our model to easily allow developers to automatically ignore exceptions that happen at runtime. We validate our solution through performance benchmarks that show how our debugging approach is comparable or better than state-of-theart debugging solutions for Big Data. Furthermore, we conduct a user study to compare our approach with another state-of-theart debugging approach, and results show a lower time to find the solution to a bug using our approach, as well as a generally good perception of the features of the debugger. [10]

Reflectivipy: building python debuggers with sub-method, partial behavioral reflection. Building debugging tools is hard and requires powerful tools and libraries. In object-oriented technologies, it is common to use fine-grained reflection to implement debuggers. In this tool presentation, we describe how partial behavioral reflection applied to sub-elements of a method helps in the implementation of advanced debugger features. As an example, we present an implementation of object-centric breakpoints in python. [26]

8.4 Software Reengineering

Participants Stéphane Ducasse, Vincent Aranega, Guillermo Polito, Anne Etien, Steven Costiou, Benoit Verhaeghe, Nicolas Anquetil, Santiago Bragagnolo, Houékpétodji Mahugnon Honoré.

Rotten Green Tests in Java, Pharo and Python: An Empirical Study. Rotten Green Tests are tests that pass, but not because the assertions they contain are true: a rotten test passes because some or all of its assertions are not actually executed. The presence of a rotten green test is a test smell, and a bad one, because the existence of a test gives us false confidence that the code under test is valid, when in fact that code may not have been tested at all. This article reports on an empirical evaluation of the tests in a corpus of projects found in the wild. We selected approximately one hundred mature projects written in each of Java, Pharo, and Python. We looked for rotten green tests in each project, taking into account test helper methods, inherited helpers, and trait composition. Previous work has shown the presence of rotten green tests in Pharo projects; the results reported here show that they are also present in Java and Python projects, and that they fall into similar categories. Furthermore, we found code bugs that were hidden by rotten tests in Pharo and Python. We also discuss two test smells-missed fail and missed skip-that arise from the misuse of testing frameworks, and which we observed in tests written in all three languages.[1]

Reuse in component-based prototyping: an industrial experience report from 15 years of reuse. At Thales Defense Mission Systems, software products first go through an industrial prototyping phase. We elaborate evolutionary prototypes which implement complete business behavior and fulfill functional requirements. We elaborate and evolve our solutions directly with end-users who act as stake-holders in the products' design. Prototypes also serve as models for the final products development. Because software products in the defense industry are developed over many years, this prototyping phase is crucial. Therefore, reusing software is a high-stakes issue in our activities. Component-oriented development helps us to foster reuse throughout the life cycle of our products. The work presented in our paper [2] stems from 15 years of experience in developing prototypes for the defense industry. We directly reuse component implementations to build new prototypes from existing ones. We reuse component interfaces transparently in multiple prototypes, whatever the underlying implementation solutions. This kind of reuse spans prototypes and final products which are deployed on different execution platforms. We reuse non-component legacy software that we integrate in our component architectures. In this case, we seamlessly augment standard classes

From GWT to Angular: An Experiment Report on Migrating a Legacy Web Application. Berger-Levrault is an international company that developed applications in GWT for more than 10 years. However, GWT is no longer actively maintained, with only one major update since 2015. To avoid being stuck with legacy technology, the company decided to migrate its applications to Angular. However, because of the size of the applications (more than 500 web pages per application), rewriting from scratch is not desirable. To ease the migration, we designed a semi-automated migration approach that helps developers migrate applications' front-end from GWT to Angular and a tool that performs the migration. We present our approach and tool. We validated the approach on concrete application migration and compared its benefits to redeveloping the application manually. We report that the semi-automated migration offers an effort reduction over a manual migration. Finally, we present recommendations for future migration projects. [5]

GUI visual aspect migration: a framework agnostic solution. With the generalization of mobile devices and Web applications, GUI frameworks evolve at a fast pace: JavaFX replaced Swing, Angular 8 replaced Angular 1.4 which had replaced GWT (Google Web Toolkit. This situation forces organizations to migrate their applications to modern frameworks regularly so they do not become obsolete. There has been research in the past on automatic GUI migration. However, and concurrently, large organisations' applications use many dierent technologies. For example, the IT company with which we are working,

Berger-Levrault, wishes to migrate applications written in generic programming language (Java/GWT), proprietary 4th generation languages (VisualBasic 6, PowerBuilder), or markup languages (Silverlight). Furthermore, one must expect that in a few years time, new frameworks will appear and new migrations will be required. Thus, there is a need for a language-agnostic migration approach allowing one to migrate various legacy GUI to the latest technologies. None of the existing solutions allow to deal with such a variety of GUI framework. They also typically focus on a subpart of the migration (i.e.how to extract a specific GUI framework) ignoring the re-engineering/forward-engineering part of the migration (which is straightforward for a single technology). This makes it difficult to adapt these solutions to other GUI frameworks. We propose an approach to migrate the GUI part of applications. It is based on meta-models to represent the visual element structure and layout. We detail how to create both the GUI extractors and generators, with guidelines to support new markup and programming languages. We evaluate our approach by implementing three extractors and generators for web-based or desktop-based user interfaces defined with generic programming languages (Java, Pharo, TypeScript) or markup languages (XML, HTML). We comment case studies on five applications, opened and closed source, of different sizes. The implementations of our generic approach detect 99% of the widgets and identify (i.e.determine the type of the widget) of them. We give examples of the migrated GUIs, both successful and not. [6]

Risk and Complexity Assessment on the Context of Language Migration. Language Migration is a highly risky and complex process. Many authors have provided different ways to tackle down the problem, but it still not completely resolved, even-more it is considered almost impossible on many circumstances. Despite the approaches and solutions available, no work has been done on measuring the risks and complexity of a migration process based on the technological gap. In this article we contribute a first iteration on Language Migration complexity metrics, we apply and interpret metrics on an industrial project. We end the article with a discussion and proposing future works. [7]

Report From The Trenches A Case Study In Modernizing Software Development Practices. One factor of success in software development companies is their ability to deliver good quality products, fast. For this, they need to improve their software development practices. We work with a medium-sized company modernizing its development practices. The company introduced several practices recommended in agile development. If the benefits of these practices are well documented, the impact of such changes on the developers is less well known. We follow this modernization before and during the COVID-19 outbreak. We present an empirical study of the perceived benefit and drawback of these practices as well as the impact of COVID-19 on the company's employees. One of the conclusions, is the additional difficulties created by obsolete technologies to adapt the technology itself and the development practices it encourages to modern standards. [9]

Software Migration: A Theoretical Framework. Software migration has been a research subject for a long time. Major research and industrial implementations were conducted, shaping not only the techniques available nowadays, but also a good part of Software evolution jargon. To understand systematically the literature and grasp the major concepts is challenging and time-consuming. Even more, research evolves, and it does based on the assumption that many words (such as migration) have a single well-known meaning that we all share. Since since these words meanings are rarely explicit, and their usage heterogeneous, these words end up polluted with multiple and many times opposite or incompatible meanings. In our quest to understand, share and contribute in this domain, we recognize this situation as a problem. To tackle down this problem we propose a taxonomy on the sub- ject as a theoretical framework grounded on a systematic literature review. In this study we contribute a bottom-up taxonomy that links from the object of a migration to the procedure nature migration, passing by migration drivers, objectives and approaches. We contribute a classification of all our readings, and a list of research directions discovered on the process of this study. [25] [22]

Migrating GUI behavior: from GWT to Angular In a collaboration with Berger-Levrault, a major IT company, we are working on the migration of GWT applications to Angular. We focus on the GUI aspect of this migration which requires a framework switch (GWT to Angular) and a programming language switch (Java to TypeScript). Previous work identified that the GUI can be split into the UI structure and the GUI

behavioral code. GUI behavioral code is the code executed when the user interacts with the UI. Although the migration of UI structure has already been studied, the migration of the GUI behavioral code has not. To help developers during the migration of their applications, we propose a generic approach in four steps that uses a meta-model to represent the GUI behavioral code. This approach includes a separation of the GUI behavioral code into events (caller code) and the code executed when an event is fired (called code). We present the approach and its implementation for a real industrial case study. The application comprises 470 Java (GWT) classes representing 56 web pages. We give examples of the migrated code. We evaluate the quality of the generated code with standard tools (SonarQube, codelizer) and compare it to another Java to TypeScript converter. The results show that our code has 53% fewer warnings and rule violations for SonarQube, and 99% fewer for codelizer. [16]

8.5 BlockChain and Smart Data

Participants Stéphane Ducasse, Giuseppe Pierro.

Evaluating Machine-Learning Techniques for Detecting Smart Ponzi Schemes. Ethereum is one of the most popular platforms for exchanging cryptocurrencies as well as the most established for peer to peer programming and smart contracts publishing [3]. The versatility of the Solidity language allows developers to program general-purpose smart contracts. Among the various smart contracts, there may be some fraudulent ones, whose purpose is to steal Ether from the network participants. A notorious example of such cases are Ponzi schemes, i.e. a financial frauds that require investors to be repaid through the investments of others who have just entered the scheme. Within the Ethereum blockchain, several contracts have been identified as being Ponzi schemes. We propose a machine learning model that uses textual classification techniques to recognize contracts emulating the behavior of a Ponzi scheme. Starting from a contracts dataset containing exclusively Ponzi schemes uploaded between 2016 and 2018, we built models able to properly classify Ponzi schemes contracts. We tested several models, some of which returned an overall accuracy of 99% on classification. The best model turned out to be the linear Support Vector Machine and the Multinomial Naive Bayes model, which provides the best results in terms of metrics evaluation. [8]

Smart-Graph: Graphical Representations for Smart Contract on the Ethereum Blockchain. The Ethereum blockchain enables executing and recording smart contracts. The smart contracts can facilitate, verify, and implement the negotiation between multiple parties, also guaranteeing transactions without a traditional legal entity. Many tools supporting the smart contracts development in different areas are flourishing because in Ethereum blockchain valuable assets are often involved. Some of the tools help the developer to find security vulnerabilities via static and/or dynamic analysis or to reduce the Gas fees consumption. Despite the plethora of such tools, there is no tool supporting smart contracts evaluation and analysis via a graphical representation for expert developers. [11]

Analysis of Source Code Duplication in Ethreum Smart Contracts. The practice of writing smart contracts for the Ethereum blockchain is quite recent and still in development. A blockchain developer should expect constant changes in the security software field, as new bugs and security risks are discovered, and new good practices are developed. Following the security practices accepted in the blockchain community is not enough to ensure the writing of secure smart contracts. We discusses the advantages and the disadvantages of code duplication in the Ethereum blockchain ecosystem. [12]

A User-Oriented Model for Oracles' Gas Price Prediction The Ethereum blockchain is a distributed database of transactions, where the Gas Oracles suggest the users the Gas price's categories to get a transaction recorded. We xplore the idea that the Gas Oracles are based on a data-centered model which does not provide users with a reliable prediction. We present an empirical study to test the reliability of the existing Gas Oracles from both the points of view of the Gas price predictions and the existing categories. The study reveals that the Gas Oracles' predictions fail more often than advertised and shows

that the Gas price categories do not correspond to the categories set by the users. Therefore we propose a user-oriented model for the Oracles' Gas price prediction, based on two Gas price categories actually corresponding to the users' interests and a new method to estimate the Gas price. The new method, performing the Poisson regression at smaller intervals of time, predicts the Gas price to pay with a lower margin of error when compared to the actual one. The predictions based on the user-oriented model thus provide the users with a more effective Gas price to set. [4]

9 Bilateral contracts and grants with industry

Thales DMS, Brest, France
Participants: Steven Costiou, from 2020.
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 (started in 2019) thesis is to experiment 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: Honore Mahugnon Houekpetodji *Analyse multi-facettes et operationnelle pour la transformation des systèmes d'information.*
ses, millions of methods).

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

Lifeware AG, Switzerland
Participants: 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).

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:

- Departmen tof 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 Inria international partners

University of Chile, Chile
Participants: Stéphane Ducasse.
A. Bergel is working on software analyses. The Moose platform heavily uses Roassal developed in the team of Bergel. We will continue our collaboration. We will use their knowledge on profilers. Prof. J.P. Sandoval is taking a new position in Chile. We co-supervised master students and received interns. We are working on supporting transformations and refactorings as well as recovering variable names.

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-supervises 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, 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).

University of Cagliari, Italy
Participants: Stéphane Ducasse, A. Pierro.
We are working with Prof. Tonelli and Marchesi from University of Cagliari. A. Pierro split his time between the two teams.

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

We had not many visitors in 2021 due to COVID.

- Quentin Ducasse, École Nationale Supérieure de Techniques Avancées, Jun 2021
- Giuseppe Pierro, University of Cagliari, Italy, until Mar 2021
- Gordana Rakic, University of Novi Sad, Serbia, Oct 2021

10.2.1 Visits to international teams

Research stays abroad Mobility was very limited due to COVID.

- Stéphane Ducasse visited the University of Prag in November 2021

10.3 European initiatives

10.3.1 FP7 & 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

10.4 National initiatives

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

ARCAD, Lab-STICC, Brest, France
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.

ANR JCJC OCRE

- From 2022 to 2024
- Partners: RMoD, SmArtSE (UCAQ, Quebec), UX Prototyping (Thales DMS, Brest).
- Participants: Steven Costiou, 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.

10.5 Regional initiatives

CPER DATA 2: Smart Data Participants: Marcus Denker, Stéphane Ducasse, Ronnie Salgado
Funding to work on exploring advanced data models integrating provisions for traceability, revocation and ownership.

11 Dissemination

Participants The Whole Team.

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

Due to COVID, many evebts had to be cancelled.

Chair of conference program committees

- Anne Etien was PC chair of IWOR21
- Nicolas Anquetil was chair of Software evolution track for QUATIC 2021

Member of the conference program committees

- Anne Etien: ICSE 2021, ICSME 2021, SANER (ERA track) 2022
- Marcus Denker: CERCIRAS 2021
- Vincent Aranega: ICPC 2021, ICSOFT 2021, SLE 2021
- Stéphane Ducasse: ECOOP 2022

Reviewer

- Vincent Aranega: SANER 2022
- Steven Costiou: SANER 2022

11.1.2 Journal

- Stéphane Ducasse: Ecoop 2022
- Stéphane Ducasse: IST Journal
- Stéphane Ducasse: Infosid
- Anne Etien: Empirical Software Engineering Journal
- Nicolas Anquetil – Empirical Software Engineering Journal
- Nicolas Anquetil – Information and Software Technology Journal
- Nicolas Anquetil – IET Software Journal
- Nicolas Anquetil – Journal of Software: Evolution and Process

11.1.3 Leadership within the scientific community

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

GT Debugging working group of the CNRS GDR GPL
from 2020
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.

11.1.4 Scientific expertise

- Nicolas Anquetil, "Crédit Impôt Recherche" expert evaluation for MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR, DE LA RECHERCHE ET DE L'INNOVATION
- Stéphane Ducasse, "Crédit Impôt Recherche" expert evaluation for MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR, DE LA RECHERCHE ET DE L'INNOVATION

11.1.5 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

- Master: Stéphane Ducasse, Programmation orientée objet, Centrale Lille, 5h CM 5hTD
- Master: Stéphane Ducasse, Meta, Université de Lille, 12hTD
- 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, Bases de données avancées, 30h, M1, Université de Lille
- Master: Anne Etien, Metamodelisation, 30h, M2, Université de Lille
- Licence: Vincent Aranega, Programmation avancée, Polytech Lille, 35h
- Licence: Vincent Aranega, Meta, Université de Lille, 10h
- Licence: Vincent Aranega, Programmation C, 45h
- Master: Vincent Aranega, Concepts et paradigmes de programmation par la pratique, 120h
- Master: Vincent Aranega, Langages et Modèles Dédiés, 30h
- Master: Vincent Aranega, Méta-Modélisation, 30h
- Master: Steven Costiou, Programmation orientée objet, Centrale Lille, 5h CM 5hTD
- Master: Steven Costiou, Conception et modélisation objet, Polytech Lille, 12h CM 12hTD
- Master: Steven Costiou, Fondamentaux du debugging, Université de Lille, 10hCM 14hTD
- 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, Projets Agiles, 12h, L2, IUT-A, Université de Lille, France
- Licence: Nicolas Anquetil, Interfaces Hommes-Machines, 32h, L2, IUT-A, Université de Lille, France

11.2.2 Supervision

- PhD: Benoît Verhaeghe, *Support à l'automatisation de la migration d'interface d'applications Web : le cas de GWT vers Angular*, Oct 21, 2021, Anne Etien, Nicolas Anquetil [20]
- PhD: Carolina Hernández, *Tools for MicroKernels*, Nov 5, 2021. Guillermo Polito and Luc Fabresse [19]
- PhD in progress: Oleksandr Zaitsev, *Machine Learning-Based Tools to Support Software Evolution*, started Jul 2019, Stéphane Ducasse, Nicolas Anquetil
- PhD in progress: Théo Rogliano, *On multiple language kernel*, started Oct 2019, Stéphane Ducasse, Luc Fabresse
- PhD in progress: Pierre Misse-Chanabier, *Modular, green, versatile Virtual Machines*, started Oct 2019, Stéphane Ducasse, Noury Bouraqadi

- PhD in progress: Mahugnon Honoré Houekpetodji, *Multi-Facet Actionable for Information System Rejuvenation*, SPI Lille, France, Stéphane Ducasse, Nicolas Anquetil, Nicolas Dias, Jérome Sudich
- 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

11.2.3 Juries

Anne Etien

- Zakaria Ournani, Université de Lille, 7 Nov 2021
- Quentin Perez, Mines d'Ales, 13 Dec 2021
- Elles Cherfa, Université Bretagne Sud, 16 Dec 2021

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

11.3.2 Articles and contents

- books.pharo.org contains two collections of books a technological one and a textbook one. Both collections are edited by S. Ducasse.
- The booklet *Concurrent Programming in Pharo* was released [18].

11.3.3 Education

- A new version of the MOOC for Pharo was released (Stéphane Ducasse) for its fourth iteration. mooc.pharo.org.

11.3.4 Interventions

- Multiple public remote Pharo Sprints in Lille.

12 Scientific production

12.1 Publications of the year

International journals

[1] V. Aranega, J. Delplanque, M. Martinez, A. P. Black, S. Ducasse, A. Etien, C. Fuhrman and G. Polito. 'Rotten Green Tests in Java, Pharo and Python: An Empirical Study'. In: *Empirical Software Engineering* (1st Sept. 2021). URL: https://hal.inria.fr/hal-03281836.

[2] P. Laborde, S. Costiou, É. Le Pors and A. Plantec. 'Reuse in component-based prototyping: an industrial experience report from 15 years of reuse'. In: *Innovations in Systems and Software Engineering* (2nd Dec. 2021). URL: https://hal.inria.fr/hal-03462995.

[3] M. Marra, G. Polito and E. Gonzalez Boix. 'A debugging approach for live Big Data applications'. In: *Science of Computer Programming* (2021). URL: https://hal.inria.fr/hal-03358830.

[4] G. A. Pierro, H. Rocha, S. Ducasse, M. Marchesi and R. Tonelli. 'A User-Oriented Model for Oracles' Gas Price Prediction'. In: *Future Generation Computer Systems* (10th Sept. 2021). URL: https://hal.inria.fr/hal-03427370.

[5] B. Verhaeghe, A. Shatnawi, A. Seriai, A. Etien, N. Anquetil, M. Derras and S. Ducasse. 'From GWT to Angular: An Experiment Report on Migrating a Legacy Web Application'. In: *IEEE Software* (2021). DOI: 10.1109/MS.2021.3101249. URL: https://hal.archives-ouvertes.fr/hal-03313462.

[6] B. Verhaeghe, N. Anquetil, A. Etien, S. Ducasse, A. Seriai and M. Derras. 'GUI visual aspect migration: a framework agnostic solution'. In: *Automated Software Engineering* 28.2 (Nov. 2021). DOI: 10.1007/s10515-021-00284-z. URL: https://hal.archives-ouvertes.fr/hal-03256021.

International peer-reviewed conferences

[7] S. Bragagnolo, A. Seriai, S. Ducasse and M. Derras. 'Risk and Complexity Assessment on the Context of Language Migration'. In: QUATIC 2021 - 14th International Conference on the Quality of Information and Communications Technology. Faro / Virtual, Portugal, 8th Sept. 2021. URL: https://hal.inria.fr/hal-03255895.

[8] G. Ibba, G. A. Pierro and M. Di. 'Evaluating Machine-Learning Techniques for Detecting Smart Ponzi Schemes'. In: 2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB). Madrid, Spain, 31st May 2021. URL: https://hal.inria.fr/hal-03358081.

[9] H. Mahugnon Honoré, N. Anquetil, S. Ducasse, F. Djareddir and J. Sudich. 'Report From The Trenches A Case Study In Modernizing Software Development Practices'. In: IEEE International Conference on Software Maintenance and Evolution–Industrial track. Luxembourg, Luxembourg, 27th Sept. 2021. URL: https://hal.inria.fr/hal-03341735.

[10] M. Marra, G. Polito and E. Gonzalez Boix. 'Practical Online Debugging of Spark-like applications'. In: IEEE QRS 2021 : International Conference on Software Security and Reliability. IEEE QRS 2021 : International Conference on Software Security and Reliability. Hainan Island, China, 6th Dec. 2021. URL: https://hal.archives-ouvertes.fr/hal-03398772.

[11] G. A. Pierro. 'Smart-Graph: Graphical Representations for Smart Contract on the Ethereum Blockchain'. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Honolulu, United States, 9th Mar. 2021. URL: https://hal.inria.fr/hal-03358120.

[12] G. A. Pierro and R. Tonelli. 'Analysis of Source Code Duplication in Ethreum Smart Contracts'. In: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Honolulu, United States, 9th Mar. 2021. URL: https://hal.inria.fr/hal-03358152.

[13] G. Polito, P. Tesone, S. Ducasse, L. Fabresse, T. Rogliano, P. Misse-Chanabier and C. H. Phillips. 'Cross-ISA Testing of the Pharo VM: Lessons Learned While Porting to ARMv8'. In: MPLR '21, Germany. Münster, Germany, 29th Sept. 2021. DOI: 10.1145/3475738.3480715. URL: https://hal.inria.fr/hal-03332033.

[14] T. Rogliano, G. Polito, L. Fabresse and S. Ducasse. 'Analyzing Permission Transfer Channels for Dynamically Typed Languages'. In: DLS 2021 - 17th ACM SIGPLAN International Symposium on Dynamic Languages. DLS 2021 - 17th ACM SIGPLAN International Symposium on Dynamic Languages. Chicago, France, 19th Oct. 2021. URL: https://hal.archives-ouvertes.fr/hal-03347573.

[15] P. Tesone, G. Polito and S. Ducasse. 'Profiling Code Cache Behaviour via Events'. In: MPLR '21. Münster, Germany, 29th Sept. 2021. DOI: 10.1145/3475738.3480720. URL: https://hal.inria.fr/hal-03332040.

[16] B. Verhaeghe, A. Shatnawi, A. Seriai, N. Anquetil, A. Etien, S. Ducasse and M. Derras. 'Migrating GUI behavior: from GWT to Angular'. In: International Conference on Software Maintenance and Evolution. International Conference on Software Maintenance and Evolution. Luxembourg city, Luxembourg, 27th Sept. 2021. URL: https://hal.archives-ouvertes.fr/hal-03341866.

[17] M. Willembrinck, S. Costiou, A. Etien and S. Ducasse. 'Time-Traveling Debugging Queries: Faster Program Exploration'. In: International Conference on Software Quality, Reliability, and Security. Hainan Island, China, 6th Dec. 2021. URL: https://hal.inria.fr/hal-03463047.

Scientific books

[18] S. Ducasse and G. Polito. *Concurrent Programming in Pharo*. 28th Sept. 2021, p. 52. URL: https://hal.inria.fr/hal-03358770.

Doctoral dissertations and habilitation theses

[19] C. Hernández Phillips. 'Bootstrap-Based Language Development: Turning an existing VM into a polyglot VM'. Université de Lille; IMT Lille Douai, 5th Nov. 2021. URL: https://tel.archives-ouvertes.fr/tel-03511998.

[20] B. Verhaeghe. 'Incremental Approach for Application GUI Migration using Metamodels'. Université de Lille, 21st Oct. 2021. URL: https://tel.archives-ouvertes.fr/tel-03428543.

Reports & preprints

[21] V. Aranega, S. Costiou and M. Denker. *Tool demo: fine-grained run-time reflection in Python with Reflectivipy*. Inria, 2nd Dec. 2021. URL: https://hal.inria.fr/hal-03463035.

[22] S. Bragagnolo, N. Anquetil, S. Ducasse, A. Seriai and M. Derras. *Software Migration: A Theoretical Framework (A Grounded Theory approach on Systematic Literature Review)*. Inria Lille Nord Europe - Laboratoire CRIStAL - Université de Lille, 2021. URL: https://hal.inria.fr/hal-03171124.

[23] M. Denker, N. Anquetil, V. Aranega, S. Costiou, S. Ducasse, A. Etien and D. Pollet. *Project-Team RMoD 2020 Activity Report*. INRIA Lille, 8th July 2021. URL: https://hal.inria.fr/hal-03281442.

[24] G. Polito, S. Ducasse and P. Tesone. *Static Basic Block Reordering Heuristics for Implicit Control Flow in Baseline JITs*. 4th Aug. 2021. URL: https://hal.archives-ouvertes.fr/hal-03313586.

Other scientific publications

[25] S. Bragagnolo, N. Anquetil, S. Ducasse, A. Seriai and M. Derras. *Appendix To Software Migration: A Theoretical Framework A Grounded Theory approach on Systematic Literature Review*. 15th Mar. 2021. URL: https://hal.inria.fr/hal-03169377.

[26] S. Costiou, V. Aranega and M. Denker. 'Reflectivipy: building python debuggers with sub-method, partial behavioral reflection'. In: GPL 2021 - Génie de la Programmation et du Logiciel : Journée du Groupement de Recherche. Online, France, 14th June 2021. URL: https://hal.inria.fr/hal-03435233.

12.2 Cited publications

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

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

[43] 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.

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

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

[46] 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.

[47] 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.

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

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

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

[55] 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.

[56] 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.

[57] 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.

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

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

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

[61] 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.

[62] 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.

[63] 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.

[64] 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.

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

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

[67] 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.

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

[69] 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.