Evref: Reflective Evolution of Ever-running Software Systems

Stéphane Ducasse

▶ **To cite this version:**

Stéphane Ducasse. Evref: Reflective Evolution of Ever-running Software Systems. Inria Lille - Nord Europe. 2023. hal-04527877

HAL Id: hal-04527877
https://hal.science/hal-04527877

Submitted on 31 Mar 2024

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.

Evref: Reflective Evolution of Ever-running Software Systems

Team leader: Stéphane Ducasse
Inria research center: Lille - Nord Europe

Field: check https://www.inria.fr/en/list-projects-teams-fields-and-themes

Theme: check https://www.inria.fr/en/list-projects-teams-fields-and-themes

In partnership with CRIStAL (UMR 9189, Université de Lille) and Berger-Levrault DRIT

Genealogy: this team is a follow-up of rmod

Summary: The objectives of Evref are to study and support the continuous evolution of large software systems in a holistic manner following three main axes: (1) analyses and approaches for migration and evolution of existing (legacy) systems, (2) new tools to support daily evolution, and (3) infrastructure to build language runtimes to support software evolution, new tools, frugal systems, and security language features. In the context of the first axis, Evref proposes a specific research agenda with Berger-Levrault R&D.

Evolving large software systems is a challenge. Decades of academic research have *somehow* produced tools and platforms that help companies to maintain their systems. But keeping legacy systems active and relevant is still a really complex task [?]. An aggravating challenge is that some of these systems can never stop (production lines, wafer production systems, auction managers, etc) and need to be updated while running at production sites. In addition, because the production environment is not the same as the development environment, the only way to spot and fix a bug is often by directly accessing software *in production, while running.*

Supporting the evolution of such *ever-running* systems is an important challenge for our industry because it must deal with more and more changing requirements and the need for dynamic adaptation. To address this challenge Evref will work on (1) *analyses and approaches based on language-specific metamodels and their accompanying processes* such as test generation, semi-automated migration, or business rule identification; (2) *new generation debuggers, profilers, and tools for reverse engineering* — we will tackle new areas such as the support for non-functional requirements (robustness, memory consumption, ...) —, (3) *language and runtime infrastructure to support evolution, green computing, security, and tooling* as a step towards self-evolvable runtimes. Evref approach is reflective in the sense that controlling the underlying execution engine it will explore different facets of evolution and tooling. From that perspective its sum will be more than its parts. A specific research agenda with Berger-Levrault will take place in the context of the first point.

Intended audience: The expected reader of this document is a software engineering researcher or language implementors. Still, we encourage experts from other fields to read it.

About the document size: We are sorry but the process basically forced us to produce a long document. Its initial version was a lot shorter but after each feedback step, reviewers wanted to know more.

Proposal outline:

Last updated: 31/03/2024

Contact: Stéphane.Ducasse@inria.fr

1 Team members

The team is composed of members of the previous RMOD team. Some members will work in the partnership with Berger-Levrault. EVREF continues to work on software evolution and language design but also opens its horizon to new research axes: how to rethink traditional tools to help software developers, how to build Virtual Machines and how such an infrastructure can help building new tools and more evolvable languages. In addition EVREF goes deeper in the use of IA for Software engineering challenges.
Steven Costiou (the only french researcher in debuggers) and Vincent Arenaga expertise on building tools reinforces the old RMOD team (Nicolas Anquetil, Marcus Denker, Stéphane Ducasse and Anne Etien). Finally, for several years now, Guillermo Polito invested in Virtual Machines construction. He became one of the only Virtual Machine researcher in France, his presence in EVREF is essential.

1.1 Permanent members

We list separately the members of the Evref-BL subproject with the R&D members of Berger-Levrault for the sake of clarity.
Faculty:

- **Head**, Stéphane Ducasse, HDR, DR1, Inria, Team leader, 2007 – ...
- Anne Etien, Professor, Univ. Lille, 2012 – ...
- Nicolas Anquetil, HDR, Associate Professor, Univ. Lille, 2009 – ...
- Marcus Denker, CRCN, Inria, Researcher, 2009 – ...
- Steven Costiou, CRCN, Inria, Researcher 2019 – ...
- Guillermo Polito, CRCN, Inria, Researcher 2022 – ...

Research Engineers:

- Christophe Demarey, Inria, Engineer 60%

1.2 Contractual members

Engineers:

- Clotilde Toullec, Inria, Engineer, June 2021 – June 2025
- Sebastian Jordamont, Inria, Engineer, July 2021 – July 2023

Pharo consortium engineers:

- Esteban Lorenzano, Inria, Engineer
- Pablo Tesone, PhD, Inria, Engineer

Ph.D. students:

- Théo Rogliano - (50 % S. Ducasse, 50% L. Fabresse - IMT), Multiple language kernels, 10/2019.
- Pierre Misse - (50 % S. Ducasse, 50% N. Bouraqadi - IMT), Transpilation for Modular Green VMs, 10/2019.
- Honoré Mahugnon - (50 % S. Ducasse, 50 % N. Anquetil) - CIM CIFRE[1] - Powerbuilder 05/2019.
- Oleksandr Zaytsev - (50 % S. Ducasse, 50 % N. Anquetil) - Arolla CIFRE - ML and Software Evolution, 07/2019.
- Santiago Bragagnolo - (50 % S. Ducasse, 50 % N. Anquetil) - Berger-Levrault CIFRE, Automatic software migration, 07/2020.
- Nour-Dijhene Agouf - (50 % S. Ducasse, 50 % A. Etien) - Arolla CIFRE, DSL for visualizations, 01/2021.
- Maximilian Willembrinck - (50 % S. Costiou, 50 % A. Etien), Queryable time-traveling Debuggers, 10/2020.

1.3 Berger-Levrault (Evref-BL) partnership

We list the members of the Evref-BL roadmap that we will present in the first research axis in Section **??** and their planned implications in the subproject (in bold).

- **Co-head**. Stéphane Ducasse, HDR, DR1, Inria, **20%**
- **Co-head**. Christophe Bortolaso: PhD, Head of Research, Berger-Levrault: **10%**.

[1] CIFRE is a PhD paid by a company.

- Anne Etien, Professor, Univ. Lille, **75%**
- Nicolas Anquetil, HDR, Associate Professor, Univ. Lille, **75%**
- Benoit Verhaeghe: PhD, Research manager at Berger-Levrault. **80%**. Co-Directing PhD Students during the partnership.
- Abderrahmane Seriai: PhD, Research manager at Berger-Levrault. **80%**. Co-Directing PhD Students during the partnership.
- Anas Shatnawi: PhD, R&D Engineer at Berger-Levrault. **20%**.
- Florent Mouysset: PhD, R&D Engineer at Berger-Levrault: **20%**. He will focus on IA based tests generation.

Ph.D. students:

- Santiago Bragagnolo: Industrial PhD thesis (CIFRE) in preparation. From the end of 2023 will be part of the R&D at Berger-Levrault: **80%**.

2 Context and challenges

The context of EVREF is software evolution and the tooling it requires. By tooling we mean tools but also infrastructure to execute programs. We identify three main challenges: how to support evolution, what are the missing tools and how can the runtime infrastructure can sustain all this?

2.1 Challenge 1: Software evolution is still a challenge

Computer and information systems are the key elements and exo-skeletons of our societies. They are omnipresent and manage the data of our lives and activities: insurance, billing, customer management, hospitals, HR, mass distribution, commerce, ... They are the keystone of organizations and their businesses. They have a life time expectancy of decades during which they must evolve to adapt to the world they model. Evolution is *ineluctable* [?]. In 2014, Deloitte placed reverse engineering, and evolution of software in the top 10 themes that will impact organizations in information technology.
Evolution has detrimental effects on software quality: developer turnover results in a *loss of knowledge* of the system which leads to sub-optimal or plainly wrong code modifications; new development directions may *conflict with the original goals* of the system and require to *break fundamental design* decisions resulting in architectural drift and architectural decay; general *software quality decrease* makes any evolution *more risky and more likely to introduce new bugs or new security breaches.*
Software evolution is a large research domain and we will focus on the most important or promising areas: (1) *Software migration to other languages/frameworks*, (2) *Semi-automatic library update*, (3) *Non-functional requirements during software evolution*, and (4) *Validation of produced artefacts (testing)*

2.1.1 Software migration to other languages/frameworks

The migration of software system is an important challenge faced by companies. Migration is still a really underestimated challenge [?]. Such migrations may happen between different languages for example from Delphi to C# [?]. However, it happens also between different frameworks such as UI frameworks (*e.g.,* GWT to Angular). This is a challenge for multiple reasons: (1) mismatches between the manipulated concepts (Procedural vs. OO), (2) language idioms and specific constructs, (3) library specificities (for example different event raising approaches) or even (4) paradigmatic approach between architectural styles. On top of these conceptual points, research on migration faces the fact that just supporting the migration of a single case is a large effort due to the interaction between the features. The validation of a migration is also difficult because decomposing the problem into smaller ones may not be adapted. Validation implies either to generate tests or to migrate tests setup which can be extremely costly when transpiling (for example there is no direct transformation between the setup of a simulated system and its transpiled version [?]).
We have experience in GUI migration which consists in reimplementing a GUI using a more recent framework. To do so, [?] first split the GUI into the visual code and the behavioral code: The visual code is composed of the widgets and the layout. [?, ?, ?, ?, ?, ?] and [?] proposed approaches to migrate the widgets parts. Additional work [?, ?] also deals with the layout to migrate accurately the visual code. Behavioral code requires a specific migration approach [?], and, to the best of our knowledge, no specific approach has been proposed yet. Moreover, in an industrial context, additional constraints to the existing GUI migration approach must be tackled. For instance, handling several GUI frameworks as input enable incremental migrations [?, ?]. Finally, once the application is migrated, the company still needs to maintain it. Thus, the migration must not prevent future developments but should provide support to it.

2.1.2 Semi-automatic library update

Library update is concerned with the fact that when a library evolves its clients should be updated. Many approaches have been proposed to help developers. Those approaches extract information from the source code, commit history [?], or code documentation. Kim *et al.* [?] proposed to find matches between the two versions of API by calculating textual similarities of method signatures collected from the source code of the two versions of the library. They defined a set of low-level API transformations (*e.g.,* package replacement, argument deletion, etc.) and performed a rule-based matching to find a mapping between the two versions of a library API. Xing *et al.* [?] compare two versions of the library's source code, detect changes to the API, and propose transformation rules together with working usage examples. Unlike Kim *et al.* they calculated the structural similarity of source code and not only the textual similarity of method signatures. The SemDiff tool [?], unlike other automatic approaches to library update, which compared two versions of library's source code, extracts information on a more granular level from the commit history of the library. It recommends changes to client systems based on

how the library reacts to its own evolution. Meng *et al.* [?] proposed a history-based matching approach which compares consecutive revisions of a library obtained from its commit history and supplies this information with the analysis of commit messages to generate transformation rules for client systems. Hora *et al.* [?] find method mappings between different releases of the same library. They analyzed the commit history to detect frequent method call replacements. This way, they mined the transformation rules by learning from the way the library adapts to the changes in its own API.
Teyton *et al.* [?] turned to the problem of library migration — replacing client dependency on a third-party library in favor of a competing library. They adopted and improved the approach of Schäfer *et al.* [?], but extracted method call changes from a commit history of clients that were already migrated. In their study, Alrubaye *et al.* [?] proposed a novel machine learning approach. They extracted features such as the similarity of method signatures and documentation, represented them as numerical vectors, and trained a machine learning classifier to label method mappings as "valid" or "invalid". Alrubaye *et al.* [?] mined the commit history of client systems that were already migrated from one third-party library to a different one and generated mappings for method replacements.
Even with all the previous works, it is really a real hurdle for companies to effectively migrate software and most of the time they solve just a part of the problem. The challenge still exists.

2.1.3 Non-functional requirements during software evolution

Software development methodologies strive to support the development of safe and secure software. However, ensuring security and safety *in the long run* is hard. Regularly, new security threats are discovered that must be corrected in *existing* systems. In addition, some threats are *system* or *application domain-specific*. For these, the number of security experts is much smaller which makes it more difficult to find and correct the threats. Concurrently, software systems are living entities that *must evolve* to accompany the needs of their users (new functionalities, new regulations, new running environments, . . .). As any Non-Functional Requirements (NFRs), safety/security issues introduce a real challenge to evolve software systems because they are *application specific* and *often not explicit and tractable* at the code level. Developers are often left to *trial and error or hacks*.
A large body of research exists to support the evolution of large software systems. The existing tools are generic and difficult to tune to specific contexts. Developers are often left with little, *ad hoc*, tooling.

1. A large body of research is concerned with software quality. Companies start to use open-source static analyzers and dashboard such as Sonar, PMD, FindBugs (for Java). However, while technical debt and design flaws are now well known, security and safety issues are still *not linked to the general notion of software evolution*. An important aspect that did not receive enough attention is the fact that old and evolving applications are more fragile with respect to security and safety breaches.

2. From an evolution standpoint, NFRs (safety/security, efficiency, memory) are difficult to handle in a generic way because they are often "hidden", being a by-product of implementation decisions all over the source code. There is *no general methodology and tools to handle them during software evolution*. NFRs are well understood at the modeling phase but *little work considers NFRs during maintenance and evolution*. Some work started to offer specific profiling tools [?].

   There is no unified tools for developers to get at the same time (i) an overview of NFRs at the level of a large application, and (ii) fine-grained understanding of a local and specific situation [?]. Our experience working with team of developers is that the cost of formulating and validating an hypothesis at the level of a NFRs is prohibitive: developers may lose weeks trying to understand and characterize a situation. There is *no tool to express and query NFRs and give an understanding* within the developer focus.

3. Security threats are often managed as external information: databases of attacks are used by tools but (i) they can only identify what has *been already reported*, and (ii) there is *a gap between the threats and their manifestation at the code level*.

4. Previously identified threats in security databases are generic, *not considering the specific domain or context* of a software application. Organizational security analysts recognize the need for Domain-Based Security approaches (*e.g.,* [?]), but security threat databases only cover generic security flaws. We identified a similar issue in the past with bad smell detection tools [?].

2.2 Challenge 2: Daily development tooling is still in its infancy

The second challenge of EVREF is to consider that software developers should be equipped with adequate tools to perform their tasks. Even if some advances were made in the areas of refactorings or metrics (technical

debt, automatic smell detection), there is still many of domains where software engineers should be helped. We identified the following three main areas we will focus on. Note that reverse engineering and program understanding challenges are also part of the first axis:
(1) *New challenges for debugging*, (2) *Multi-layer profilers*, and (3) *Reverse engineering.*

2.2.1 New challenges for debugging

Debugging is difficult and costly: developers debug more than 50% of their time [?, ?, ?, ?]. *Hard bugs* are particularly difficult to understand, and without understanding bugs, it is impossible to fix them [?]. In 1997, it was observed that more than 50% of hard bugs were difficult to understand because of a distance between their source and the observable error they produce [?]. In other words, it is hard to identify the source of hard bugs, and thus to understand them. This is still a difficult problem today [?, ?], as mainstream debuggers are not adapted to several hard bugs scenarios [?].
When debugging object-oriented programs, one may identify a suspicious object that needs to be investigated. However, mainstream debuggers only provide a call-stack based perspective that shows the executed code in sequential order (what code called what code). *From stack-based perspectives, it is difficult to know where to apply debugging operations* because one cannot know at run time where a suspicious object is used in the code. A typical example is the debugging of complex user interfaces where thousands of objects of the same class are involved. Traditional tools such as breakpoints are impractical to use: it is impossible to break the execution for each one of the objects. To debug only the right object, developers have to manually filter the objects by inserting complex conditional instructions into the source code. In the case of hard bugs, it is common that they cannot express those conditions due to a lack of information or tools [?].
Another common problem with hard bugs is their *reproducibility* [?]. Reproducing bugs traditionally requires to stop the program and run it again many times under controlled parameters to narrow down the exact conditions causing the bug [?, ?]. But *some real world hard bugs are not easily reproducible* because they happen only from time to time, under obscure conditions [?, ?, ?]. This hinders their understanding and makes bug fixing extremely hard.

2.2.2 Multi-layer profilers

Profiling program is an important activity that is often relegated to second-zone practice. However, it is central to understand the actual behavior of programs. The performance of a given application can be measured in terms of speed, memory, or other resource consumption (*e.g.,* disk, network, energy) [?, ?, ?].
The action of profiling a program introduces a measuring perturbation. The action of measuring impacts the results of the measure. For example, collecting samples of a program introduces pauses in the execution of the program, so these pauses affect the measured time. It is not possible to minimize the impact of measuring problems and maximize the precision of the measure [?, ?, ?].
For minimizing the impact of the measuring problem, there are different techniques that combine different levels of measuring impact and precision. For example, timing-based sampling may be affected by the measurement granularity. If the sampling time is bigger than the execution time of some pieces of code, these pieces of code are never registered by the profiler [?, ?, ?]. Profiling techniques are categorized in the following general families:

Timing-based sampling techniques. Every given period, They use a parallel thread/process to take samples of the process stack under analysis. Their precision depends on the sampling rate and the granularity of the code under analysis. This technique does not modify the existing code [?, ?, ?].

Instrumentation techniques. These techniques instrument the code under analysis to interweave the collection of samples. These techniques allows the user to get precise information but can affect the overall execution performance of the code under analysis [?, ?, ?, ?].

Memory leak detection techniques. They include variation of timing techniques, but with the objective to detect the allocation and free of memory structures. They are used to detect object life-cycle and provide dynamic escape-analysis information [?, ?, ?, ?, ?].

Event-based techniques. They are based on the generation and storage of discrete events during the execution of the analyzed program. The resulting set of events is later analyzed to reconstruct the behavior of the analyzed application. They allow the detection of existing relations between the events and complex interactions of the given program. They require to modify the runtime environment to generate a rich set of events [?, ?, ?].

Hardware-assisted techniques. To overcome some of the limitations, some techniques use hardware specific features to measure. They use information available in the architecture such as execution counters, cache

statistics, and memory manager events [?, ?, ?, ?, ?]. These same techniques can be extrapolated to get information from Virtual Machines [?, ?].

Finally, there is a large body of work on the data analysis of profiling. Bergel proposed [?] to use profiler to identify where caches can get an impact. Sandoval [?, ?] worked on identifying speed regression between consecutive versions. Bertuli *et al.* used metrics to quantify the mass of (often repetitive) generated information [?, ?, ?]. Nagarajan *et al.* used profiling information to improve the automatic-refactoring experience [?]. And, profiling has been used to measure the power performance of different algorithms and implementations [?, ?].

2.2.3 Reverse engineering

Program understanding is a large and important field. It ranges from identifiers identification to software visualization. Many publications analyze program identifiers [?, ?, ?] and some focus on the analysis of class names [?, ?]. For instance, Osman *et al.* show, through a survey that involved 32 developers, that good class names improve comprehension and are within the most important elements in class diagrams [?]. On the contrary, when a class is badly named, a developer may have to check carefully its definition and analyze how it is related to its superclass [?]. For example, quickly understanding whether an abstraction is a model or a view in a MVC triad is key to avoid mistakes or misinterpretations. Butler [?] shows that bad identifiers affect code quality and is correlated to bugs. There is a plethora of publications about program visualization [?, ?, ?]. S. Ducasse worked on some program visualizations: class blueprints to understand the internal logic of a class [?], polymetric view (to get a first understanding of hierarchies) [?], package blueprints (to help remodularizing), distribution maps (to understand the spread of a property on a system) [?]. More recently, Merino *et al.* [?] proposed an ontology to discover visualizations. This work reinforces the idea that there is a large body of research and that visualizations should be adapted to the task. Recently Slater *et al.* [?] proposed corpusVis to analyze the metrics of large system. There is also work on evaluating visualizations [?, ?, ?, ?, ?, ?, ?, ?].

2.3 Challenge 3: Language virtual machine development incurs high cost

The third EVREF axis covers Language Virtual Machines (VMs) *i.e.,* the environment needed to execute programs (composed of a virtual machine and language core). Language Virtual machines allow application portability between different platforms and better usage of resources, making them an ideal target for programming language implementation. For example, they are needed to run Java, Javascript or Python. For these reasons language VMs are nowadays pervasive in every laptop, server, web browser, and smartphone, and are used in critical applications such as stock exchange, banking, insurance and health [?].
As such, Virtual Machines are important as a vector of research in other fields and they cross-cut the other axes of EVREF. For example, we will design dedicated virtual machines for new tools, ever-running systems, security and green computing. It is then crucial to be able to easily build customized Virtual Machines. The field is large so we focus on (1) *Virtual machine construction* and (2) *Dynamic and adaptive optimizations in virtual machines.*

2.3.1 Virtual machine construction

Recent work acknowledges that Virtual Machine construction incurs a high cost in practice [?] because of the complexity and inter-dependence of its many components [?] *i.e.,* interpreter, compilers, garbage collectors, concurrency model are all tailored to each other in the search for performance. It is indeed a case where the whole is more than the sum of its parts. Virtual Machines are indeed highly complex engineering pieces, often handcrafted by experts, that mix state-of-the-art compilation techniques with operating-system resource management. However, besides some well-known techniques, published in research venues, most knowledge and technology around virtual machines is concentrated in large companies such as Microsoft, Google, and Oracle, making Virtual Machine construction difficult, and experiments on them difficult to reproduce and replicate.
Wimmer *et al.* argue that components can be pre-built and, up to some degree, generated automatically from specifications. In this direction, VMKit [?] and MMtk [?] attempt to provide generic components that can be used to build Virtual Machines. The LLVM compiler infrastructure [?] provides generic components to build compilers, but is more adapted to ahead-of-time compilation scenarios and thus generally not used in Virtual Machines.
In contrast, other solutions propose to generate components. For example, Pypy [?] generates ahead-of-time a language JIT compiler from a bytecode interpreter, and Truffle [?] generates at runtime a language JIT compiler from an AST bytecode interpreter. In the opposite direction, recently [?] proposed to generate an interpreter from a compiler. However, these specifications are partial: they mainly allow one to refine the execution engine,

showing a degradation in memory management in some cases [?], and their underlying technology lies in some cases in the control of private companies. It is also worth noticing that most efforts cited above target only speed improvements. However, in nowadays applications other constraints have emerged, such as space and energy efficiency, for example in the fields of IoT, robotics and green computing [?].

There is a need for a better way to design and build dedicated virtual machines. Such virtual machines should be able to expose specific information to tool builders or language designers to model new security abstractions, to support the evolution of ever-running system or to propose alternate consumption power for green computing or IoT devices.

2.3.2 Dynamic and adaptive optimizations in virtual machines

Virtual Machines achieve high performance thanks to aggressive optimization techniques that observe and adapt the execution through runtime adaptations such as just-in-time compilation [?]. Inline caches [?] and polymorphic inline caches [?] are dynamic compiler optimizations for object-oriented code, where function calls depend on the receiver types. Using these techniques, code is first compiled using a generic slow version, and dynamically-linked using code patching techniques based on types observed at runtime. Inline caches show very good results at optimizing polymorphic code by avoiding the cost of method lookups.

Polymorphic inline caches are useful for capturing fine-grained runtime statistics per call-site, needed by speculative optimizers [?]. Speculative or adaptive optimizers use the execution statistics and generate specialized machine code by speculating on previous execution patterns: the types found, the paths taken. Different JIT compilation strategies and frameworks have been shown effective for different kinds of applications. Method-based compilers [?] take as compilation units a single method. Method granularity is a popular approach amongst compilers because methods have well-established boundaries. Trace-based compilers [?, ?] take linear execution traces that start at any point during the execution and end at any other point, even cross-method, and specialize those traces that are very common during the execution. Region-based compilers [?] use *regions of code* such as inter-method traces and loops as their compilation units.

Adaptive optimizations and compilers present many challenges that make their usage and implementation difficult in practice [?], for example:

Statistics quality vs. runtime profiling cost. Execution statistics are taken at run-time using similar techniques than those explained in Section ??. However, runtime-profiling is generally the cause of large execution overheads, especially when done improperly. For example, many works report slowdowns ranging between 30%-100%, and some report overheads of over 100x [?]. At the same time, incomplete or misleading statistics can be the cause of performance degradations.

Generated code quality vs. compilation cost. Several techniques have been developed over the years to improve compiled code quality with low compilation costs. For example, selective optimizations and compilations propose to optimize only a sub-set of the code [?]. Code specializations and customizations propose to generate specialized versions of code depending on several runtime properties such as receiver types or runtime values observed [?]. More recently, Chevalier-Boisvert *et al.* proposed *lazy block versioning*, *i.e.,* to use multiple versions of basic blocks as a way to avoid type checks [?, ?].

3 A holistic research agenda articulated around 3 axes

Evref is built around a holistic vision of the eternal software challenge. It acknowledges that we need to be able to work on different levels to support the evolution of software under different scenarios. The fact that we will work on a full software stack (still making progress in each area) creates a situation where the team will be in the position to think and propose solutions that would not be possible otherwise. The reflective stress in the project title is that the axes can reflect and influence each other and can help each other in client/provider of problems or solutions.

The agenda defined with Berger-Levrault offers a set of evolution challenges faced by real business units of information systems. Such challenges are still unresolved challenges that any software company has to struggle with: testing to control migration, migration to new technology, business rule identification and software maps are key challenges. In addition and while this is not considered within the BL axis, it should be noted that evolution can happen when the software to be updated is running and that migration should happen while the system is executing.

3.1 Research axes within EVREF

The research axes in EVREF are built to form an articulated whole around the challenges of evolution of constantly changing and running systems. They are based on the experience and research made in RMOD during 12 years. The three axes are interconnected, often in client relationships *e.g.,* profilers requires low-level information provided by virtual machines but virtual machines require advanced profilers. Controlling virtual machines opens the doors of many possibilities both at the level of tools but also language design for isolation or security.

- **Axis 1 – Evolution of ever-running systems.** This axis is about how to effectively evolve large and complex software. This covers a large spectrum of topics such as visualization, metrics, analysis, ... This includes for example migration from one language to other one or from a library version to another one. This is within this axis that the team will work in partnership with Berger-Levrault. The axis is built around the Moose platform and its current redesign effort.
- **Axis 2 – New generation tools for daily tasks.** This axis is about how to offer advanced tools for everyday development: it focuses on debuggers, profilers and tools to reverse engineer code. It follows the work around debugging started in RMOD.
- **Axis 3 – A generative approach to modular and versatile virtual machines.** This axis is about how to improve the building of virtual machines to support their exploration and application to tools, security, green computing, ... This axis is also providing infrastructure for the other axes. The exploratory action is an important basis for this axis. In addition, interactions with the Pharo consortium engineers and the use of the industry level Pharo virtual machine will naturally take place.

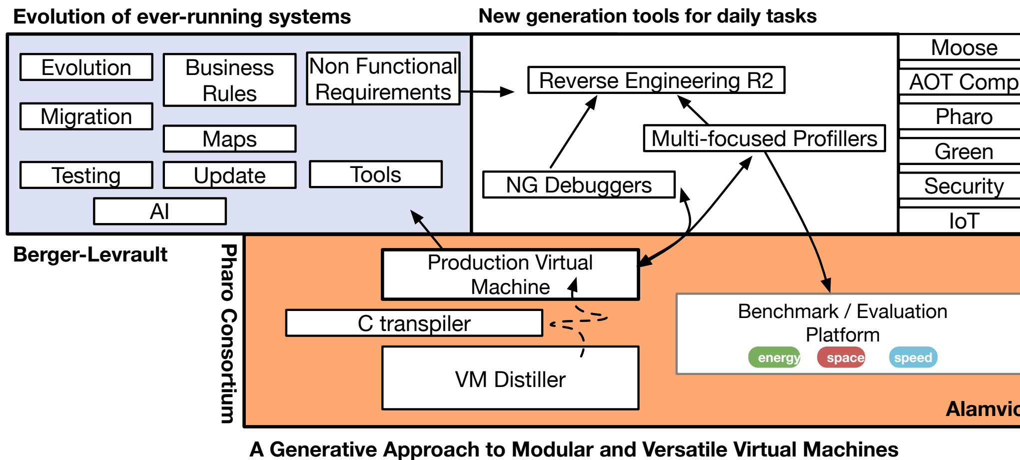

Figure 1: EVREF' vision: Three interacting axes.

There are possible and welcomed overlaps between areas covered in the interaction with Berger-Levrault: for example transpilation is the basis of the Pharo VM compilation tool chain, migration is a topic of interest for Berger-Levrault. Still we list such items in their respective axis because the research agenda of EVREF is larger than its interaction with Berger-Levrault. A cross-fertilization oo the same topic will naturally happen but without one taking over the others.

The third axis, *A generative approach to modular and versatile virtual machines*, will also support the other axes by exposing specific runtime information (such as exposing Polymorphic inline caches, possibility of instrument object creation,...), or offering the possibility to extend the virtual with new or modified low-level functionality. It will also take into account the needs and feedback from the tool builders.

In each of the axes, we expect to have several PhDs, build research prototypes with potential transfer to the corresponding platforms.

EVREF is the continuity of RMOD with the emergence of two new domains. The main differences (besides the roadmap with Berger-Levrault) are the emphasis on *new tools (axis - 2)* and *virtual machine design(axis - 3)* in place of the *language design* axis of RMOD.

3.2 About EPC and synergy with Berger-Levrault

EVREF is one of the first joint team with a SME. This is new and this raises challenges. We are not naive but the RMOD team members agreed that this adventure is worth. In addition we will do retrospective analyses. Even though intellectual property issues, access to private data, and actual work with the Berger-Levrault team could have happened via bilateral contracts as we did in the past, having a joint team has a real added value at multiple levels:

- **Gain of energy.** When we sign bilateral contracts, there are always details that should be re-examined. With the current setup the team sets a frame once for all.
- **Cross fertilisation.** Even we have designed specific topics to work on, our seminar will systematically include works from both sides that are not related to the exact topics of the collaboration. For example, we plan to present the transpilation chain of the Pharo VM and its challenges, the work on Concolic Testing but also any work related to test analysis, selection, and new tools.
- **Closer connection.** We will have regular seminars, workshops and visits. In addition, B. Verhaeghe previously RMOD's PhD will be member of Berger-Levrault R&D. Other EVREF students or engineers will have the possibility to present their work and get in touch with the company. In the opposite, some members of Berger-Levrault R&D can benefit from a more experimental research process and possibly enroll in a PhD thesis.
- **Financial stability.** Finally, not having to look for contracts to be able to do what we planned is a chance for the team to get focused and work. In addition, being able to hire engineers will push the Moose platform. In addition making sure that Moose can be used by others will favor its further adoption.
- **About freedom.** RMOD has a long history and practice of working with companies, it will continue. The intellectual property model that we use makes sure that all the partners of the team benefit from all the results and development efforts. Therefore, we create a clear win-win situation and avoid a competition between our partners. Berger-Levrault will benefit, as any other partners, from the development effort and results made in the context of other contracts.

EVREF will benefit from the synergy with Berger-Levrault R&D. Now EVREF's roadmap has been designed to be a superset of the roadmap identified with Berger-Levrault. We believe that it will give enough freedom to the team members to work on challenges of interest and have access to shared resources to achieve this goal.

3.3 Axis 1 – Evolution of ever-running systems

Members: N. Anquetil, S. Ducasse, A. Etien — on some subtopics (tests/bugs/transpilation): G. Polito, P. Tesone, M. Denker and S. Costiou

Supporting software evolution is an important and challenging topic inherently linked to software. Indeed, software models the world and the world is changing. Therefore software evolution is ineluctable. In EVREF we will work on *fundamental* aspects of software evolution:

- **Towards automatic evolution.** We will work on supporting semi-automated evolution in the case of libraries update. We will extend our work and support both the library developers and their users to migrate to more recent versions by analyzing past activities and learning automated rules.
- **Migration.** We will enhance our metamodel-based approach of front-end migration to support interlanguage migration.
- **Non-functional requirement identification and extraction.** To help developers during their maintenance tasks we will take into account non-functional requirements (NFR) and propose software maps and reverse engineering techniques to reveal such hidden software aspects.
- **NFR aware code transformations.** We will extend refactorings to support domain-specific and non-functional requirements.

First period PhD: Within the first period[2] of the team we will propose a PhD on *"Non-functional requirement identification"*.

[2] A team is evaluated every four years and we consider it as a period.

3.3.1 Towards semi-automatic evolution

We focus on how to automatically identify mandatory changes to migrate an application from one version of the library it uses to another. We will extend our previous work on machine learning techniques based on association rule discovery to support the identification and automatic inference of automated migration rules using a language semantics analysis [?]. We want to extend this approach to the full stack (taking into account deprecation, API changes, and library migration).

Learning migrations from past activities. We will extend our preliminary work on the automated generation of automatic transformation rules. Our approach takes the developer activities made on *a system* to generate migration rules to be applied to *the clients of the system*. For example, we analyzed all the activities to produce Pharo 3.0 from Pharo 2.0 and we learned transformation rules to apply to systems built on top of Pharo 2.0. Our approach learns, using association rules, the most appropriate transformations to migrate from one version to the other. We will take into account language specific semantics (overloading, overwriting) as well as coarser-grained changes such as library API changes.

Learning migrations from other migrations. To complement our approach, we will learn from migrations already performed between pairs of libraries to propose new migration plans and enhance our migration model.

Library characterization and API transformation. To be able to migrate from one library to another, we need to be able to characterise it in terms of API and features if offers. We will use natural language processing techniques to characterize libraries. We will use static analysis and automata to identify more complex transformations on argument use.

Leveraging tests and dynamic information. Most of the migration strategies take a static perspective on a software system. We will enhance this with a dynamic perspective. We will take as input the fact that Unit Tests cover the migrated system. We will use program traces and dynamic information as input for our machine learning techniques. In addition, we will perform transformations while the system is running. This way we minimise false positives in dynamically typed languages such as JavaScript or Pharo.

3.3.2 Non-functional requirement identification and extraction

Non-functional requirements includes security, safety, speed/memory optimizations ... Such non-functional requirements may be captured by design document and methodologies, however their exact knowledge rarely survives fifteen years of evolution.

Quali-secure flaws identification. Security is a subset of software quality and we will treat it using software quality paradigms. From that perspective we coined the term *quali-secure flaws* to name the software quality issues that may lead to security or safety issues. Of course, the purists will say that there is a difference between safety and security issues. Yes there are. Neverthlesse, we take the perspective that in a first period identifying them requires the same logic. As such we will work on the identification of software quality issues ("bad smells" or "anti-patterns") that may lead to security or safety threat. In particular we will work on learning quali-secure flaws and not just rely on external threat databases that cannot cover in house specific concerns [?]. This will result in a list of such quality issues but also in a methodology and tools to identify system specific quali-secure flaws [?].

Operationalizable non-functional requirements. Identifying software quality issues that may lead to security threats (first point) is a way to make security *operationalizable*: able to be manipulated directly [?, ?]. Software benchmarking is a way to operationalize another NFR: efficiency under heavy load. We want to take into account different kinds of NFRs (security, efficiency, memory consumption) because they can represent other aspects of general software robustness. We will expand and generalize the results of the first point (detecting quali-secure flaws with anti-patterns) to identify other concrete manifestations of NFRs in the source code. We propose to do this by identifying specific design patterns. For example patterns of code used to save memory or to save computation time. Once NFRs are made explicit, they should be exposed to developers, so that they can take adequate actions.

From the results of the first two problems, we will propose NFR aware tools to manipulate the source code. Such tools will include: (1) tools to automatically detect security threats in the source code from software quality issues, (2) tools to support software engineers in understanding NFR (memory consumption, security, efficiency) and (3) tools to transform source code while preserving a NFR (such as security or efficiency).

3.3.3 Practical testing

Tests are a key asset to support software evolution and software development. They are also extremely important in the context of dynamically typed languages and promoted by agile methodologies. We will work on how to improve test quality (coverage), test generation but also how tests can be used as seeds for sources of dynamic type.

Bug view coverage. We will work on a *bug coverage*: Lot of works have been made on test coverage *i.e.,* how a piece of code is covered by tests. Now there is catch. Sometimes developers have tests for a given piece of code and still bugs occur inside that piece of code. We want to guide developers to enhance their tests by analysing existing tests using the *presence of bugs on the exact same tested* code. We want to understand how tests assessed according to the bugs they cover and not such the paths or other structural properties of the code.

Automatic test selection. We will revisit our previous work on automated test selection for dynamically typed languages [**?**, **?**]. The idea is to assess the heuristics to select tests to execute after a change occurred. In particular, we will also revisit the *implementation* strategies to compute selected tests: Indeed in dynamically-typed languages static analysis is not powerful enough to decide exactly how to detect dependencies (to compute which tests should be rerun), so other techniques such as program spies should be used and such spies can change program executions.

Flaky test identification. A flaky test is a test that intermittently fails often due to external reasons such as firewall, limited memory, speed limitations on test servers.... Flaky tests are difficult to identify because they do not fail all the time, and on different occasions. They force developers to allocate resources to address their failures while this is often not necessary, because flaky tests are not bogus tests per se. The Pharo consortium is facing flaky tests in its production servers and will provide data to us. We will investigate the possibilities to use AI techniques to identify them.

Test generation. Often software does not have tests or has poor tests. There is a need of approaches to support test generation on one hand and test improvement on the other (such as test amplification [**?**, **?**], carving [**?**, **?**], pseudo test [**?**]). We will investigate how AI can be applied to such areas both learning from existing tests or based on tests of the libraries that are used. A particular scenario we will study is the possibility to generate tests for the evolution of used libraries. We are currently evaluating Concolic testing to generate better inputs [**?**, **?**, **?**].

3.4 Axis 1 – Software evolution with Berger-Levrault

Members: N. Anquetil, C. Bortolaso, S. Ducasse, A. Etien, F. Mouysset, A. Seriai, A. Shatnawi, B. Verhaeghe

A special roadmap has been designed with the first Software Evolution axis to handle the interaction with Berger-Levrault R&D in the context of an EPC (Equipe Projet Commune). As its title number shows it, this roadmap is included in the first research axis.

Evref will focus on designing tools to help software teams in handling technical debts, software evolution, and migration. In fact, while tools are available on the market to design and implement new and up-to-date software, there are still a very limited number of methods and tools to deal with legacy code and its migration to new technological stacks. Managing legacy code and maintaining existing software represent nevertheless a crucial activity for the software market. This is particularly the case of administration software systems which often need to be maintained for several decades.

Following this perspective, the following topics will be the focus of our interaction and research:

- **Objective 1 - AI-based tests generation**: Automatically generate software tests from existing code and execution contexts.
- **Objective 2 - Supporting migration / tools for transpilation**: Defining a set of tools and methods to assist the migration of legacy software (language, frameworks, architectural style).
- **Objective 3 - Business rule extraction**: Being able to identify the part of code modeling business rules.
- **Objective 4 - Software systems maps**: How to extract specific information in source code for a given task.

3.4.1 AI-based tests generation

Testing is very often underestimated by software development teams because it is difficult. Writing tests is also experienced as a very time-consuming task, in particular, because tests need to be maintained along with the

source-code modifications. In practical industrial developments, the writing of test cases is often pushed aside the development of new features and correction of bugs. Yet testing remains essential to ensure that a system behaves correctly. This is also true in the case of migration (See Objective 2), for which all the unit tests have to be generated to verify that generated code is correct.
We, therefore, seek to generate unit tests for existing systems that do not have them. The problem can be broken down into (1) identifying part of code (functions, methods) to be tested, (2) generating a credible test context, (3) generating a test case (input data), and (4) generating the oracle that validates the correct execution of the system in the test case.
The objective is to use *Machine Learning* (ML) algorithms to abstract system states from real execution traces. This involves modeling the data and the problem to adapt the right ML algorithm.
The subject is rich and we believe that such a thesis will be only a first step.

3.4.2 Support for migration / transpilation

Ensuring the durability of an application in the long term is a real challenge. High-profile problems such as the Y2K bug[3] have shown that replacing these old applications is not easy. Even today, many information systems rely on applications developed in the 60s and 70s, in COBOL, a technology that is too old and yet still very lively because of the sheer size of the source code it represents. For example, according to Reuters, in the USA, 43% of American banking systems are based on Cobol, 95% of machine cash withdrawals are based on Cobol, 220 billion lines of code are used daily (http://fingfx.thomsonreuters.com/gfx/rngs/USA-BANKS-COBOL/010040KH18J/). Many Berger-Levrault's solutions are in this situation. Several BL's solutions in human resources, public finance, health care, etc. have been designed and developed decades ago. They rely on obsolete monolithic architectural styles and rich clients, old programming languages and technology such as PowerBuilder, Delphi, Visual Basic 6 and Access. Some of these solutions are used by several thousands of public administrations and represents millions of lines of code. The volume and layered complexity make their redevelopment from scratch impossible even with large amount of resources.
This is why automatic migration appears to be the right approach to migrate millions of lines of code from old languages to recent technologies. Berger-Levrault and RMOD have experienced this problem together several times and it is now the time to look for a generic and perennial solution. It is indeed very likely that these problems of technological obsolescence will occur again in 10 years with current technologies. This is a vast subject with many ramifications that will justify further work:

- Migration of architectural style and *framework*;
- Migration with a paradigm shift (*e.g.,* procedural to object-oriented);
- How to support the migration of a system, for example how to migrate its tests, or how to *debug* the generated code;
- How to define a reusable approach (*tools and methodology*) that can be instantiated every time such problem occurs with various languages and technologies.

RMOD already has some experience in this field with:

- A CIFRE thesis (at Berger-Levrault), soon to be completed, which gave very good results on the migration from GWT to Angular;
- A CIFRE thesis (at Berger-Levrault), initiated this year, also dealing with code migration;
- A thesis in-progress (independent of Berger-Levrault) tackles the transpilation from Pharo to C for our virtual machine;
- A CIFRE thesis (independent of Berger-Levrault) in progress works on automatic migration (adapting a system from one version of a library to the next). This approach could be adapted to the problem of transpilation by learning from one transpilated system to help transpilate the next.

3.4.3 Extracting business rules

In public administration, software is regularly exposed to incremental evolutions, because behavior have to stay in line with regulation and law. Consequently, after twenty years of incremental modifications and extensions of new features the source code embeds a very large quantity of knowledge. It results in complete dissemination of business logic in the code of applications. A typical example is a 20 years old medical billing management tool for which the knowledge of business rules is extremely diluted in its development teams, among users (external customers), or sometimes completely lost. Consequently, the reimplementation of such a system

[3] Year 2000 bug, when years expressed on two digits (98 for 1998) would create problems.

appears particularly tricky. Worse, the loss of know-how and subtleties relative to the dilution of this business logic in these management software systems could lead to situations where modification would be impossible or extremely complex without functional regressions. This type of scenario is unacceptable for software that must constantly adapt to changes in practices and regulations. For that type of reason, we want to build mechanisms able to re-extract and re-model the business logic from the existing source-code. This problem remains a well-known major concern underlying any code evolution or migration operation.
RMOD, on its side (independently of Berger-Levrault), initiated work in this direction:

- Submission of an ANR project on the creation of support tools for software evolution taking into account *non-functional requirements*. The project wants to enable developers to make changes to the application code by understanding their implications on non-functional requirements (*e.g.,* security, use/abuse of CPU/memory/disk/network resources).
- A thesis has been initiated in the past on the understanding of algorithms to improve their quality without impacting performance. But there is still the will to develop this subject and we are working on alternatives, for example by developing our capacities for fine code analysis.

3.4.4 Software system maps

Software systems and in particular the ones having a long life-time are *full of hidden* knowledge and design choices. Such information can be of diverse sources such as the architecture, communication protocols (socket, JSON web services, files), exposition of API, but also non-functional requirements such as the adherence to old conventions, use of old components, etc. Often such important knowledge is *simply lost*, even when developers put significant efforts in writing and maintaining documentation. Software developers responsible of evolution do not have access to the big picture and work with limited local and often *too generic* information.
Do we imagine being treated by a surgeon that would not have X-rays to get information from our body before an important surgery? This is exactly what is happening when modifying software systems. There is a clear need for a systematic approach to *extract dedicated tailorable knowledge software maps.*
Reverse engineering consists of applying specialized analyses to extract more abstract views [**?**, **?**]. These analyses are often dependent on the objective sought and the technology used in the analyzed system [**?**]. We note that the tools proposed by the scientific community are over-specialized, making it possible to meet a very specific need under very constrained conditions. To be able to leave the laboratories and enter the industrial world, such tools must be gain in flexibility and adaptability to a wider range of situations.
Extracting the key views that *enable decisions* to be made remains a scientific challenge. Indeed, these views depend on the local context (business, domain, process, constraints related to the technology used) and must take into account different sources of information such as structural information [**?**, **?**], data flows between components, bug reports [**?**], profilers [**?**, **?**] and others.
There is a plethora of works dealing with program visualization and their introduction [**?**, **?**, **?**]. S. Ducasse has worked on a certain number: class blueprints (to understand how a class is structured), polymetric view (to get a first understanding of hierarchies), package blueprints (to assist remodularization), distribution maps (to understand how a property is distributed over a system). These visualizations cover different levels of abstraction and objectives. However, they raise a certain number of questions:

- Which visualization is relevant for a given task?
- What information should be collected, filtered, and transformed for what task?
- How can a visualization be adapted for a specific case? (Which visualization for the identification of architectural deviations? For the identification of microservices and data flows between subsystems?)
- To which extent, the graphic visualization of source code and its properties will help development teams in enhancing their architectural styles, code quality, refactoring, solving bugs, etc.?

Our scientific hypothesis, to help the understanding of the applications, is to propose an infrastructure for the definition of software maps that are adaptable and scriptable by users: this adaptability will be based on two points: (1) the extensibility of a code metamodel and (2) the definition of a visualization framework based on a metamodel for visualizations.

First period PhDs: The PhDs on Test generation and Business rule extraction will start in October 2022.

3.5 Axis 2 – New generation tools for daily tasks

Members: S. Costiou, S. Ducasse, A. Etien — on some subtopics (profilers): G. Polito, M. Denker, N. Anquetil and P. Tesone (Pharo consortium)

Designing a tool is bold: it requires to have a really working solution to a problem, to have validated it, and to be able to expose it to its users. We use the term tool here, not in a reductive and pejorative definition but in its bold sense: working tools to empower end-users. We will work on tools to support developers with a focus on improving daily development tasks.

- **New generation debuggers** will propose new debugging techniques such as object-centric and back-in-time debuggers.
- **Multi-layered profilers** will rethink profilers and how systems are benchmarked.
- **Reverse engineering revisited** will revisit reverse engineering techniques taking into account non-functional requirements such as memory consumption, security concerns and others.

First Period PhDs: We will propose the following PhDs on *"Practical back in time debuggers"* and *"Advanced multi-stagged profilers"*.

3.5.1 New generation debuggers

Back-in-time debuggers record executions to exploit them afterward. A recorded execution can be navigated post-mortem, *guaranteeing to observe the exact same behavior, state, and output* as the original one. Developers are certain to deterministically observe buggy behaviors, and they can therefore study them at will. Like any system, ever-running systems are subject to non-reproducible bugs and would greatly benefit from a deterministic way of reproducing those bugs. For example, imagine a satellite having a bug when in a certain orbit position every year. Using a back-in-time debugger, *rewinding the execution to the point in time where the bug occurred avoids waiting for bug reproduction* for a year.
However, we face several challenges when recording and navigating executions of ever-running systems, whose execution never ends. First, back-in-time debuggers are not mainstream. It is unclear what are their requirements from the execution engine point of view, nor how to build robust and efficient back-in-time debuggers. Second, ever-running systems have a potentially infinite execution. The amount of recorded data cannot be anticipated and is potentially huge. Therefore, those executions cannot be fully recorded and we have to scope that recording to important debugging information. It is a major problem to define what a back-in-time debugger should record and in the same time make sure to avoid losing relevant execution data. Finally, even a scoped record of an ever-running system's execution might produce a massive data dump. Developers need to query those recorded executions to obtain the most relevant debugging information [?]. Today, little is known about querying executions for debugging, and that topic remains totally unexplored in the case of ever-running systems.

Dynamic information and advanced debugging. Tracing execution produces large amounts of data [?]. First, such information should be interpreted in the context of its static counterpart [?]. Second, it requires filtering or query languages to identify meaningful parts and bugs. Over the years new approaches to debugging have emerged: Lencevius proposed using a query language to navigate stack traces [?] and Zeller proposed automated debugging [?]. Back-in-time debuggers are examples of new ways to support debugging by supporting navigation in the past of the computation [?]. Lienhard proposes taking into account object aliases to enhance debugging [?]. More recently, Object-centric debuggers proposed interesting debugging facilities centered around single objects and execution flow scope [?]. Yet other approaches to building debuggers are emerging with *moldable* tooling [?]: debuggers can be adapted to specific frameworks or libraries.
We are building experience in back-in-time and alias-based debugging and we want to extend this with the idea of moldable debuggers to adapt to domain-specific or compilation-specific techniques. We will propose query-based languages to filter and aggregate dynamic information.

Object-centric debugging. The objective is to *deliver the full potential of object-centric debugging.* In Evref, we will set *the first generation of object-centric debuggers.* For this, we will shape systematic methods for developers to choose when and how to use object-centric debuggers for maximum debugging efficiency. These debuggers will lower the cost of tracking and understanding hard bugs in object-oriented programs.
To that end, we will cover the lack of knowledge (fundamental, practical and empirical) and definition that hinders the implementation, the evaluation, the dissemination, and the adoption of the technique. Namely, we will define how to *identify and to obtain objects to debug.* We will define and study object-centric concurrent operators for the *tracking and the understanding of concurrency bugs.* Finally, we will identify the scenarios for which object-centric debugging provides a significant advantage compared to traditional call stack-based

debugging. We will evaluate the technique through *large-scale empirical evaluations and industrial use cases.* We will build debuggers, not only prototypes but real software for real users that will be transferred into the open-source and the industrial worlds. We will build advanced debugger prototypes, but they could later become real software for real users.

3.5.2 Multi-focus profilers

We will work on the design of profilers both from a memory and execution speed point of view.

Speed profilers. For speed monitoring, one of the difficulties is to fusion important information such as the number of times a message is sent [?] with the real execution speed [?, ?]. Indeed one is often collected on a call-site basis while the second is one based on stack sampling, therefore there is one-to-one mapping. This has an impact on the comparison between the two forms of information.

We will work on an important aspect that tools should offer: *multiple connected layers of abstractions* — a developer should be able to profile high-level code and get reports at the level of the used language. However, such results should also be exposed at a lower level of abstractions such as bytecode level for compiler designer or even further down at the level of IR (intermediate representation) or assembly for developers working on JIT. Tools often exist but they are disconnected forcing developers to make ad-hoc bridges with potential mismatches and gaps when switching levels. Virtual Machine benchmarking gets impacted by the crossing and transformations done during such abstraction traversal.

Memory profilers. In languages with managed memory, the memory consumption, *i.e.,* the creation of newly allocated objects, is transparent for developers [?, ?]. Such transparency is a problem because developers do not get proper feedback. There is no visible difference between a method creating 100 objects and a method creating and reusing one. It may be the case that 100 objects are needed or it can be simply plain wrong. We want to provide feedback to the developers in their default tools and, for example, annotate method ASTs with memory consumption or the number of created objects. The garbage collector (GC) stress and fluctuations induced by the GC are also important information that it is not easy to report and monitor between multiple versions.

Evolution monitoring. In addition, we will explore how to measure and report speed/memory changes over versions [?]. There are several difficult points to explore: (1) how to compare multiple version executions [?], because we cannot just compare the speed/memory spent in a single method because calls can be nested or moved around. It means that a drop of time spent in a method may be due to fewer invocations or cache impact, (2) building benchmark environments is difficult in particular when we are talking about managed runtime with peak performance warm-up phase [?]. Such profilers also require to rethink the notion of benchmarks. Dacapo Benchmarks [?] are an initial step in that direction. Our first experiment convinced us that more efforts are needed.

3.5.3 Revisiting reverse engineering: a dynamic perspective

Developers are daily reading code that they do not know. Still, there is really limited support to help to understand and reverse engineer knowledge [?]. This topic is partly related to the non-functional treatment described in previous sections and also the profiling.

Leverage dynamic information. In a dynamically-typed language, it is difficult to statically identify methods really invoked during execution, many false positives can appear. Tools should take advantage of previous executions, test availability, and also VM runtime internal information exposition such as PICs (polymorphic inline caches). Refactorings, system navigation can take advantage of such information. We will design and measure new navigation and reverse engineering browsers.

Presenting hidden aspects. In managed runtimes, concurrency and memory consumption are information that is not exposed to the developer. The developer has to read between the lines and know that a given method may be involved in a critical section to take such important information into account. We will work on ways to present execution-related information. A particularly challenging scenario is the design (or refactoring) of a thread-safe collection hierarchy.

"Smarter" navigation. Reverse engineering can be supported not only by having more information but also by exposing and navigating such information in better ways [?]. The window or tab plague is not solved in modern IDEs. The developer tends to open too many tabs and only uses a couple of them. We will revisit the Autumn Leaves concept to automatically discard hidden/unused/obsolete tabs and keep relevant information [?]. We will enhance the concept with domain semantic elements to guide reverse engineering.

3.6 Axis 3 – A generative approach to modular and versatile virtual machines

Members: G. Polito, S. Ducasse and P. Tesone (Pharo Consortium) — on some subtopics (tests): A. Etien

Virtual machines are key assets both from an engineering and research point of view. As extremely complex pieces of software (advanced virtual machines include a garbage collector, multi-layered interpreters, and a speculative inliner), they raise the question of their definition, construction, and validation. As a research vehicule, they are keys for innovation at the level of language design, security, ever-running systems, or green computing. This axis is based on the work the team did together with the Pharo consortium and the support of the Alamvic Inria Exploratory Action led by G. Polito.

Main objective. EVREF will explore how Virtual Machines are designed as a **whole**, and how they are optimized for a *large range of concerns* that include not only execution speed but also energy and space consumption, for applicability in *security, green computing, IoT and robotics*. Such research effort will take place in the context of the Pharo virtual machine and its associated production chain.

- **A transpilation chain.** Based on our current architecture, we will design a transpilation chain that will take into account heuristics (memory, concurrency, chipset, speed).
- **Metamodeling and DSL for VM building.** VM optimizations are complex and spread over many aspects of the logic, we will evaluate how such optimizations can be represented and extracted to be recomposed using a domain-specific language.
- **JIT compilers and optimizers.** Building modern Just in time compilers and native dynamic optimizer is a difficult task but key to support modern language execution, we want to assess the design and architecture of alternative dynamic optimizer.
- **New evaluation methodologies.** A VM is a complex piece of software with adaptative behavior we will work on ways to measure performance to be able to gather actionable information.

First period PhDs: During the first period of the team we will propose the following PhDs: *"Virtual machine transpilation chain"* and *"Revisiting speculative inlining architecture"*.

3.6.1 A transpilation chain to produce a VM

Based on the current rudimentary but working transpilation chain of the Pharo open-source VM, we will create a *VM compilation toolchain* where a VM is generated from a high-level language specification plus optimization hints. Imagine for example constructing a Javascript VM for an IoT device. Our application has no specific speed considerations nor energy consumption restrictions because it's not on battery. In such a case, our compilation toolchain takes a specification of the Javascript language accompanied by the disk and memory restrictions of the device and produces a dedicated VM for such constraints.
The compilation chain will be based on program transformations (DSL AST to C-AST, C-AST to optimized/modified C-AST, C-AST to C). A particular focus will be put on the clear identification of phases, the injection and composition of heuristics (hints such as speed optimization, memory consumption, concurrency model), and the early feedback to the VM designer. Indeed transpiling implies that the designers are using a different language than the one executed and it is challenging to report errors from the base language to their source in the high-level language.
We will model heuristics to the as *optimization heuristics*. In contrast with existing approaches that aim at reusing VM components, we aim at *reusing knowledge* encoded as optimization heuristics. Ideally, heuristics are language independent and thus applicable to different programming languages. There is a definitive link with the GWT to Angular transpilation approach with BL.

3.6.2 Metamodeling and DSL for VM building

VM optimizations are complex and spread over many aspects of the logic, we will evaluate how such optimizations can be represented and extracted to be recomposed using a domain-specific language.

Optimization heuristic sxtraction. We will extract optimizations from existing Virtual Machine components and express them as heuristics that are then recombinable to produce a working VM. Currently, language-specific Virtual Machine optimizations are encoded and diluted in the Virtual Machine source code. Such optimizations are encoded by programming language designers and Virtual Machine experts, and include for example the separation of slow execution paths from fast execution paths [?, ?], determining the compilation scope of the JIT compiler [?] and the modeling of the life cycle of memory [?]. Initially, we

will cover heuristics about execution speed as they are the best known and used, to focus on how they are expressed. A second step involves finding new heuristics about space and energy consumption.

Domain-specific language for virtual machine specification. We will define one (or more) domain-specific language (DSL) to specify Virtual Machines. Virtual Machines will be expressed in an optimization agnostic fashion. A translator will take a specification and a set of heuristics and generate a Virtual Machine that maximizes the outcome for the given heuristics, similarly to how a Meta Object Protocol or open implementation behave [?, ?, ?, ?]. This objective is particularly counterintuitive and controversial, as Virtual Machine experts usually want to have full control over the Virtual Machine behavior, and we propose a radically opposite optimization agnostic approach, where optimizations are introduced as crosscutting concerns. This will be based on the transpiling toolchain mentioned above.

Combining heuristics for emergent fields (fusion with above). We will define combinations of heuristics to make VMs practical in emergent fields such as IoT and robotics. This task involves studying how optimization heuristics combine and conflict with each other (e.g., it is common that modularity has a cost in performance [?]), define a formal framework for their combination such as the one by Click [?], and establish a first set of heuristics that are applicable in the named fields.

3.6.3 JIT compilers and dynamic optimizers

Modern virtual machines such as V8 are often built around multiple tiers: an optimized interpreter (Ignition in the case of V8 with TurboFan), a baseline JIT whose purpose is to natively compile methods without execution profiles, and a speculative optimizer that performs more advanced optimizations based on program execution [?]. The overall design of such important components is often at the cost of duplicating logic and internal representation.

Architecture and modeling of dynamic optimizations. The internals of a JIT, like any modern compiler, is based on intermediate representations (IR). Such IRs are used to represent control flow graphs and instructions. Many different designs of compiler IRs exist both in the literature and in industrial Virtual Machines: low vs. high level, linear vs. tree vs. DAG IRs, with stack vs. register machines, infinite vs. fixed registers. The choice of an internal representation impacts the possible optimizations it supports and their definitions. For example, new approaches like *sea of nodes* mix control flow and instructions at the same level of abstraction and efficiently encode instruction dependencies, allowing for better instruction scheduling and simpler implementations of optimizations. We will use the infrastructure developed in Section **??** to evaluate such architectural choices.

Another important aspect of this architecture is the different tiers and their interactions. For example, during RMOD C. Béra proposed in his PhD [?] (based on the idea of E. Miranda who engineered 5 industrial virtual machines) that the speculative optimizer (Sista) would work on producing unsafe bytecodes [?, ?] and reuse the first tier JIT as a native code generator. Sista is in essence a reflective JIT compiler since it can modify the optimization logic of program execution. We will reassess Sista's architecture. In Sista, the optimizer transforms unoptimized stack-based linear bytecode into a high-level infinite register-based linear IR with basic blocks, applies transformations on top of this IR, and then transforms the optimized code back to stack-based linear bytecode. This also means that much of the information Sista obtains during the optimization phase (*e.g.,* register calculations, control flow structure), that could be beneficial to the code generation tier is discarded. We want to compare such an approach with more traditional approaches such as the one of modern Javascript or Java Virtual. When comparing approaches, an important factor is often neglected: this factor is the development cost.

Compiler and memory manager cross-benefits. Modern garbage collection schemes are, as well as modern JIT compilers, also based on runtime statistics and optimization heuristics [?]. We will explore how memory management and adaptive optimizations can have a positive symbiotic impact on each other. Several systems have explored the idea of using adaptive compilation to remove pressure on the memory manager, and vice-versa using memory management statistics to drive compiler decisions. Escape analysis allows one to inline object state in the execution stack to avoid heap allocations [?, ?, ?]. Dynamic object-allocation strategies adapt object storage on a per-object granularity based on their runtime usages [?]. Pre-tenuring optimizes object allocation in generational garbage collections based on previous observations of object life-cycles [?].

Domain-specific learning/profiling/optimising. Dynamic optimizers are inherently [?] auto-adaptative systems. We will explore the mechanisms to support dedicated object layout [?, ?]. This implies understanding what aspects to profile and how to convert such profiling information into tractable transformations at the object layout and memory organization levels.

Free MOPs. Reflective operations are costly when they are used. In addition, when implemented naively, they are costly when they are not used [?]. We want to revisit the ideas of zero-cost reflective operations with a traditional JIT compiler. In particular, we want to see how we propose the possibility to design dedicated MOPs and support also reflective virtual machine [?] only paying for the used features.

Development cost. Strangely enough, it is accepted that building a virtual machine incurs large teams of dedicated developers as this is the case in Google around V8, Oracle around the JVM. We are interested in measuring the engineering cost of developing and validating dynamic optimizations [?, ?]. The hypothesis behind the infrastructure we will develop as mentioned in Section **??** is that level of abstractions should support developers in the development of virtual machines. Jikes [?] or VMKit [?] wanted to offer a platform for the exploration of VMs. We share the goal with such now terminated projects. In addition, the tooling as mentioned in Sections **??** and **??** plays also an important place. The moldability [?] of the tools is important and letting developers the possibility to develop their own tooling is a key aspect of productivity.

3.6.4 New evaluation methodologies

Finally, an important objective, although cross-cutting, is to define new methodologies and metrics to evaluate VMs as they exist today to evaluate execution speed [?, ?, ?]. Besides execution speed, the JVM Spec benchmarks measure different qualities of a program [?] but they present micro benchmarks that have been argued not representative of real applications. This point is related to the challenge of multi-profilers exposed in the second axis and the challenges of monitoring reliably execution information.

To add to the previous point, but in a pervasive manner through this axis, there is an important challenge both from a engineering and research point of view that we are starting to address: how to (1) be *able to change* an aspect in a VM (GC, object representation, optimisation strategies, internal compiler representation (*e.g.,* sea of nodes, basic blocks)), and (2) *assess* the impact of a change. We were among the first researchers proposing the validation of JIT dynamic optimisations [?]. With the help of the Pharo consortium and the Alamvic project we are investigating different approaches to test Virtual Machines. We are currently evaluating differential testing using concolic testing [?, ?, ?] in the context of compiler testing [?].

3.6.5 Enabling research venues

Besides the four topics identified above, our work on Virtual Machines will fuel research on other topics:

- **Ever-running systems**. We want to explore how we can execute multiple versions of the same language and its runtime. In Erlang [?], two different versions of a module can be active at the same time. When code is loaded, the runtime retains both the old and new versions. Calling conventions define which version is called. This allows a module to continue executing old code until it is restarted. If a third version is loaded, all processes executing the oldest code are killed. Erlang focuses on providing a robust model for dynamic code loading.

  Modifying a running application requires adapting the structure of objects to a new class version. Multiple versions of the same class can coexist and instances of these different versions should migrate [?]. There has been work on schema evolution for OODS such as O2 [?], GemStone [?] or Objectivity/DB, Versant [?]. Gemstone supports the management of multiple versions of the same class with transparent migration.
- **Dynamic software update.** Without requiring a specific VM as in [?], Rubah [?] is one of few systems which supports dynamic software updates of Java code. However, it does not support core library updates. We designed Espell, an infrastructure that supports dynamic software updates even of core libraries [?, ?, ?]. We may revisit such a domain.
- **VM API for tool builders.** While existing information extracted by systems like JVMTI is rich, it does not extract alias information. In addition, we should control the influence of probes on the program behavior, this is crucial for concurrency issues. We will work on ways to support the users to select which information should be extracted, at which point in time, and at which scope. In particular, we will work on support for back-in-time debuggers.
- **Low-level abstractions for security.** Process isolation is an old technique for providing more secure systems, pioneered by operating systems: a running program in an OS process cannot reference the memory of another process. Several proposals for memory isolation at language level exist including the Java specification for *isolates* [?], MVM [?] (which pioneered the isolate abstraction), KaffeOS, JX, and O-JVM. Microsoft developed an operating system called Singularity [?]. In Singularity, Software Isolated Processes (SIP) are holders of processing resources. We want to explore how we can support different levels of isolation: From

isolated spaces to hardware supported security. Several systems support side-by-side applications. But the interoperability between such systems is often not possible.

- **Support for green computing.** Building systems going faster and faster is one goal but we would also like to explore how we can build more eco-friendly execution engines and tools providing more adequate information to the developer.

4 Application domains

EVREF does not have specific application domains *per se*. In the past, we worked on missile sending devices, industrial robots, information systems, insurance, banking and open-source software. We will work on open-source and proprietary software to help companies with which we interact. We usually work on large and old systems.
Nevertheless, we see several areas of application domains.

- Understanding and maintaining AI-based software systems. There will be more and more software systems using AI and their evolution and maintenance will have to take into account their particularity.
- Frugal or green virtual machines. Most virtual machines are optimized for execution speed, still we would like to explore frugal VM.
- More secure runtime. From a security point of view, we would like to see how Virtual machines can take advantage of dedicated hardware.

There are also problems we do not plan to address:

- Our software analyses are not good for pointer analysis or template-based programming and we do not intent to work there.
- We do not plan to work on fine-grained execution analysis such as maximal stack consumption because this is highly dependent of the language analyzed and we are focusing on the evolution of large systems.
- We do not plan to work on concurrency or race condition analysis, even if supporting debugging concurrent program is a topic on which we are planning to work in another iteration of the team.

5 Software

Software plays a central role in EVREF. First as input but also as produced artefact. However the production of software is not a goal by itself, it is a means to get concrete feedback on real data (often large software systems). To reduce the time required to be in a situation to design and evaluate a new approach, the team built over the years two important platforms: Moose and Pharo. In addition, such platforms are the substrate of team member collaboration. Often Pharo engineers are getting feedback from Moose developers and this is an important feedback loop.

Moose (http://www.moosetechnology.org) is a data analysis and software platform. It is composed of several generic and scriptable engines (visualization, interface construction, modular parser-combiner, ...). Moose is used by several research groups and some companies such as Berger-Levrault. Moose has been used for more than 300 research papers including master's degrees http://scg.unibe.ch/scgbib?_k=phspe-mp&query=moose. We are currently revisiting Moose to make it more modular. Moose is under BSD license.

For the Moose platform, Clotilde Toullec is the engineer helping to develop it and another engineer will reinforce the effort.

Pharo (http://www.pharo.org) is a dynamically-typed object language, development environment, and a virtual machine. It runs on the following platforms: mac, linux, windows, android, and iOS on Intel32/64,ARM32/64. Pharo is composed of several frameworks (compiler tool chain, IDE, Networking libraries, graphic libraries . . .). Pharo is supported by an industrial consortium: http://consortium.pharo.org. The consortium is now composed of 28 industrial members, 24 academic members. The consortium pays two full-time expert engineers. Core development is supported by 3 or 4 engineers. Pharo includes an industrial-strength virtual machine running on Mac, Windows, Raspberry, in 32/64bits, and Intel/ARM. Pharo is under MIT license.

Pharo9.0 :

- is composed of 730 packages for 9,000 classes and 120,000 methods,

- has around 230 forks on http://www.github.com/pharo-project/pharo,
- more than 30 universities are using it to teach in the world.

For Pharo, the consortium is a strong help. We will help the consortium growing to able to hire a third engineer.

Software as a central asset. EVREF will continue to develop the two main software platforms Moose and Pharo produced by RMOD. Their development is a large investment but it is needed to be able to realize our scientific roadmap and to create a feedback loop from the practice to fuel future research ideas.
Each of our research effort, being it a PhD or not, is based on developing some sort of software. Most of the time, the development effort can be consequent but it is needed to be in a position to validate research ideas. The use of our platform reduces the time to develop and avoids the reimplementation of the lower layers helping to focus on the new and valuable part. Our process is then to assess whether there is a value in the developed artefact, if it can be reused, extended and be the basis for further research. When this is the case, we go over an extra effort to consolidate it and integrate it in our platform. Such effort can be consequent and may require the help of a platform engineer.
Most of the development made in the context of the first research axis (software evolution) will be integrated into the Moose platform. The development made on the second axis (new generation tools) will happen and transferred to Pharo when mature enough. The work of the third axis (Virtual Machines) will take its root from VMMaker, the current Virtual Machine transpilation chain of the open-smalltalk VM that the Pharo VM inherits from. New development will take place in this context. We started to work on a new version and will create step by step a new transpilation chain as well as a new virtual machine family to be able to experiment new approaches. Moose being developed on top of Pharo, it will also indirectly benefit from work of the second and third axis.

6 Transfer

RMOD the previous team produced two startups (Synectique and Codaxis), two startup projects (Cells, Pharo-pro), created an industrial consortium, several CIFREs, and bilateral contracts. For EVREF, we plan the same because the transfer is really important for the team. The creation of the team with Berger-Levrault is already a large effort of transfert.
We will

- Continue growing the open-source communities around Moose and Pharo.
- Work with companies via direct contracting and bilateral contracting such as CIFRE PhD program.
- Work with Berger-Levrault.
- Interact with companies via the Pharo consortium and continue growing the Pharo consortium. It is important to consider that the Pharo consortium is a kind of separate team managed by S. Ducasse and living symbiotically with EVREF. So it is another layer of work. The consortium is in a post bootstrap phase where it is in need of more engineering power and at the same time would benefit from a dedicated person working full time on promoting Pharo and meeting Pharo customers, but the need for engineering power makes that we focus on the engineer effort.
- We would like to support another startup, but for that we would need people with time and will.

7 Positioning and collaborations

EVREF is clearly positioning itself in the scientific challenge: "Eternal software systems" of the Inria strategic plan 2018-2022. In France, there is no team working on the challenges we identified, the closest one are Diverse and the software engineering group of the Labri. Diverse has a focus on devops and variability and the software engineering group of the Labri shares some interest in software quality and analyses but also focuses on distributed systems.

7.1 Positioning within Inria

There is no team tackling software evolution and tooling/infrastructure the same way as we propose. Still related the following teams share some experience with EVREF research efforts:

- The approach of DIVERSE is completely different from the one of EVREF. The management of the diversity, through variability is the leading idea of the DIVERSE team. In particular, they focus on how to automatically compose and synthesize software diversity from design to runtime. Evolution, language and tools are somehow a side effect. In EVREF, evolution, language and tools are the fundamental focus of the team. Nevertheless, we join each other on some topics like Modeling and Languages Engineering and advanced testing that we tackle differently. Moreover, if they use VM, or debuggers for example, they do not focus on these topics.
- The Whisper Inria team is working on patch identification and applications (via the Coccinelle tool). In addition, Whisper focuses on infrastructure such as resources access, hypervisors, managed runtime.... Whisper, similarly to EVREF is at the interface operating systems, software engineering and programming languages — EVREF focusing on dynamically-typed languages and deploying Pharo.
- Indes Inria team work on language security for languages in the cloud. Indes has knowledge on compilation, JIT and dynamically-typed languages like EVREF. We want to see how virtual machines design and implementation can be made "easier" or at least more accessible.

7.2 Positioning within cristal

None of the Cristal teams is working on software evolution per se and the tooling/infrastructure aspect EVREF takes.

- Sycomore, a new Inria team of Cristal, is focusing on real-time scheduling analysis. EVREF has some expertise but limited to concerns around virtual machine thread scheduling.
- 2XS is working on detecting intrusion in the Internet or radio protocols.
- EVREF (mostly axis 1) and Caramel are manipulating models and metamodels. However, the purposes are different. Caramel develops theory on models to compose and reuse them. EVREF massively uses models and metamodels to manage software evolution.
- Carbon is focusing on modeling aspects and its links with users. Once again, the links between the two teams rely on the massively use of models and metamodels but for different purposes. Moreover, thematics and challenges around axes 2 and 3 are not considered neither by Caramel nor Carbon.
- Spirals (Inria team) is working on security aspects (such as browser fingerprints), cloud and formalisation of components. Objectives of Spirals are completely different from those of EVREF. Nevertheless, in some contexts Spirals members may have to manage evolution or maintenance but it is at the margins. In the opposite, EVREF challenges may tackle security but it is just an example of non-functional requirements and not a target per se.

7.3 Positioning at international

Research groups on software evolution. There is a number of research groups on reengineering. Here a list of the most important ones, even if they are not exactly working on software evolution and migration.

- SERG: The Software Engineering Research Group https://se.ewi.tudelft.nl is working on software evolution. They do not really have a focus on tools and do not look at infrastructure of languages.
- The Seal group from the University of Zurich is focusing on technologies related to the development of large, complex, and long-living software systems. For that purpose, software development methodologies and paradigms are needed to provide evolvability and maintainability characteristics of software. Recently they proposed a runtime architecture-based approach for the dynamic evolution of distributed component-based systems.
- M. Lanza and his group at University of Lugano is focusing on software visualization.
- LORE (Lab On REengineering) of the University of Antwerp focuses mostly on clones and tests (mutation).
- Geodes (headed by H. Sahraoui) and his team of the University of Montreal are working on search-based techniques for Software engineering (multiobjectives search applied to refactorings) but also in the past quality models and visualizations.
- Y.-G. Guéhéneuc of the University of Concordia is focusing on the identification of design patterns and lack of design quality. We want to focus on larger abstractions and not only focus on design patterns. We are in contact in the context of the SadPC associated team.
- J. Maletic of the Kent State University is working on templates migration and other code analysis to support the understanding and evolution of application. We are evaluating the use of the SrcML tools to

parse C/C++ code.

- In France, the team Génie Logiciel of Labri led by J.-R. Falleri is working on software evolution and software quality. Concerning software evolution and quality, they mostly adopt an empirical software engineering approach.

Research groups on tools. We do not know any research groups on debuggers in France. Microsoft Research is currently working on back-in-time debuggers such as Tardis, and on REPT (2018) and Kernel-REPT (2020), reverse debuggers for software and kernel failures in deployed systems. They ship REPT with the Windows operating system and stress the importance of reverse and back-in-time debuggers for investigating failures in real-life deployed systems. The only team in europe we know are:

- The team of A. Zeller is expert on tests such as test carving and delta debugging techniques.
- The team of E. Gonzalez-Boix from VUB is working on debugger for concurrent programming systems as well as map/reduce applications. We collaborate with them.

Research groups on virtual machines. There is a limited number of research groups on virtual machines.

- L. Tratt has been working on Virtual Machines (mainly benchmarking) at King's College. The team is now working on a Tracing JIT for RUST.
- S. Marr at Kent University is working on Virtual Machines with a focus on concurrency issues. He is using simple VM (SOM) and collaborate with the Truffle group. He is also working on benchmarking.
- The software system group led by H.P. Mossenbock of Kepler Universität Linz is using the Graal Java VM (mainly Oracle) to rewrite program ASTs at runtime and invoke the underlying JIT.
- In France, G. Thomas worked on Garbage collection. Parkas's Inria team worked on compiler optimization such as loop unrolling. M. Serrano is working on ahead of time compilation. But this is not really Virtual Machines as a whole.

In EVREF (and in collaboration with the Pharo consortium), we are building a full VM from top to bottom: it comprises GC and memory management, FFI, object representation, optimizing compilers (JIT). We hope to restart to work on dynamic optimization.

7.4 Collaborations

VUB – SOFT — E. Boix Gonzalez and Coen De Roover. We collaborate since several years with the Soft team (previously PROG) of the Vrije Universiteit Brussels. We got a large number of exchanges between our two teams. G. Polito co-supervise the PhD of Matteo Mara with E. Boix Gonzales on debugging map reduce applications. The COVID blocked our recent effort to launch a new period.

Université du Chili à Santiago — A. Bergel. 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.

Université de Chicoutimi au Québec — F. Petrillo. We started to 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.

Thales DMS, Brest, France — E. Le Pors. We collaborate with Dr. E. Le Pors, leader of the Thales DMS software prototyping team. His team builds complex industrial systems, source of hard bugs that we study for the design and the empirical evaluation of new debuggers.

Montreal. C. Furhman ETS - Y.-G. Guéhéneuc / Columbia. We started to collaborate with several groups in Montreal and the SADPC associated team should have started in 2020 but was postponed due to COVID.

NoviSad. G. Rakic and G. Milosavljevic. We started to collaborate with two groups of NoviSad University. We hope post COVID will let us restart our efforts.

Working on new collaborations. Prof. J.P. Sandoval is taking a new position in Chile. We plan to continue working with his team on Profilers.

7.5 National networks

- Steven Costiou is co-responsible of the working group of the GDR GPL on debugging.

- Anne Etien is co-responsible of the working group of the GDR GPL on Software Engineering and Artificial Intelligence.

A CVs

N. Anquetil (HDR). He completed my PhD in 1996 at University of Montréal. Since then, he worked successively at University of Ottawa (Canada), Federal University of Rio de Janeiro (Brazil), Catholic University of Brasilia (Brazil), and Ecole des Mines de Nantes (France). I am now Associate Professor (MCF / HDR) at University of Lille. He co-founded Synectique: a company that was selling software analysis tools. He is now working on the new generation Moose software analysis platform.

S. Costiou. He is a permanent Inria researcher (CRCN). Before that, he worked six years in the industry as a software developer in various areas (defense, aerospace, point-of-sale software, etc.). I then did research on unanticipated software adaptation during my PhD at Université de Bretagne Occidentale (France), before becoming a permanent researcher at Inria Lille - Nord Europe. Today, he is working on finding new ways of debugging. He is interested in the identification and the study of the properties that programming languages and their infrastructure (i.e., virtual machines) must exhibit to support new debugging features that effectively help debugging. This research spans different topics: reflection and meta-programming, object-centric instrumentation, dynamic software adaptation, dynamic languages, and virtual machines.

M. Denker. He is a permanent researcher (CRCN) at Inria Lille - Nord Europe. His research focuses on reflection and meta-programming for dynamic languages. He is an active participant in the Pharo open source community and a Pharo core developer. Before joining Inria, he was a postdoc at the PLEIAD lab/DCC University of Chile and the Software Composition Group, University of Bern. Marcus Denker received a Ph.D. in Computer Science from the University of Bern/Switzerland in 2008 and a Dipl.-Inform. (MSc) from the University of Karlsruhe/Germany in 2004. He co-founded ZWEIDENKER GmbH (Cologne, Germany) in 2009. He is a member of ACM, GI, and a board member of ESUG.

C. Demarey. He is an Inria engineer. He worked on Virtual Machines for embedded devices. He was responsible for the CI inria infrastructure. With RMOD he worked on supporting the bootstrap, package management and the Pharo Launcher. Christophe participated to the development of TousAnticovid, the national french COVID application of the french government.

A. Etien. She is currently full Professor at the University of Lille. She is doing research in the RMOD team since 2012 on software maintenance of large legacy software systems and more specifically on software quality, tests, software cartography... She regularly works with companies to answer their requirements around these problematics. Before she worked on model-driven engineering and more specifically on model transformations. And before that, she worked during her PhD at the University Paris 1 on the evolution of Information systems and requirement engineering. Finally, she published around hundred research papers and supervised eight PhD students.

G. Polito (Ph.D.). He is currently CNRS research engineer. Guillermo holds a PhD from the University of Lille. He did a postdoc at the Free University of Brussels where he is supervising Matteo Marra with Prof. Gonzales-Boix. He co-supervised three PhD with Prof. Fabresse and Bouraqadi from IMT Douai. His thesis results (a bootstrapping architecture and applications) are used daily to produce Pharo. He is expert in language design, compiler, and leading the effort around Virtual Machine: the emergent field of EVREF.

C. Bortolaso (Ph.D.). He is the head of research programs at Berger-Levrault. He holds a Ph.D. in Computer Science focused on human computer interaction. He has contributed for more than 10 years to numerous research and software development projects in France and Canada, in various industrial sectors such as defense, culture, energy, public services, and healthcare. Today, in charge of research programs at Berger-Levrault, he coordinates a team of researchers in multiple domains ranging from software engineering, human-computer interaction, artificial intelligence, and natural language processing.

A. Seriai (Ph.D.). He is in charge of software engineering research programs at Berger-Levrault. Abderrahmane Seriai holds a Ph.D. in computer science from the University of Montreal (Canada) and University of Bretagne Sud since 2015. His doctoral studies focused on the problem of migrating from object-oriented to component-oriented applications. He worked at Orange-Labs and CEA as a research and development engineer in the fields of software engineering and embedded systems security. He joined Berger-Levrault in 2017 to focus on applied research in the field of software engineering. He actively participates in the design, analysis, and development of use cases for new technologies (migration, software architecture, security technologies (migration, software architecture, blockchain, product line, AI for software engineering, etc.).

Stéphane Ducasse

Senior Researcher (Directeur de recherche de première classe) 53 years old
Inria Lille Nord Europe - Cristal laboratory University of Lille
mailto:stephane.ducasse@inria.fr
http://stephane.ducasse.free.fr

Keywords: Dynamically typed languages, Language Design and Security, Software Engineering, Object-Oriented Programming, Reflective Programming, Meta-Object Protocol, Meta-Modeling, Reengineering, Reverse Engineering, Program Understanding, Integrated Development Environments, Teaching Novices.

Indexes: According to Google Scholar, *H-index:* 58 for 15000 citations.

Education and Titles

May 2002	PrivatDozent of the University of Bern.
Sep 2001	Habilitation à diriger des recherches of Université Pierre et Marie Curie (Paris 6).
1993/1996	Ph.D. Thesis of the Université de Nice-Sophia Antipolis (Laboratoire I3S).

Professional Employment History

2013	Co-Founder of Synectique (spinoff).
2012	Promoted **First Class** Inria Research Director (only a 1/3 of directors are first class).
2011 – 14	**Scientific Deputy Director** of Inria Lille-Nord Europe Research Lab.
2007 – ...	**Inria Research Director** (equiv. to Full Professor). Founder and leader of the RMOD Inria team: 6 permanent researchers, 2 postdocs, 4 engineers, 7 Ph.Ds.
2005 – 07	**Full Professor** of Université de Savoie, Leader of the Software Engineering team (5 permanent members, 3 Ph.Ds, 1 postdoc).
2002 – 05	**Swiss National Foundation Professor** at University of Bern (20% accept). Leader of RECAST project and jointly in charge of the Software Composition Group directed by Prof. O. Nierstrasz.
1996 – 02	**Lecturer** at the University of Bern in the Software Composition Group.

Awards

2012	Inria Prime d'excellence scientifique - Scientific Excellence Award.
2011	Distinguished Visiting Fellowship Award of the Royal Academy of Engineering.
2010	ESUG 2010 best Smalltalk book for *Dynamic Web Development with Seaside*, 2010.
2008	Best award paper of IEEE Working Conference on Reverse Engineering 2008.
2003	Best award paper of Joint Modular Language Conference 2003.

Funding ID

2021. Thales 55 K / Siemens 110 Keuros
2019-2020. Lifeware 220 Keuros / Arolla 320 Keuros
2018-2019. CIM 160 Keuros
2017-2018. Utocat 160 Keuros / Berger-Levrault 320 Keuros
2015-2018. Thales 45 KEuros + 120 Keuros - WordLine 45 KEuros + 120 Keuros
2012-2015. SafePython - FUI Systematic - 120 Keuros
2011-2014. Resilience project - Security in Javascript - FUI Systematic - 240 Keuros
2010-2014. ANR Project Cutter - Reengineering (LIRMM + INRIA) - 150 Keuros (one Ph.D.)
2008-2010. Squale project - Reengineering (AirFrance, Peugeot, Qualixo) - 260 Keuros (1 Ph.D. 1 postdoc)
2005-2008. ANR Project Recast - Reengineering - 130 Keuros (20% acceptance)
2002-2006. Swiss National Science Foundation 2002 Professorship - 500 Keuros (18% acceptance)

Publications

54 Internat. Journals	3 Trans. of Softw. Eng. (TSE), 1 Trans. of Progr. Lang. and Syst. (TOPLAS), 7 J. of Computer Lang., Syst. and Struc., 3 J. of Soft. Maint. and Evol. (JSME), 4 Infor. and Soft. Tech. (IST), SOSYM, IEEE Software, etc
104 Internat. Conferences	9 OOPSLA (18%), 3 ECOOP (16 %), 1 PLDI (16%), 1 FSE (16%), 1 ASE (9%), 8 ICSM (21%-35%), 9 WCRE (25%-35%), 2 UML(25%), 6 TOOLS (25%)
	4 Invited Papers, 3 Book Chapters, 3 National Journals, 11 Books

Professional Activities

8 Habilitation evaluation, 26 Ph.D. thesis evaluation
28 Ph.D. thesis supervisions - defended
6 International Conference General Chair: ESUG International Conference on Smalltalk (05, 06, 09, 10 - 150 participants), WCRE 2009 (90 ppts), LMO 2005 (60 ppts).
International Journal Steering Board: open-archive Journal of Object Technology (http://www.jot.fm)
40 International Conference Program Comittee Participation
ECOOP Core A*, Models Core B, ICSM Core A, WCRE Core B, TOOLS Core B, CSMR Core B.
80 International Conference reviewer for OOPSLA, ECOOP, FSE/ESEC, ICSM, MODELS...
International Journal reviewer: Transaction on Software Engineering (Core A*), Journal of Software Maintenance and Evolution: Research and Practice, Journal of Software and Systems Modeling, Information and Software Technology, Journal on Computer Programming (Core A),...
Research Network Building: European Science Foundation Network: "RELEASE: Research Links to Explore and Advance Software Evolution" (02-05).

Keynotes and Invited Courses

Conference Keynotes	Modularity 2017, Software Language Engineering 2015, Software Composition 2009, Smalltalks 2009
Invited Papers	with O. Nierstrasz, Software Composition 2005, ESEC/FSE'05, Generative Programming and Component Engineering, 2005
Invited Lectures	University of Skikda, Algeria (3 days), ENSI Tunisia (3 days), Univ. Novi Sad, Serbia (3 days), 2018
	University of Cagliari (3 days), 2017
	University of Lviv (7 days), 2016
	University of Prague (5 days), 2016
	Universitat Politècnica de Catalunya (5 days), 2010
	Universidad de Buenos Aires (3 days), 2009
	Università di Torino (5 days), 2007, 2008
	Università di Torino Timisoara (5 days), 2002
Invited Seminars	University of Skikda, Univ. of Novi Sad, Serbia 2018
	University of Turino, 2017

Platforms

Pharo. http://www.pharo.org - Co-founder / industrial consortium leader
Moose. http://moosetechnology.org - Co-founder / maintainer

Scientific Reviewer

I act as reviewer for IST (Information Society Technology), SSF (Stiftelsen för Strategisk Forskning Sweden), EPSRC (UK), for the Fonds de Recherche sur la Nature et les Technologies of Quebec – Canada, NWO (Dutch National Research Council), NSERC (Natural Sciences and Engineering Research Council of Canada), and ANR (Agence Nationale de la Recherche), IWT-Flanders (Belgium), and SSF (Swedish Science Foundation).

References

Prof. O. Nierstrasz, University of Bern, oscar@iam.unibe.ch
Prof. A. Black Portland State University black@cs.pdx.edu
Directeur de recherche CNRS **J.-P. Briot**, Laboratoire d'informatique de Paris 6, jean-pierre.briot@lip6.fr
Prof. T. D'Hondt, Head of the PROG Laboratory of the University of Brussels tjdhondt@vub.ac.be
Prof. J.-M. Jezequel, IRISA - Université de Rennes, jezequel@irisa.fr