RESEARCH CENTRE

**Rennes - Bretagne Atlantique**

**IN PARTNERSHIP WITH:**

**CNRS, Institut national des sciences appliquées de Rennes, Université Rennes 1**

# 2020 ACTIVITY REPORT

# Project-Team DIVERSE

## Diversity-centric Software Engineering

**IN COLLABORATION WITH: Institut de recherche en informatique et systèmes aléatoires (IRISA)**

**DOMAIN**

**Networks, Systems and Services, Distributed Computing**

**THEME**

**Distributed programming and Software engineering**

# Contents

# Project-Team DIVERSE

*Creation of the Team: 2014 January 01, updated into Project-Team: 2014 July 01*

## Keywords

### Computer sciences and digital sciences

A1.2.1. – Dynamic reconfiguration

A1.3.1. – Web

A1.3.6. – Fog, Edge

A2.1.3. – Object-oriented programming

A2.1.10. – Domain-specific languages

A2.5. – Software engineering

A2.5.1. – Software Architecture & Design

A2.5.2. – Component-based Design

A2.5.3. – Empirical Software Engineering

A2.5.4. – Software Maintenance & Evolution

A2.5.5. – Software testing

A2.6.2. – Middleware

A2.6.4. – Ressource management

A4.4. – Security of equipment and software

A4.8. – Privacy-enhancing technologies

### Other research topics and application domains

B3.1. – Sustainable development

B3.1.1. – Resource management

B6.1. – Software industry

B6.1.1. – Software engineering

B6.1.2. – Software evolution, maintenance

B6.4. – Internet of things

B6.5. – Information systems

B6.6. – Embedded systems

B8.1.2. – Sensor networks for smart buildings

B9.5.1. – Computer science

B9.10. – Privacy

# 1 Team members, visitors, external collaborators

## Research Scientists

- Djamel Eddine Khelladi [CNRS, Researcher]
- Olivier Zendra [Inria, Researcher, from Oct 2020]

## Faculty Members

- Olivier Barais [Team leader, Univ de Rennes I, Professor, HDR]
- Mathieu Acher [Univ de Rennes I, Associate Professor]
- Arnaud Blouin [INSA Rennes, Associate Professor, HDR]
- Johann Bourcier [Univ de Rennes I, Associate Professor, HDR]
- Stéphanie Challita [Univ de Rennes I, Associate Professor, from Sep 2020]
- Benoit Combemale [Univ de Rennes I, Professor, HDR]
- Jean-Marc Jezequel [Univ de Rennes I, Professor, HDR]
- Noel Plouzeau [Univ de Rennes I, Associate Professor]

## Post-Doctoral Fellows

- Juliana Alves Pereira [Univ de Rennes I, until Feb 2020]
- Raounak Benabidallah [Univ de Rennes I, from Sep 2020]
- Dorian Leroy [Inria, from Mar 2020]
- Xhevahire Ternava [Univ de Rennes I, from Oct 2020]
- Oscar Luis Vera Perez [Univ de Rennes I, from Jan 2020]
- Nan Zhang Messe [Univ de Rennes I, from Sep 2020]

## PhD Students

- June Benvegnu-Sallou [Univ de Rennes I]
- Anne Bumiller [Orange, CIFRE, from Oct 2020]
- Emmanuel Chebbi [Inria]
- Antoine Cheron [Zengularity SAS, CIFRE]
- Fabien Coulon [Obeo, CIFRE]
- Jean-Emile Dartois [Institut de recherche technologique B-com, until Feb 2020]
- Cassius De Oliveira Puodzius [Inria, from Oct 2020]
- Pierre Jeanjean [Inria]
- Gwendal Jouneaux [Univ de Rennes I, from Oct 2020]
- Quentin Le Dilavrec [Univ de Rennes I, from Oct 2020]
- Dorian Leroy [Université de Vienne - Autriche, until Jan 2020]

- Luc Lesoil [Univ de Rennes I]
- Gauthier Lyan [Keolis, CIFRE]
- Hugo Martin [Univ de Rennes I]
- Lamine Noureddine [Inria, from Oct 2020]
- Alif Akbar Pranata [Inria]
- Alexandre Rio [Univ de Rennes I, from Mar 2020 until Apr 2020]

### Technical Staff

- Didier Vojtisek [Inria, Engineer]

### Interns and Apprentices

- Janice Conquet [Univ de Rennes I, from Feb 2020 until Jul 2020]
- Philemon Houdaille [Univ de Rennes I, from Jun 2020 until Aug 2020]
- Gwendal Jouneaux [Inria, from Feb 2020 until Aug 2020]
- Quentin Le Dilavrec [Inria, from Feb 2020 until Jul 2020]
- Corentin Ollivier [Univ de Rennes I, from Jun 2020 until Sep 2020]
- Georges Aaron Randrianaina [Univ de Rennes I, from Jun 2020 until Aug 2020]

### Administrative Assistant

- Sophie Maupile [CNRS]

### Visiting Scientists

- Nelly Bencomo [Aston University, from January 2020 until Jun 2020]
- Gunter Mussbacher [Université de Montréal - Canada, from Mar 2020 until Jun 2020]

### External Collaborator

- Gurvan Le Guernic [DGA]

# 2 Overall objectives

## 2.1 Overall objectives

DIVERSE's research agenda targets core values of software engineering. In this fundamental domain we focus and develop models, methodologies and theories to address major challenges raised by the emergence of several forms of diversity in the design, deployment and evolution of software-intensive systems. Software diversity has emerged as an essential phenomenon in all application domains born by our industrial partners. These application domains range from complex systems brought by systems of systems (addressed in collaboration with Thales, Safran, CEA and DGA) and Instrumentation and Control (addressed with EDF) to pervasive combinations of Internet of Things and Internet of Services (addressed with TellU and Orange) and tactical information systems (addressed in collaboration with civil security). Today these systems seem to be radically all different, but we envision a strong convergence of the scientific principles that underpin their construction and validation, bringing forwards sane and reliable methods for the design of **flexible and open yet dependable systems**. Flexibility and openness are both

critical and challenging software layer properties that must deal with the following four dimensions of diversity: **diversity of languages**, used by the stakeholders involved in the construction of these systems; **diversity of features**, required by the different customers; **diversity of runtime environments**, in which software has to run and adapt; **diversity of implementations**, which are necessary for resilience by redundancy.

In this context, the central software engineering challenge consists in handling **diversity** from variability in requirements and design to heterogeneous and dynamic execution environments. In particular, this requires considering that the software system must adapt, in unpredictable yet valid ways, to changes in the requirements and environment. Conversely, explicitly handling diversity is a great opportunity to allow software to spontaneously explore alternative design solutions. Concretely, we want to provide software engineers with the following abilities:

- to characterize an "envelope" of possible variations;
- to compose envelopes (to discover new macro envelopes in an opportunistic manner);
- to dynamically synthesize software inside a given envelop.

The major scientific objective that we must achieve to provide such mechanisms for software engineering is summarized below:

**Scientific objective for DIVERSE:** To automatically **compose and synthesize software diversity** from design to runtime to **address unpredictable evolution of software-intensive systems**

Software product lines and associated variability modeling formalisms represent an essential aspect of software diversity, which we already explored in the past, and this aspect stands as a major foundation of DIVERSE's research agenda. However, DIVERSE also exploits other foundations to handle new forms of diversity: type theory and models of computation for the composition of languages; distributed algorithms and pervasive computation to handle the diversity of execution platforms; functional and qualitative randomized transformations to synthesize diversity for robust systems.

# 3 Research program

## 3.1 Scientific background

## 3.2 Model-Driven Engineering

Model-Driven Engineering (MDE) aims at reducing the accidental complexity associated with developing complex software-intensive systems (e.g., use of abstractions of the problem space rather than abstractions of the solution space) [112]. It provides DIVERSE with solid foundations to specify, analyze and reason about the different forms of diversity that occur through the development lifecycle. A primary source of accidental complexity is the wide gap between the concepts used by domain experts and the low-level abstractions provided by general-purpose programming languages [83]. MDE approaches address this problem through modeling techniques that support separation of concerns and automated generation of major system artifacts from models (*e.g.,* test cases, implementations, deployment and configuration scripts). In MDE, a model describes an aspect of a system and is typically created or derived for specific development purposes [66]. Separation of concerns is supported through the use of different modeling languages, each providing constructs based on abstractions that are specific to an aspect of a system. MDE technologies also provide support for manipulating models, for example, support for querying, slicing, transforming, merging, and analyzing (including executing) models. Modeling languages are thus at the core of MDE, which participates in the development of a sound *Software Language Engineering* [1], including a unified typing theory that integrate models as first class entities [115].

Incorporating domain-specific concepts and high-quality development experience into MDE technologies can significantly improve developer productivity and system quality. Since the late nineties, this realization has led to work on MDE language workbenches that support the development of domain-specific modeling languages (DSMLs) and associated tools (*e.g.,* model editors and code generators). A DSML provides a bridge between the field in which domain experts work and the implementation

[1] See http://planet-sl.org

(programming) field. Domains in which DSMLs have been developed and used include, among others, automotive, avionics, and the emerging cyber-physical systems. A study performed by Hutchinson et al. [89] indicates that DSMLs can pave the way for wider industrial adoption of MDE.

More recently, the emergence of new classes of systems that are complex and operate in heterogeneous and rapidly changing environments raises new challenges for the software engineering community. These systems must be adaptable, flexible, reconfigurable and, increasingly, self-managing. Such characteristics make systems more prone to failure when running and thus development and study of appropriate mechanisms for continuous design and runtime validation and monitoring are needed. In the MDE community, research is focused primarily on using models at design, implementation, and deployment stages of development. This work has been highly productive, with several techniques now entering a commercialization phase. As software systems are becoming more and more dynamic, the use of model-driven techniques for validating and monitoring runtime behavior is extremely promising [98].

## 3.3 Variability modeling

While the basic vision underlying *Software Product Lines* (SPL) can probably be traced back to David Parnas' seminal article [105] on the Design and Development of Program Families, it is only quite recently that SPLs are emerging as a paradigm shift towards modeling and developing software system families rather than individual systems [102]. SPL engineering embraces the ideas of mass customization and software reuse. It focuses on the means of efficiently producing and maintaining multiple related software products, exploiting what they have in common and managing what varies among them.

Several definitions of the *software product line* concept can be found in the research literature. Clements *et al.* define it as *a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way* [103]. Bosch provides a different definition [72]: *A SPL consists of a product line architecture and a set of reusable components designed for incorporation into the product line architecture. In addition, the PL consists of the software products developed using the mentioned reusable assets.* In spite of the similarities, these definitions provide different perspectives of the concept: *market-driven*, as seen by Clements *et al.*, and *technology-oriented* for Bosch.

SPL engineering is a process focusing on capturing the *commonalities* (assumptions true for each family member) and *variability* (assumptions about how individual family members differ) between several software products [78]. Instead of describing a single software system, a SPL model describes a set of products in the same domain. This is accomplished by distinguishing between elements common to all SPL members, and those that may vary from one product to another. Reuse of core assets, which form the basis of the product line, is key to productivity and quality gains. These core assets extend beyond simple code reuse and may include the architecture, software components, domain models, requirements statements, documentation, test plans or test cases.

The SPL engineering process consists of two major steps:

1. **Domain Engineering**, or *development for reuse*, focuses on core assets development.

2. **Application Engineering**, or *development with reuse*, addresses the development of the final products using core assets and following customer requirements.

Central to both processes is the management of **variability** across the product line [85]. In common language use, the term *variability* refers to *the ability or the tendency to change*. Variability management is thus seen as the key feature that distinguishes SPL engineering from other software development approaches [73]. Variability management is thus growingly seen as the cornerstone of SPL development, covering the entire development life cycle, from requirements elicitation [117] to product derivation [122] to product testing [101, 100].

Halmans *et al.* [85] distinguish between *essential* and *technical* variability, especially at requirements level. Essential variability corresponds to the customer's viewpoint, defining what to implement, while technical variability relates to product family engineering, defining how to implement it. A classification based on the dimensions of variability is proposed by Pohl *et al.* [107]: beyond **variability in time** (existence of different versions of an artifact that are valid at different times) and **variability in space** (existence of an artifact in different shapes at the same time) Pohl *et al.* claim that variability is important

to different stakeholders and thus has different levels of visibility: **external variability** is visible to the customers while **internal variability**, that of domain artifacts, is hidden from them. Other classification proposals come from Meekel *et al.* [95] (feature, hardware platform, performances and attributes variability) or Bass *et al.* [64] who discusses about variability at the architectural level.

Central to the modeling of variability is the notion of *feature*, originally defined by Kang *et al.* as: *a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems* [91]. Based on this notion of *feature*, they proposed to use a *feature model* to model the variability in a SPL. A feature model consists of a *feature diagram* and other associated information: *constraints* and *dependency rules*. Feature diagrams provide a *graphical tree-like notation depicting the hierarchical organization of high level product functionalities* represented as features. The root of the tree refers to the complete system and is progressively decomposed into more refined features (tree nodes). Relations between nodes (features) are materialized by *decomposition edges* and *textual constraints*. Variability can be expressed in several ways. Presence or absence of a feature from a product is modeled using *mandatory* or *optional features*. Features are graphically represented as rectangles while some graphical elements (e.g., unfilled circle) are used to describe the variability (e.g., a feature may be optional).

Features can be organized into *feature groups*. Boolean operators *exclusive alternative (XOR)*, *inclusive alternative (OR)* or *inclusive (AND)* are used to select one, several or all the features from a feature group. Dependencies between features can be modeled using *textual constraints*: *requires* (presence of a feature requires the presence of another), *mutex* (presence of a feature automatically excludes another). Feature attributes can be also used for modeling quantitative (e.g., numerical) information. Constraints over attributes and features can be specified as well.

Modeling variability allows an organization to capture and select which version of which variant of any particular aspect is wanted in the system [73]. To implement it cheaply, quickly and safely, redoing by hand the tedious weaving of every aspect is not an option: some form of automation is needed to leverage the modeling of variability [68]. Model Driven Engineering (MDE) makes it possible to automate this weaving process [90]. This requires that models are no longer informal, and that the weaving process is itself described as a program (which is as a matter of facts an executable meta-model [99]) manipulating these models to produce for instance a detailed design that can ultimately be transformed to code, or to test suites [106], or other software artifacts.

## 3.4 Component-based software development

Component-based software development [116] aims at providing reliable software architectures with a low cost of design. Components are now used routinely in many domains of software system designs: distributed systems, user interaction, product lines, embedded systems, etc. With respect to more traditional software artifacts (e.g., object oriented architectures), modern component models have the following distinctive features [79]: description of requirements on services required from the other components; indirect connections between components thanks to ports and connectors constructs [93]; hierarchical definition of components (assemblies of components can define new component types); connectors supporting various communication semantics [76]; quantitative properties on the services [71].

In recent years component-based architectures have evolved from static designs to dynamic, adaptive designs (e.g., SOFA [76], Palladio [69], Frascati [96]). Processes for building a system using a statically designed architecture are made of the following sequential lifecycle stages: requirements, modeling, implementation, packaging, deployment, system launch, system execution, system shutdown and system removal. If for any reason after design time architectural changes are needed after system launch (e.g., because requirements changed, or the implementation platform has evolved, etc) then the design process must be reexecuted from scratch (unless the changes are limited to parameter adjustment in the components deployed).

Dynamic designs allow for *on the fly* redesign of a component based system. A process for dynamic adaptation is able to reapply the design phases while the system is up and running, without stopping it (this is different from a stop/redeploy/start process). Dynamic adaptation processes support *chosen adaptation*, when changes are planned and realized to maintain a good fit between the needs that the system must support and the way it supports them [92]. Dynamic component-based designs rely on a component meta-model that supports complex life cycles for components, connectors, service specification, etc. Advanced dynamic designs can also take platform changes into account at runtime,

without human intervention, by adapting themselves [77, 119]. Platform changes and more generally environmental changes trigger *imposed adaptation*, when the system can no longer use its design to provide the services it must support. In order to support an eternal system [70], dynamic component based systems must separate architectural design and platform compatibility. This requires support for heterogeneity, since platform evolution can be partial.

The Models@runtime paradigm denotes a model-driven approach aiming at taming the complexity of dynamic software systems. It basically pushes the idea of reflection one step further by considering the reflection layer as a real model "something simpler, safer or cheaper than reality to avoid the complexity, danger and irreversibility of reality [110]". In practice, component-based (and/or service-based) platforms offer reflection APIs that make it possible to introspect the system (to determine which components and bindings are currently in place in the system) and dynamic adaptation (by applying CRUD operations on these components and bindings). While some of these platforms offer rollback mechanisms to recover after an erroneous adaptation, the idea of Models@runtime is to prevent the system from actually enacting an erroneous adaptation. In other words, the "model at run-time" is a reflection model that can be uncoupled (for reasoning, validation, simulation purposes) and automatically resynchronized.

Heterogeneity is a key challenge for modern component based system. Until recently, component based techniques were designed to address a specific domain, such as embedded software for command and control, or distributed Web based service oriented architectures. The emergence of the Internet of Things paradigm calls for a unified approach in component based design techniques. By implementing an efficient separation of concern between platform independent architecture management and platform dependent implementations, *Models@runtime* is now established as a key technique to support dynamic component based designs. It provides DIVERSE with an essential foundation to explore an adaptation envelop at run-time.

Search Based Software Engineering [87] has been applied to various software engineering problems in order to support software developers in their daily work. The goal is to automatically explore a set of alternatives and assess their relevance with respect to the considered problem. These techniques have been applied to craft software architecture exhibiting high quality of services properties [84]. Multi Objectives Search based techniques [80] deal with optimization problem containing several (possibly conflicting) dimensions to optimize. These techniques provide DIVERSE with the scientific foundations for reasoning and efficiently exploring an envelope of software configurations at run-time.

## 3.5 Validation and verification

Validation and verification (V&V) theories and techniques provide the means to assess the validity of a software system with respect to a specific correctness envelop. As such, they form an essential element of DIVERSE's scientific background. In particular, we focus on model-based V&V in order to leverage the different models that specify the envelop at different moments of the software development lifecycle.

Model-based testing consists in analyzing a formal model of a system (*e.g.*, activity diagrams, which capture high-level requirements about the system, statecharts, which capture the expected behavior of a software module, or a feature model, which describes all possible variants of the system) in order to generate test cases that will be executed against the system. Model-based testing [118] mainly relies on model analysis, constraint solving [81] and search-based reasoning [94]. DIVERSE leverages in particular the applications of model-based testing in the context of highly-configurable systems and [120] interactive systems [97] as well as recent advances based on diversity for test cases selection [88].

Nowadays, it is possible to simulate various kinds of models. Existing tools range from industrial tools such as Simulink, Rhapsody or Telelogic to academic approaches like Omega [104], or Xholon [2]. All these simulation environments operate on homogeneous environment models. However, to handle diversity in software systems, we also leverage recent advances in heterogeneous simulation. Ptolemy [75] proposes a common abstract syntax, which represents the description of the model structure. These elements can be decorated using different directors that reflect the application of a specific model of computation on the model element. Metropolis [65] provides modeling elements amenable to semantically equivalent mathematical models. Metropolis offers a precise semantics flexible enough to support different models of computation. ModHel'X [86] studies the composition of multi-paradigm models relying on different

[2] http://www.primordion.com/Xholon/

models of computation.

Model-based testing and simulation are complemented by runtime fault-tolerance through the automatic generation of software variants that can run in parallel, to tackle the open nature of software-intensive systems. The foundations in this case are the seminal work about N-version programming [63], recovery blocks [108] and code randomization [67], which demonstrated the central role of diversity in software to ensure runtime resilience of complex systems. Such techniques rely on truly diverse software solutions in order to provide systems with the ability to react to events, which could not be predicted at design time and checked through testing or simulation.

## 3.6 Empirical software engineering

The rigorous, scientific evaluation of DIVERSE's contributions is an essential aspect of our research methodology. In addition to theoretical validation through formal analysis or complexity estimation, we also aim at applying state-of-the-art methodologies and principles of empirical software engineering. This approach encompasses a set of techniques for the sound validation contributions in the field of software engineering, ranging from statistically sound comparisons of techniques and large-scale data analysis to interviews and systematic literature reviews [113, 111]. Such methods have been used for example to understand the impact of new software development paradigms [74]. Experimental design and statistical tests represent another major aspect of empirical software engineering. Addressing large-scale software engineering problems often requires the application of heuristics, and it is important to understand their effects through sound statistical analyses [62].

## 3.7 Research axis

Figure 1 illustrates the four dimensions of software diversity, which form the core research axis of DIVERSE: the **diversity of languages** used by the stakeholders involved in the construction of these systems; the **diversity of features** required by the different customers; the **diversity of runtime environments** in which software has to run and adapt; the **diversity of implementations** that are necessary for resilience through redundancy. These four axes share and leverage the scientific and technological results developed in the area of model-driven engineering in the last decade. This means that all our research activities are founded on sound abstractions to reason about specific aspects of software systems, compose different perspectives and automatically generate parts of the system.

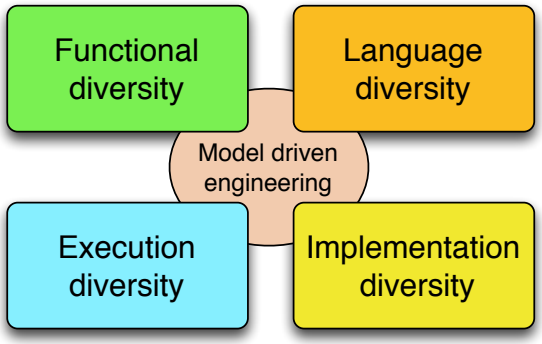


Figure 1: The four research axes of DIVERSE, which rely on a MDE scientific background

## 3.8 Software Language Engineering

The engineering of systems involves many different stakeholders, each with their own domain of expertise. Hence more and more organizations are adopting Domain Specific Modeling Languages (DSMLs) to allow domain experts to express solutions directly in terms of relevant domain concepts [112, 83]. This new trend raises new challenges about designing DSMLs, evolving a set of DSMLs and coordinating the use of multiple DSLs for both DSL designers and DSL users.

### Challenges

**Reusability** of software artifacts is a central notion that has been thoroughly studied and used by both academics and industrials since the early days of software construction. Essentially, designing reusable artifacts allows the construction of large systems from smaller parts that have been separately developed and validated, thus reducing the development costs by capitalizing on previous engineering efforts. However, it is still hardly possible for language designers to design typical language artifacts (e.g. language constructs, grammars, editors or compilers) in a reusable way. The current state of the practice usually prevents the reusability of language artifacts from one language to another, consequently hindering the emergence of real engineering techniques around software languages. Conversely, concepts and mechanisms that enable artifacts reusability abound in the software engineering community.

**Variability** in modeling languages occur in the definition of the abstract and concrete syntax as well as in the specification of the language's semantics. The major challenges met when addressing the need for variability are: (i) to set principles for modeling language units that support the modular specification of a modeling language; and (ii) to design mechanisms to assemble these units into a complete language, according to the set of authorized variation points for the modeling language family.

A new generation of complex software-intensive systems (for example smart health support, smart grid, building energy management, and intelligent transportation systems) gives new opportunities for leveraging modeling languages. The development of these systems requires expertise in diverse domains. Consequently, different types of stakeholders (e.g., scientists, engineers and end-users) must work in a coordinated manner on various aspects of the system across multiple development phases. DSMLs can be used to support the work of domain experts who focus on a specific system aspect, but they can also provide the means for coordinating work across teams specializing in different aspects and across development phases. The support and integration of DSMLs leads to what we call **the globalization of modeling languages**, *i.e.* the use of multiple languages for the coordinated development of diverse aspects of a system. One can make an analogy with world globalization in which relationships are established between sovereign countries to regulate interactions (e.g., travel and commerce related interactions) while preserving each country's independent existence.

### Scientific objectives

We address reuse and variability challenges through the investigation of the time-honored concepts of substitutability, inheritance and components, evaluate their relevance for language designers and provide tools and methods for their inclusion in software language engineering. We will develop novel techniques for the modular construction of language extensions with support to model syntactical variability. From the semantics perspective, we investigate extension mechanisms for the specification of variability in operational semantics, focusing on static introduction and heterogeneous models of computation. The definition of variation points for the three aspects of the language definition provides the foundations for the novel concept Language Unit (LU) as well as suitable mechanisms to compose such units.

We explore the necessary breakthrough in software languages to support modeling and simulation of heterogeneous and open systems. This work relies on the specification of executable domain specific modeling languages (DSMLs) to formalize the various concerns of a software-intensive system, and of models of computation (MoCs) to explicitly model the concurrency, time and communication of such DSMLs. We develop a framework that integrates the necessary foundations and facilities for designing and implementing executable and concurrent domain-specific modeling languages. This framework also provides unique features to specify composition operators between (possibly heterogeneous) DSMLs. Such specifications are amenable to support the edition, execution, graphical animation and analysis of heterogeneous models. The objective is to provide both a significant improvement to MoCs and DSMLs design and implementation and to the simulation based validation and verification of complex systems.

We see an opportunity for the automatic diversification of programs' computation semantics, for example through the diversification of compilers or virtual machines. The main impact of this artificial diversity is to provide flexible computation and thus ease adaptation to different execution conditions. A combination of static and dynamic analysis could support the identification of what we call *plastic computation zones* in the code. We identify different categories of such zones: (i) areas in the code in which the order of computation can vary (e.g., the order in which a block of sequential statements

is executed); (ii) areas that can be removed, keeping the essential functionality [114] (e.g., skip some loop iterations); (iii) areas that can replaced by alternative code (e.g., replace a try-catch by a return statement). Once we know which zones in the code can be randomized, it is necessary to modify the model of computation to leverage the computation plasticity. This consists in introducing variation points in the interpreter to reflect the diversity of models of computation. Then, the choice of a given variation is performed randomly at run time.

## 3.9 Variability Modeling and Engineering

The systematic modeling of variability in software systems has emerged as an effective approach to document and reason about software evolution and heterogeneity (*cf.* Section 3.3). Variability modeling characterizes an "envelope" of possible software variations. The industrial use of variability models and their relation to software artifact models require a complete engineering framework, including composition, decomposition, analysis, configuration and artifact derivation, refactoring, re-engineering, extraction, and testing. This framework can be used both to tame imposed diversity and to manage chosen diversity.

### Challenges

A fundamental problem is that the **number of variants** can be exponential in the number of options (features). Already with 300 boolean configuration options, approximately $10^{90}$ configurations exist – more than the estimated count of atoms in the universe. Domains like automotive or operating systems have to manage more than 10000 options (e.g., Linux). Practitioners face the challenge of developing billions of variants. It is easy to forget a necessary constraint, leading to the synthesis of unsafe variants, or to under-approximate the capabilities of the software platform. Scalable modelling techniques are therefore crucial to specify and reason about a very large set of variants.

Model-driven development supports two approaches to deal with the increasing number of concerns in complex systems: multi-view modeling, *i.e.* when modeling each concern separately, and variability modeling. However, there is little support to combine both approaches consistently. Techniques to integrate both approaches will enable the construction of a consistent set of views and variation points in each view.

The design, construction and maintenance of software families have a major impact on **software testing**. Among the existing challenges, we can cite: the selection of test cases for a specific variant; the evolution of test suites with integration of new variants; the combinatorial explosion of the number of software configurations to be tested. Novel model-based techniques for test generation and test management in a software product line context are needed to overcome state-of-the-art limits we already observed in some projects.

### Scientific objectives

We aim at developing scalable reasoning techniques to **automatically analyze** variability models and their interactions with other views on the software intensive system (requirements, architecture, design, code). These techniques provide two major advancements in the state of the art: (1) an extension of the semantics of variability models in order to enable the definition of attributes (*e.g.*, cost, quality of service, effort) on features and to include these attributes in the reasoning; (2) an assessment of the consistent specification of variability models with respect to system views (since variability is orthogonal to system modeling, it is currently possible to specify the different models in ways that are semantically meaningless). The former aspect of analysis is tackled through constraint solving and finite-domain constraint programming, while the latter aspect is investigated through automatic search-based and learning-based techniques for the exploration of the space of interaction between variability and view models.

We aim at developing procedures to **reverse engineer** dependencies and features' sets from existing software artefacts – be it source code, configuration files, spreadsheets (e.g., product comparison matrices) or requirements. We expect to scale up (e.g., for extracting a very large number of variation points) and guarantee some properties (e.g., soundness of configuration semantics, understandability of ontological

semantics). For instance, when building complex software-intensive systems, textual requirements are captured in very large quantities of documents. In this context, adequate models to formalize the organization of requirements documents and automated techniques to support impact analysis (in case of changes in the requirements) have to be developed.

## 3.10 Heterogeneous and dynamic software architectures

Flexible yet dependable systems have to cope with heterogeneous hardware execution platforms ranging from smart sensors to huge computation infrastructures and data centers. Evolution possibilities range from a mere change in the system configuration to a major architectural redesign, for instance to support addition of new features or a change in the platform architecture (e.g., new hardware is made available, a running system switches to low bandwidth wireless communication, a computation node battery is running low, etc). In this context, we need to devise formalisms to reason about the impact of an evolution and about the transition from one configuration to another. It must be noted that this axis focuses on the use of models to drive the evolution from design time to runtime. Models will be used to (i) systematically define predictable configurations and variation points through which the system will evolve; (ii) develop behaviors necessary to handle unforeseen evolution cases.

### Challenges

The main challenge is to provide new homogeneous architectural modelling languages and efficient techniques that enable continuous software reconfiguration to react to changes. This work handles the challenges of handling the diversity of runtime infrastructures and managing the cooperation between different stakeholders. More specifically, the research developed in this axis targets the following dimensions of software diversity.

Platform architectural heterogeneity induces a first dimension of imposed diversity (type diversity). Platform reconfiguration driven by changing resources define another dimension of diversity (deployment diversity). To deal with these imposed diversity problems, we will rely on model based runtime support for adaptation, in the spirit of the dynamic distributed component framework developed by the Triskell team. Since the runtime environment composed of distributed, resource constrained hardware nodes cannot afford the overhead of traditional runtime adaptation techniques, we investigate the design of novel solutions relying on Models@runtime and on specialized tiny virtual machines to offer resource provisioning and dynamic reconfiguration.

Diversity can also be an asset to optimize software architecture. Architecture models must integrate multiple concerns in order to properly manage the deployment of software components over a physical platform. However, these concerns can contradict each other (*e.g.*, accuracy and energy). In this context, we investigate automatic solutions to explore the set of possible architecture models and to establish valid trade-offs between all concerns in case of changes.

### Scientific objectives

**Automatic synthesis of optimal software architectures.** Implementing a service over a distributed platform (*e.g.*, a pervasive system or a cloud platform) consists in deploying multiple software components over distributed computation nodes. We aim at designing search-based solutions to (i) assist the software architect in establishing a good initial architecture (that balances between different factors such as cost of the nodes, latency, fault tolerance) and to automatically update the architecture when the environment or the system itself change. The choice of search-based techniques is motivated by the very large number of possible software deployment architectures that can be investigated and that all provide different trade-offs between qualitative factors. Another essential aspect that is supported by multi-objective search is to explore different architectural solutions that are not necessarily comparable. This is important when the qualitative factors are orthogonal to each other, such as security and usability for example.

**Flexible software architecture for testing and data management.** As the number of platforms on which software runs increases and different software versions coexist, the demand for testing environments also increases. For example, the number of testing environments to test a software patch or upgrade is the product of the number of execution environments the software supports and the number

of coexisting versions of the software. Based on our first experiment on the synthesis of cloud environment using architectural models, our objective is to define a set of domain specific languages to catch the requirement and to design cloud environments for testing and data management of future internet systems from data centers to things. These languages will be interpreted to support dynamic synthesis and reconfiguration of a testing environment.

**Runtime support for heterogeneous environments.** Execution environments must provide a way to account or reserve resources for applications. However, current execution environments such as the Java Virtual Machine do not clearly define a notion of application: each framework has its own definition. For example, in OSGi, an application is a component, in JEE, an application is most of the time associated to a class loader, in the Multi-Tasking Virtual machine, an application is a process. The challenge consists in defining an execution environment that provides direct control over resources (CPU, Memory, Network I/O) independently from the definition of an application. We propose to define abstract resource containers to account and reserve resources on a distributed network of heterogeneous devices.

## 3.11 Diverse implementations for resilience

Open software-intensive systems have to evolve over their lifetime in response to changes in their environment. Yet, most verification techniques assume a closed environment or the ability to predict all changes. Dynamic changes and evolution cases thus represent a major challenge for these techniques that aim at assessing the correctness and robustness of the system. On the one hand, DIVERSE will adapt V&V techniques to handle diversity imposed by the requirements and the execution environment, on the other hand we leverage diversity to increase the robustness of software in face of unforeseen situations. More specifically, we address the following V&V challenges.

### Challenges

One major challenge to build flexible and open yet dependable systems is that current software engineering techniques require architects to foresee all possible situations the system will have to face. However, openness and flexibility also mean unpredictability: unpredictable bugs, attacks, environmental evolution, etc. Current fault-tolerance [108] and security [82] techniques provide software systems with the capacity of detecting accidental and deliberate faults. However, existing solutions assume that the set of bugs or vulnerabilities in a system does not evolve. This assumption does not hold for open systems, thus it is essential to revisit fault-tolerance and security solutions to account for diverse and unpredictable faults.

Diversity is known to be a major asset for the robustness of large, open, and complex systems (*e.g.*, economical or ecological systems). Following this observation, the software engineering literature provides a rich set of work that rely on implementation diversity in software systems in order to improve robustness to attacks or to changes in quality of service. These works range from N-version programming to obfuscation of data structures or control flow, to randomization of instruction sets. An essential and active challenge is to support the automatic synthesis and evolution of software diversity in open software-intensive systems. There is an opportunity to further enhance these techniques in order to cope with a wider diversity of faults, by multiplying the levels of diversity in the different software layers that are found in software-intensive systems (system, libraries, frameworks, application). This increased diversity must be based on artificial program transformations and code synthesis, which increase the chances of exploring novel solutions, better fitted at one point in time. The biological analogy also indicates that diversity should emerge as a side-effect of evolution, to prevent over-specialization towards one kind of diversity.

### Scientific objectives

The main objective is to address one of the main limitations of N-version programming for fault-tolerant systems: the manual production and management of software diversity. Through automated injection of artificial diversity we aim at systematically increasing failure diversity and thus increasing the chances

of early error detection at run-time. A fundamental assumption for this work is that software-intensive systems can be “good enough” [109, 121].

**Proactive program diversification.** We aim at establishing novel principles and techniques that favor the emergence of multiple forms of software diversity in software-intensive systems, in conjunction with the software adaptation mechanisms that leverage this diversity. The main expected outcome is a set of meta-design principles that maintain diversity in systems and the experimental demonstration of the effects of software diversity. Higher levels of diversity in the system provide a pool of software solutions that can eventually be used to adapt to situations unforeseen at design time (bugs, crash, attacks, etc.). Principles of automated software diversification rely on the automated synthesis of variants in a software product line, as well as finer-grained program synthesis combining unsound transformations and genetic programming to explore the space of mutational robustness.

**Multi-tier software diversification.** We name multi-tier diversification the fact of diversifying several application software components simultaneously. The novelty of our proposal, with respect to the software diversity state of the art, is to diversify the application-level code (for example, diversify the business logic of the application), focusing on the technical layers found in web applications. The diversification of application software code is expected to provide a diversity of failures and vulnerabilities in web server deployment. Web server deployment usually adopts a form of the Reactor architecture pattern, for scalability purposes: multiple copies of the server software stack, called request handlers, are deployed behind a load balancer. This architecture is very favorable for diversification, since by using the multiplicity of request handlers running in a web server we can simultaneously deploy multiple combinations of diverse software components. Then, if one handler is hacked or crashes the others should still be able to process client requests.

# 4 Application domains

Information technology affects all areas of society. The need to develop software systems is therefore present in a huge number of application domains. One of the goals of software engineering is to *apply a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software* whatever the application domain.

As a result, the team covers a wide range of application domain and never refrains from exploring a particular field of application. Our primary expertise is in complex, heterogeneous and distributed systems. While we historically collaborated with partners in the field of systems engineering, it should be noted that for several years now, we have investigated several new areas in depth:

- the field of web applications, with the associated design principles and architectures, for applications ranging from cloud-native applications to the design of modern web front-ends.
- the field of scientific computing in connection with the CEA DAM, Safran and scientists from other disciplines such as the ecologists of the University of Rennes 1. In this field where the writing of complex software is common, we explore how we could help scientists using software engineering approach in particular the use of SLE and approximate computing techniques.
- the field of large software systems such as the kernel Linux or other open-source projects. In this field, we explore in particular the variability management, the support of co-evolution and the use of polyglot approaches.

# 5 Highlights of the year

## 5.1 Awards

Jean-Marc Jézéquel has been awarded the ACM/IEEE MODELS 2020 Career Award.

The paper "Sampling Effect on Performance Prediction of Configurable Systems: A Case Study" (Juliana Alves Pereira, Mathieu Acher, Hugo Martin and Jean-Marc Jézéquel ) received the best paper Award at ICPE 2020 - 11th ACM/SPEC International Conference on Performance Engineering, ACM, Apr 2020, Edmonton, Canada.

# 6 New software and platforms

## 6.1 New software

### 6.1.1 amiunique

**Name:** amiunique

**Keywords:** Privacy, Browser fingerprinting

**Scientific Description:** The amiunique web site has been deployed in 2014 in the context of the DiverSE team research activities on browser fingerprinting to understand how software diversity can be leveraged to mitigate the impact of fingerprinting on the privacy of users. In 2018, it was migrated to the Spirals team where the research on browser fingerprinting still continues to this day.

The web site has yielded multiple datasets of genuine fingerprints to understand the multiple facets of browser fingerprinting and how they can be used on the web to reinforce security. The web site presents regular updates to include the latest development in web technology and understand their impact of users' privacy.

The whole source code of amiunique is open source and is distributed under the terms of the MIT license.

Main innovative features:

- canvas fingerprinting
- WebGL fingerprinting
- advanced JS features (platform, DNT, etc.)

Impact: The website has been visited by more than 3,000,000 unique visitors since its creation and it has been showcased in several professional forums and tutorial sessions over the years. It produced multiple datasets over the years that were used in articles published in top-tier conferences. Amiunique has received in 2018 the prize "Protection de la vie privée" granted by Inria and the CNIL. The research around fingerprints in amiunique has also been a source of influence for the Brave web browser.

**Functional Description:** This web site aims at informing visitors about browser fingerprinting and possible tools to mitigate its effect, as well as at collecting data about the fingerprints that can be found on the web. It collects browser fingerprints with the explicit agreement of the users (they have to click on a button on the home page). Fingerprints are composed of 17 attributes, which include regular HTTP headers as well as the most recent state of the art techniques (canvas fingerprinting, WebGL information).

**URL:** https://amiunique.org/

**Authors:** Pierre Laperdrix, Antonin Durey, Walter Rudametkin Ivey

**Contacts:** Benoit Baudry, Pierre Laperdrix

**Partners:** INSA Rennes, Université de Lille

### 6.1.2 FAMILIAR

**Keywords:** Software line product, Configators, Customisation

**Scientific Description:** FAMILIAR (for FeAture Model scrIpt Language for manIpulation and Automatic Reasoning) is a language for importing, exporting, composing, decomposing, editing, configuring, computing "diffs", refactoring, reverse engineering, testing, and reasoning about (multiple) feature models. All these operations can be combined to realize complex variability management tasks. A comprehensive environment is proposed as well as integration facilities with the Java ecosystem.

**Functional Description:** Familiar is an environment for large-scale product customisation. From a model of product features (options, parameters, etc.), Familiar can automatically generate several million variants. These variants can take many forms: software, a graphical interface, a video sequence or even a manufactured product (3D printing). Familiar is particularly well suited for developing web configurators (for ordering customised products online), for providing online comparison tools and also for engineering any family of embedded or software-based products.

**URL:** http://familiar-project.github.com

**Contact:** Mathieu Acher

**Participants:** Aymeric Hervieu, Benoit Baudry, Didier Vojtisek, Edward Mauricio Alferez Salinas, Guillaume Bécan, Joao Bosco Ferreira-Filho, Julien Richard-Foy, Mathieu Acher, Olivier Barais, Sana Ben Nasr

#### 6.1.3 GEMOC Studio

**Name:** GEMOC Studio

**Keywords:** DSL, Language workbench, Model debugging

**Scientific Description:** The language workbench put together the following tools seamlessly integrated to the Eclipse Modeling Framework (EMF):

- Melange, a tool-supported meta-language to modularly define executable modeling languages with execution functions and data, and to extend (EMF-based) existing modeling languages.
- MoCCML, a tool-supported meta-language dedicated to the specification of a Model of Concurrency and Communication (MoCC) and its mapping to a specific abstract syntax and associated execution functions of a modeling language.
- GEL, a tool-supported meta-language dedicated to the specification of the protocol between the execution functions and the MoCC to support the feedback of the data as well as the callback of other expected execution functions.
- BCOoL, a tool-supported meta-language dedicated to the specification of language coordination patterns to automatically coordinates the execution of, possibly heterogeneous, models.
- Sirius Animator, an extension to the model editor designer Sirius to create graphical animators for executable modeling languages.

**Functional Description:** The GEMOC Studio is an Eclipse package that contains components supporting the GEMOC methodology for building and composing executable Domain-Specific Modeling Languages (DSMLs). It includes two workbenches: The GEMOC Language Workbench: intended to be used by language designers (aka domain experts), it allows to build and compose new executable DSMLs. The GEMOC Modeling Workbench: intended to be used by domain designers to create, execute and coordinate models conforming to executable DSMLs. The different concerns of a DSML, as defined with the tools of the language workbench, are automatically deployed into the modeling workbench. They parametrize a generic execution framework that provides various generic services such as graphical animation, debugging tools, trace and event managers, timeline.

**URL:** http://gemoc.org/studio.html

**Authors:** Didier Vojtisek, Benoît Combemale, Cédric Brun, François Tanguy, Joël Champeau, Julien DeAntoni, Xavier Crégut

**Contacts:** Benoît Combemale, Julien DeAntoni

**Participants:** Didier Vojtisek, Dorian Leroy, Erwan Bousse, Fabien Coulon, Julien DeAntoni

**Partners:** IRIT, ENSTA, I3S, OBEO, Thales TRT

### 6.1.4 Kevoree

**Keywords:** M2M, Dynamic components, Iot, Heterogeneity, Smart home, Cloud, Software architecture, Dynamic deployment

**Scientific Description:** Kevoree is an open-source models@runtime platform (http://www.kevoree.org ) to properly support the dynamic adaptation of distributed systems. Models@runtime basically pushes the idea of reflection [132] one step further by considering the reflection layer as a real model that can be uncoupled from the running architecture (e.g. for reasoning, validation, and simulation purposes) and later automatically resynchronized with its running instance.

Kevoree has been influenced by previous work that we carried out in the DiVA project [132] and the Entimid project [135] . With Kevoree we push our vision of models@runtime [131] farther. In particular, Kevoree provides a proper support for distributed models@runtime. To this aim we introduced the Node concept to model the infrastructure topology and the Group concept to model semantics of inter node communication during synchronization of the reflection model among nodes. Kevoree includes a Channel concept to allow for multiple communication semantics between remoteComponents deployed on heterogeneous nodes. All Kevoree concepts (Component, Channel, Node, Group) obey the object type design pattern to separate deployment artifacts from running artifacts. Kevoree supports multiple kinds of very different execution node technology (e.g. Java, Android, MiniCloud, FreeBSD, Arduino, ...).

Kevoree is distributed under the terms of the LGPL open source license.

Main competitors:

- the Fractal/Frascati eco-system (http://frascati.ow2.org/doc/1.4/frascati-userguide.html).
- SpringSource Dynamic Module (http://spring.io/)
- GCM-Proactive (http://proactive.inria.fr/)
- OSGi (http://www.osgi.org)
- Chef
- Vagran (http://vagrantup.com/)

Main innovative features:

- distributed models@runtime platform (with a distributed reflection model and an extensible models@runtime dissemination set of strategies).
- Support for heterogeneous node type (from Cyber Physical System with few resources until cloud computing infrastructure).
- Fully automated provisioning model to correctly deploy software modules and their dependencies.
- Communication and concurrency access between software modules expressed at the model level (not in the module implementation).

**Functional Description:** Kevoree is an open-source models@runtime platform to properly support the dynamic adaptation of distributed systems. Models@runtime basically pushes the idea of reflection one step further by considering the reflection layer as a real model that can be uncoupled from the running architecture (e.g. for reasoning, validation, and simulation purposes) and later automatically resynchronized with its running instance.

**URL:** http://kevoree.org/

**Authors:** Jean Emile Dartois, Aymeric Hervieu, Olivier Barais

**Contact:** Olivier Barais

**Participants:** Aymeric Hervieu, Benoit Baudry, Francisco-Javier Acosta Padilla, Inti Gonzalez Herrera, Ivan Paez Anaya, Jacky Bourgeois, Jean Emile Dartois, Johann Bourcier, Manuel Leduc, Maxime Tricoire, Mohamed Boussaa, Noël Plouzeau, Olivier Barais

#### 6.1.5 Melange

**Name:** Melange

**Keywords:** Model-driven engineering, Meta model, MDE, DSL, Model-driven software engineering, Dedicated langage, Language workbench, Meta-modelisation, Modeling language, Meta-modeling

**Scientific Description:** Melange is a follow-up of the executable metamodeling language Kermeta, which provides a tool-supported dedicated meta-language to safely assemble language modules, customize them and produce new DSMLs. Melange provides specific constructs to assemble together various abstract syntax and operational semantics artifacts into a DSML. DSMLs can then be used as first class entities to be reused, extended, restricted or adapted into other DSMLs. Melange relies on a particular model-oriented type system that provides model polymorphism and language substitutability, i.e. the possibility to manipulate a model through different interfaces and to define generic transformations that can be invoked on models written using different DSLs. Newly produced DSMLs are correct by construction, ready for production (i.e., the result can be deployed and used as-is), and reusable in a new assembly.

Melange is tightly integrated with the Eclipse Modeling Framework ecosystem and relies on the meta-language Ecore for the definition of the abstract syntax of DSLs. Executable meta-modeling is supported by weaving operational semantics defined with Xtend. Designers can thus easily design an interpreter for their DSL in a non-intrusive way. Melange is bundled as a set of Eclipse plug-ins.

**Functional Description:** Melange is a language workbench which helps language engineers to mashup their various language concerns as language design choices, to manage their variability, and support their reuse. It provides a modular and reusable approach for customizing, assembling and integrating DSMLs specifications and implementations.

**URL:** http://melange-lang.org

**Contacts:** Benoît Combemale, Thomas Degueule, Olivier Barais, Jean-Marc Jézéquel, Didier Vojtisek

**Participants:** Arnaud Blouin, Benoît Combemale, David Mendez Acuna, Didier Vojtisek, Dorian Leroy, Erwan Bousse, Fabien Coulon, Jean-Marc Jézéquel, Olivier Barais, Thomas Degueule

#### 6.1.6 DSpot

**Keywords:** Software testing, Test amplification

**Functional Description:** DSpot is a tool that generates missing assertions in JUnit tests. DSpot takes as input a Java project with an existing test suite. As output, DSpot outputs new test cases on console. DSpot supports Java projects built with Maven and Gradle

**URL:** https://github.com/STAMP-project/dspot

**Authors:** Simon Allier, Benoit Baudry, Marcelino Rodriguez Cancio, Martin Monperrus

**Contacts:** Benoit Baudry, Benjamin Danglot

**Participants:** Benoit Baudry, Martin Monperrus, Benjamin Danglot

**Partner:** KTH Royal Institute of Technology

#### 6.1.7 ALE

**Name:** Action Language for Ecore

**Keywords:** Meta-modeling, Executable DSML

**Functional Description:** Main features of ALE include:

- Executable metamodeling: Re-open existing EClasses to insert new methods with their implementations
- Metamodel extension: The very same mechanism can be used to extend existing Ecore metamodels and insert new features (eg. attributes) in a non-intrusive way
- Interpreted: No need to deploy Eclipse plugins, just run the behavior on a model directly in your modeling environment
- Extensible: If ALE doesn't fit your needs, register Java classes as services and invoke them inside your implementations of EOperations.

**URL:** http://gemoc.org/ale-lang/

**Contact:** Benoît Combemale

**Partner:** OBEO

#### 6.1.8 InspectorGuidget

**Keywords:** Static analysis, Software testing, User Interfaces

**Functional Description:** InspectorGuidget is a static code analysing tool. InspectorGuidget analyses UI (user interface/interaction) code of a software system to extract high level information and metrics. InspectorGuidget also finds bad UI coding pratices, such as Blob listener instances. InspectorGuidget analyses Java code.

**URL:** https://github.com/diverse-project/InspectorGuidget

**Publications:** hal-01499106v5, hal-01308625v2

**Contact:** Arnaud Blouin

**Participants:** Arnaud Blouin, Benoit Baudry

#### 6.1.9 Descartes

**Keywords:** Software testing, Mutation analysis

**Functional Description:** Descartes evaluates the capability of your test suite to detect bugs using extreme mutation testing.

Descartes is a mutation engine plugin for PIT which implements extreme mutation operators as proposed in the paper *Will my tests tell me if I break this code?*.

**URL:** https://github.com/STAMP-project/pitest-descartes

**Publications:** hal-01870976, hal-01867423

**Contacts:** Benoit Baudry, Oscar Luis Vera Perez, Olivier Barais, Martin Monperrus

**Participants:** Oscar Luis Vera Perez, Benjamin Danglot, Benoit Baudry, Martin Monperrus

**Partner:** KTH Royal Institute of Technology

#### 6.1.10 PitMP

**Name:** PIT for Multi-module Project

**Keywords:** Mutation analysis, Mutation testing, Java, JUnit, Maven

**Functional Description:** PIT and Descartes are mutation testing systems for Java applications, which allows you to verify if your test suites can detect possible bugs, and so to evaluate the quality of your test suites. They evaluate the capability of your test suite to detect bugs using mutation testing (PIT) or extreme mutation testing (Descartes). Mutation testing does it by introducing small changes or faults into the original program. These modified versions are called mutants. A good test suite should able to kill or detect a mutant. Traditional mutation testing works at the instruction level, e.g., replacing ">" by "<=", so the number of generated mutants is huge, as the time required to check the entire test suite. That's why Extreme Mutation strategy appeared. In Extreme Mutation testing, the whole body of a method under test is removed. Descartes is a mutation engine plugin for PIT which implements extreme mutation operators. Both provide reports combining, line coverage, mutation score and list of weaknesses in the source.

**URL:** https://github.com/STAMP-project/pitmp-maven-plugin

**Contact:** Caroline Landry

**Partners:** CSQE, KTH Royal Institute of Technology, ENGINEERING

# 7 New results

## 7.1 Results on Variability modeling and management

**Participants** Mathieu Acher, Arnaud Blouin, Jean-Marc Jezequel.

In general, we are currently exploring the use of machine learning for variability-intensive systems in the context of VaryVary ANR project https://varyvary.github.io.

### 7.1.1 Deep software variability

**Empirical Assessment of Generating Adversarial Configurations for Software Product Lines** Software product line (SPL) engineering allows the derivation of products tailored to stakeholders' needs through the setting of a large number of configuration options. Unfortunately, options and their interactions create a huge configuration space, which is either intractable or too costly to explore exhaustively. Instead of covering all products, machine learning (ML) approximates the set of acceptable products (e.g., successful builds, passing tests) out of a training set (a sample of configurations). However, ML techniques can make prediction errors yielding non-acceptable products wasting time, energy and other resources. We apply adversarial machine learning techniques to the world of SPLs and craft new configurations faking to be acceptable configurations but that are not and vice-versa [39]. It allows to diagnose prediction errors and take appropriate actions. We developed two adversarial configuration generators on top of state-of-the-art attack algorithms and capable of synthesizing configurations that are both adversarial and conform to logical constraints. We empirically assessed our generators using two case studies: an industrial video synthesizer (MOTIV) and an industry-strength, open-source Web-app configurator (JHipster). For the two cases, our attacks yield (up to) a 100% misclassification rate without sacrificing the logical validity of adversarial configurations. This work lays the foundations of a quality assurance framework for ML-based SPLs.

**Sampling Effect on Performance Prediction of Configurable Systems: A Case Study** Numerous software systems are highly configurable and provide a myriad of configuration options that users can tune to fit their functional and performance requirements (e.g., execution time). Measuring all configurations of a system is the most obvious way to understand the effect of options and their interactions, but is too costly or infeasible in practice. Numerous works thus propose to measure only a few configurations (a sample) to learn and predict the performance of any combination of options' values. A challenging issue is to sample a small and representative set of configurations that leads to a good accuracy of performance prediction models. A recent study devised a new algorithm, called distance-based sampling, that obtains state-of-the-art accurate performance predictions on different subject systems. In this work, we replicate this study through an in-depth analysis of x264, a popular and configurable video encoder. We systematically measure all 1,152 configurations of x264 with 17 input videos and two quantitative properties (encoding time and encoding size) [43]. Our goal is to understand whether there is a dominant sampling strategy over the very same subject system (x264), i.e., whatever the workload and targeted performance properties. The findings from this study show that random sampling leads to more accurate performance models. However, without considering random, there is no single "dominant" sampling, instead different strategies perform best on different inputs and non-functional properties, further challenging practitioners and researchers.

**Deep Software Variability** Configuring software is a powerful means to reach functional and performance goals of a system. However, many layers (hardware, operating system, input data, etc.), themselves subject to variability, can alter performances of software configurations. For instance, configuration options of the x264 video encoder may have very different effects on x264's encoding time when used with different input videos, depending on the hardware on which it is executed. In this work, we coin the term *deep software variability* to refer to the interaction of all external layers modifying the behavior or non-functional properties of a software. Deep software variability challenges practitioners and researchers: the combinatorial explosion of possible executing environments complicates the understanding, the configuration, the maintenance, the debug, and the test of configurable systems. There are also opportunities: harnessing all variability layers (and not only the software layer) can lead to more efficient systems and configuration knowledge that truly generalizes to any usage and context [51].

### 7.1.2 Managing the software variability at the source code level

**A framework for managing the imperfect modularity of variability implementations** In many industrial settings, the common and varying features of related software-intensive systems, as their reusable units, are likely to be implemented by a combined set of traditional techniques. Features do not align perfectly well with the used language constructs, e.g., classes, thus hindering the management of implemented variability. Herein, we provide a detailed framework to capture, model, and trace this imperfectly modular variability in terms of variation points with variants [40]. We describe an implementation of this framework, as a domain-specific language, and report on its application on four subject systems and usage for variability management, showing its feasibility.

## 7.2 Results on Software Language Engineering

**Participants** Olivier Barais, Benoit Combemale, Jean-Marc Jézéquel, Gurvan Leguernic, Noel Plouzeau, Didier Vojtisek.

### 7.2.1 Foundations

**A Hitchhiker's Guide to Model-Driven Engineering for Data-Centric Systems.** A broad spectrum of application domains are increasingly making use of heterogeneous and large volumes of data with varying degrees of humans in the loop. The recent success of Artificial Intelligence (AI) and, in particular, Machine Learning (ML) further amplifies the relevance of data in the development, maintenance, evolution, and execution management of systems built with model-driven engineering techniques. Applications

include critical infrastructure areas such as intelligent transportation, smart energy management, public healthcare, and emergency and disaster management; many of these systems are considered socio-technical systems given the human, social, and organizational factors that must be considered during the system life-cycle. In [27], we introduce a conceptual reference framework – the *Models and Data* (MODA) framework – to support a data-centric and model-driven approach for the integration of heterogeneous models and their respective data for the entire life-cycle of socio-technical systems.

**Software Language Extension Problem** The problem of software language extension and composition drives much of the research in *Software Language Engineering* (SLE). Although various solutions have already been proposed, there is still little understanding of the specific ins and outs of this problem, which hinders the comparison and evaluation of existing solutions. In [33], we introduce the Language Extension Problem as a way to better qualify the scope of the challenges related to language extension and composition. The formulation of the problem is similar to the seminal Expression Problem introduced by Wadler in the late nineties, and lift it from the extensibility of single constructs to the extensibility of groups of constructs, i.e., software languages. We provide a comprehensive definition of the actual constraints when considering language extension, and believe the Language Extension Problem will drive future research in SLE, the same way the original Expression Problem helped to understand the strengths and weaknesses of programming languages and drove much research in programming languages.

**Comparing and Classifying Model Transformation Reuse Approaches across Metamodels** Model transformations are essential elements of model-driven engineering (MDE) solutions, as they enable the automatic manipulation of models. MDE promotes the creation of domain-specific metamodels, but without proper reuse mechanisms, model transformations need to be developed from scratch for each new metamodel. In this work [25], our goal is to understand whether transformation reuse across metamodels is needed by the community, evaluate its current state, identify practical needs and propose promising lines for further research. For this purpose, we first report on a survey to understand the reuse approaches used currently in practice and the needs of the community. Then, we propose a classification of reuse techniques based on a feature model and compare a sample of specific approaches—model types, concepts, *a posteriori* typing, multilevel modeling, typing requirement models, facet-oriented modeling, mapping operators, constraint-based model types, and design patterns for model transformations—based on this feature model and a common example. We discuss strengths and weaknesses of each approach, provide a reading grid used to compare their features, compare with community needs, identify gaps in current transformation reuse approaches in relation to these needs and propose future research directions.

**Modular and Distributed IDE** Integrated Development Environments (IDEs) are indispensable companions to programming languages. They are increasingly turning towards Web-based infrastructure. The rise of a protocol such as the Language Server Protocol (LSP) that standardizes the separation between a language-agnostic IDE, and a language server that provides all language services (e.g., auto completion, compiler...) has allowed the emergence of high quality generic Web components to build the IDE part that runs in the browser. However, all language services require different computing capacities and response times to guarantee a user-friendly experience within the IDE. The monolithic distribution of all language services prevents to leverage on the available execution platforms (e.g., local platform, application server, cloud). In contrast with the current approaches that provide IDEs in the form of a monolithic client-server architecture, we explore in [45] the modularization of all language services to support their individual deployment and dynamic adaptation within an IDE. We evaluate the performance impact of the distribution of the language services across the available execution platforms on four EMF-based languages, and demonstrate the benefit of a custom distribution.

**A Principled Approach to REPL Interpreters** Read-eval-print-loops (REPLs) allow programmers to test out snippets of code, explore APIs, or even incrementally construct code, and get immediate feedback on their actions. However, even though many languages provide a REPL, the relation between the language as is and what is accepted at the REPL prompt is not always well-defined. Furthermore, implementing a REPL for new languages, such as DSLs, may incur significant language engineering cost. In [59] we survey

the domain of REPLs and investigate the (formal) principles underlying REPLs. We identify and define the class of sequential languages, which admit a sound REPL implementation based on a definitional interpreter, and present design guidelines for extending existing language implementations to support REPL-style interfaces (including computational notebooks). The obtained REPLs can then be generically turned into an exploring interpreter, to allow exploration of the user's interaction. The approach is illustrated using three case studies, based on MiniJava, QL (a DSL for questionnaires), and eFLINT (a DSL for normative rules). We expect sequential languages, and the consequent design principles, to be stepping stones towards a better understanding of the essence of REPLs.

**Behavioral Interfaces for Executable DSLs** Executable domain-specific languages (DSLs) enable the execution of behavioral models. While an execution is mostly driven by the model content (e.g., control structures), many use cases require interacting with the running model, such as simulating scenarios in an automated or interactive way, or coupling the model with other models of the system or environment. The management of these interactions is usually hardcoded into the semantics of the DSL, which prevents its reuse for other DSLs and the provision of generic interaction-centric tools (e.g., event injector). In [35], we propose a metalanguage for complementing the definition of executable DSLs with explicit behavioral interfaces to enable external tools to interact with executed models in a unified way. We implemented the proposed metalanguage in the GEMOC Studio and show how behavioral interfaces enable the realization of tools that are generic and thus usable for different executable DSLs.

**Runtime Monitoring for Executable DSLs** Runtime monitoring is a fundamental technique used throughout the lifecycle of a system for many purposes, such as debugging, testing, or live analytics. While runtime monitoring for general purpose programming languages has seen a great amount of research, developing such complex facilities for any executable Domain Specific Language (DSL) remains a challenging, reoccurring and error prone task. A generic solution must both support a wide range of executable DSLs (xDSLs) and induce as little execution time overhead as possible. In [36], our contribution is a fully generic approach based on a temporal property language with a semantics tailored for runtime verification. Properties can be compiled into efficient runtime monitors that can be attached to any kind of executable discrete event model within an integrated development environment. Efficiency is bolstered using a novel combination of structural model queries and complex event processing. Our evaluation on three xDSLs shows that the approach is applicable with an execution time overhead of 121% (on executions shorter than 1s), to 79% (on executions shorter than 20s) making it suitable for model testing and debugging.

**Live Modeling** Live modeling has been recognized as an important technique to edit behavioral models while being executed and helps in better understanding the impact of a design choice. In the context of Model-driven Development (MDD) models can be executed by interpretation or by the translation of models into existing programming languages, often by code generation. This work is concerned with the support of live modeling in the context of state machine models when they are executed by code generation [24]. To this end, we propose an approach that is completely independent of any live programming support offered by the target language. This independence is achieved with the help of a model transformation that equips the model with support for features that are required for live modeling. A subsequent code generation then produces a self-reflective program that allows changes to the model elements at runtime (through synchronization of design and runtime models). We have applied the approach in the context of UML-RT and created a prototype (Live-UMLRT) that provides a full set of services for live modeling of UML-RT state machines such as re-execution, adding/removing states and transitions, and adding/removing action code. We have evaluated the prototype on several use cases. The evaluation shows that (1) generation of a self-reflective and model instrumentation can be carried out with reasonable performance, and (2) our approach can apply model changes to the running execution faster than the standard approach that depends on the live programming support of the target language.

**Automatic Generation of Truffle-based Interpreters for Domain-Specific Languages** Numerous language workbenches have been proposed over the past decade to ease the definition of Domain-Specific Languages (DSLs). Language workbenches enable language designers to specify DSLs using high-level

metalanguages and to generate their implementation (e.g. parsers, interpreters) and tool support (e.g. editors, debuggers) automatically. However, little attention has been given to the performance of the resulting interpreters. In many domains where performance is key (e.g. scientific and high-performance computing), this forces language designer to handcraft ad hoc optimizations in the interpreter implementations, or to lose compatibility with tool support. In [34] we propose to systematically exploit the domain-specific information of language specifications to derive optimized Truffle-based language interpreters executed over the GraalVM. We implement our approach on top of the Eclipse Modeling Framework (EMF) by complementing its existing compilation chain with Truffle-specific information, which drives the GraalVM to benefit from an optimized just-in-time compilation. A key benefit of our approach is that it leverages existing language specifications and does not require additional information from language designers who remain oblivious of Truffle's low-level intricacies and JIT optimizations in general, while staying compatible with tool support. We evaluate our approach using a representative set of four DSLs and eight conforming programs. Compared to the standard interpreters generated by EMF running on the GraalVM, we observe an average speed-up of x1.14, ranging from x1.07 to x1.26. Although the benefits vary slightly from one DSL or program to another, we conclude that our approach yields substantial performance gains while remaining non-intrusive of EMF abstractions.

**Loop Aggregation for Approximate Scientific Computing** Trading off some accuracy for better performance in scientific computing is an appealing approach to ease the exploration of various alternatives on complex simulation models. Existing approaches involve the application of either time-consuming model reduction techniques or resource-demanding statistical approaches. Such requirements prevent any opportunistic model exploration, e.g., exploring various scenarios on environmental models. This limits the ability to analyse new models for scientists, to support trade-off analysis for decision-makers and to empower the general public towards informed environmental intelligence. In [58], we present a new approximate computing technique, aka. loop aggregation, which consists in automatically reducing the main loop of a simulation model by aggregating the corresponding spatial or temporal data. We apply this approximate scientific computing approach on a geophysical model of a hydraulic simulation with various input data. The experimentation demonstrates the ability to drastically decrease the simulation time while preserving acceptable results with a minimal set-up. We obtain a median speed-up of 95.13% and up to 99.78% across all the 23 case studies.

### 7.2.2 Applications

**Toward Model-driven Sustainability Evaluation** Sustainability has emerged as a concern of central relevance. As a wicked problem, it poses challenges to business-as-usual in many areas, including that of modeling. In [30], we address a question at the intersection of model-driven engineering and sustainability research: *How can we better support sustainability by bringing together model-driven engineering, data, visualization and self-adaptive systems, to facilitate engagement, exploration, and understanding of the effects that individual and organizational choices have on sustainability?* We explore this question via an idealized vision of an evaluation environment that facilitates integration and mapping of models from multiple diverse sources, visual exploration, and evaluation of what-if scenarios, for stakeholders with divergent perspectives. The article identifies research challenges to be addressed to enable decision making to support sustainability and provides a map of sustainability modeling issues across disciplines.

**Modeling Languages in Industry 4.0: an Extended Systematic Mapping Study** Industry 4.0 integrates cyber-physical systems with the Internet of Things to optimize the complete value-added chain. Successfully applying Industry 4.0 requires the cooperation of various stakeholders from different domains. Domain-specific modeling languages promise to facilitate their involvement through leveraging (domain-specific) models to primary development artifacts. We aim to assess the use of modeling in Industry 4.0 through the lens of modeling languages in a broad sense. Based on an extensive literature review, we updated our systematic mapping study on modeling languages and modeling techniques used in Industry 4.0 (Wortmann et al., Conference on model-driven engineering languages and systems (MODELS'17), IEEE, pp 281–291, 2017) to include publications until February 2018 [41]. Overall, the updated study considers 3344 candidate publications that were systematically investigated until 408 relevant

publications were identified. Based on these, we developed an updated map of the research landscape on modeling languages and techniques for Industry 4.0. Research on modeling languages in Industry 4.0 focuses on contributing methods to solve the challenges of digital representation and integration. To this end, languages from systems engineering and knowledge representation are applied most often but rarely combined. There also is a gap between the communities researching and applying modeling languages for Industry 4.0 that originates from different perspectives on modeling and related standards. From the vantage point of modeling, Industry 4.0 is the combination of systems engineering, with cyber-physical systems, and knowledge engineering. Research is currently splintered along topics and communities and accelerating progress demands for multi-disciplinary, integrated research efforts.

**Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges** Digital Twins have emerged since the beginning of this millennium to better support the management of systems based on (real-time) data collected in different parts of the operating systems. Digital Twins have been successfully used in many application domains, and thus, are considered as an important aspect of Model-Based Systems Engineering (MBSE). However, their development, maintenance, and evolution still face major challenges, in particular: (i) the management of heterogeneous models from different disciplines, (ii) the bidirectional synchronization of digital twins and the actual systems, and (iii) the support for collaborative development throughout the complete life cycle. In the last decades, the Model-Driven Engineering (MDE) community has investigated these challenges in the context of software systems. Now the question arises, which results may be applicable for digital twin engineering as well. In [44], we identify various MDE techniques and technologies that may contribute to tackle the three mentioned digital twin challenges as well as outline a set of open MDE research challenges that need to be addressed in order to move towards a digital twin engineering discipline.

**Opportunities in Intelligent Modeling Assistance** Modeling is requiring increasingly larger efforts while becoming indispensable given the complexity of the problems we are solving. Modelers face high cognitive load to understand a multitude of complex abstractions and their relationships. There is an urgent need to better support tool builders to ultimately provide modelers with intelligent modeling assistance that learns from previous modeling experiences, automatically derives modeling knowledge, and provides context-aware assistance. However, current intelligent modeling assistants (IMAs) lack adaptability and flexibility for tool builders, and do not facilitate understanding the differences and commonalities of IMAs for modelers. Such a patchwork of limited IMAs is a lost opportunity to provide modelers with better support for the creative and rigorous aspects of software engineering. In [37, 54], we present a conceptual reference framework (RF-IMA) and its properties to identify the foundations for intelligent modeling assistance. For tool builders, RF-IMA aims to help build IMAs more systematically. For modelers, RF-IMA aims at facilitating comprehension, comparison, and integration of IMAs, and ultimately at providing more intelligent support. We envision a momentum in the modeling community that leads to the implementation of RF-IMA and consequently future IMAs. We identify open challenges that need to be addressed to realize the opportunities provided by intelligent modeling assistance.

## 7.3 Results on Heterogeneous and dynamic software architectures

**Participants** Olivier Barais, Arnaud Blouin, Johann Bourcier, Stéphanie Challita, Benoit Combemale, Djamel khelladi.

We have selected two main group of contributions for DIVERSE's research axis #3: one is in the field of architecture modeling in particular for the cloud computing field and a second one in the domain of runtime management of resources for dynamically adaptive system.

### 7.3.1 Software architecture and cloud modeling

**Model-Based Cloud Resource Management with TOSCA and OCCI** With the advent of cloud computing, different cloud providers with heterogeneous cloud services (compute, storage, network, applications,

etc.) and their related Application Programming Interfaces (APIs) have emerged. This heterogeneity complicates the implementation of an interoperable cloud system. Several standards have been proposed to address this challenge and provide a unified interface to cloud resources. The Open Cloud Computing Interface (OCCI) thereby focuses on the standardization of a common API for Infrastructure-as-a-Service (IaaS) providers while the Topology and Orchestration Specification for Cloud Applications (TOSCA) focuses on the standardization of a template language to enable the proper definition of the topology of cloud applications and their orchestrations on top of a cloud system. TOSCA thereby does not define how the application topologies are created on the cloud. Therefore, we analyse the conceptual similarities between the two approaches and we study how we can integrate them to obtain a complete standard-based approach to manage both cloud infrastructure and cloud application layers [26]. We propose an automated extensive mapping between the concepts of the two standards and we provide TOSCA Studio, a model-driven tool chain for TOSCA that conforms to OCCI. TOSCA Studio allows to graphically design cloud applications as well as to deploy and manage them at runtime using a fully model-driven cloud orchestrator based on the two standards. Our contribution is validated by successfully transforming and deploying three cloud applications: WordPress, Node Cellar and Multi-Tier.

### 7.3.2 Leveraging unused heterogeneous resources for modular applications with SLA guarantees

**Leveraging Cloud Unused Heterogeneous Resources for Applications with SLA Guarantees** Efficient resource management is an important dimension in the field of cloud computing for both economic and ecological reasons. It has been observed that the resources of cloud infrastructures are only used to an average of 20%. In order to improve its business model, a cloud provider must seek to optimise the use of all its hardware resources without ever violating the minimum quality of service it has contracted with its customers. The objective of this work is to exploit the unused heterogeneous resources of the cloud for applications with guaranteed quality of service [61]. For this purpose, the work provides four main contributions. The first one focuses on the estimation of the real capacity of a virtualized machine using SSD storage devices, taking into account the variable performance caused by interferences. The second aims at estimating the future unused resources of a cloud infrastructure and anticipating the risks of impact on the quality of service. A third contribution demonstrates the possibility of efficiently exploiting unused cloud resources for large data applications without disrupting resource providers' applications. Finally, a final contribution proposes to verify the proper execution of an application in a non-trusted environment.

**ReLeaSER: A Reinforcement Learning Strategy for Optimizing Utilization Of Ephemeral Cloud Resources** Cloud data center capacities are over-provisioned to handle demand peaks and hardware failures, which leads to poor resource utilization. One way to improve resource utilization and thus reduce the total cost of ownership is to offer unused resources (referred to as ephemeral resources) at a lower price. However, reselling resources needs to meet the expectations of its customers in terms of quality of service. The goal is thus to maximize the amount of reclaimed resources while avoiding SLA penalties. To achieve that goal, cloud providers have to estimate their future utilization to provide availability guarantees. The prediction should consider a safety margin for resources to react to unpredictable workloads. The challenge is to find the safety margin that provides the best trade-off between the amount of resources to reclaim and the risk of SLA violations. Most state-of-the-art solutions rely on a fixed safety margin for all types of metrics (e.g., CPU, RAM). However, a single fixed margin does not consider various workloads variations over time, which may lead to SLA violations or/and poor utilization. In order to tackle these challenges, we propose ReLeaSER [46], a Reinforcement Learning strategy for optimizing the ephemeral resource utilization in the cloud. ReLeaSER dynamically tunes the safety margin at host level for each resource metric. The strategy learns from past prediction errors (that caused SLA violations). Our solution reduces significantly the SLA violation penalties on average by 2.7x and up to 3.4x. It also improves considerably the cloud providers potential savings by 27.6% on average and up to 43.6%.

**Salamander: a Holistic Scheduling of MapReduce Jobs on Ephemeral Cloud Resources** Most cloud data centers are overprovisioned and underutilized, primarily to handle peak loads and sudden failures. This has motivated many researchers to reclaim the unused resources, which are by nature ephemeral, to run data-intensive applications at a lower cost. Hadoop MapReduce is one of those applications. However,

it was designed on the assumption that resources are available as long as users pay for the service. In order to make it possible for Hadoop to run on unused (ephemeral) resources, we have designed a heterogeneity- and volatility-aware holistic scheduler [47], consisting of three different components: (1) a MapReduce task and job scheduler that relies on a global vision of resource utilization predictions, (2) a scheduler-based data placement strategy that improves the data locality, and (3) a reactive QoS controller that ensures customers' service-level agreement (SLA) and minimizes interference between co-located workloads. Our framework makes it possible to take advantage of ephemeral resources efficiently. Indeed, for a given set of jobs, it reduces the overall execution time by up to 47.6% and an average of 18.7% as compared to state-of-the-art strategies.

## 7.4 Results on Diverse Implementations for Resilience

**Participants** Olivier Barais, Benoit Baudry, Arnaud Blouin, Johann Bourcier, Stéphanie Challita, Benoit Combemale, Jean-Marc Jézéquel, Djamel khelladi, Olivier Zendra.

Diversity is acknowledged as a crucial element for resilience, sustainability and increased wealth in many domains such as sociology, economy and ecology. Yet, despite the large body of theoretical and experimental science that emphasizes the need to conserve high levels of diversity in complex systems, the limited amount of diversity in software-intensive systems is a major issue. This is particularly critical as these systems integrate multiple concerns, are connected to the physical world, run eternally and are open to other services and to users. Here we present our latest observational and technical results about (i) observations of software evolution and co-evolution, (ii) observations of software diversity related to software security and privacy and (iii) software verification (testing) to study and assess the validity of software.

### 7.4.1 Software Co-evolution

**Co-Evolving Code with Evolving Metamodels** Metamodels play a significant role to describe and analyze the relations between domain concepts. They are also cornerstones to build a software language (SL) for a given domain and its associated tooling. Metamodel definition generally drives code generation of a core API. The latter is further enriched by developers with additional code implementing advanced functionalities, e.g. checkers, recommenders, etc. When a SL evolves to its next version, its metamodels evolve as well before regeneration of the core API code. As a result, the developers added code both in the core API and the SL toolings may be impacted and thus may need to be co-evolved accordingly. Many approaches support the co-evolution of various artifacts when metamodels evolve. However, not the co-evolution of code. Our work [50] fills this gap. We propose a semi-automatic co-evolution approach based on change propagation. The premise is that knowledge of the metamodel evolution changes can be propagated by means of resolutions to drive the code co-evolution. Our approach leverages on the abstraction level of metamodels where a given metamodel element has often different usages in the code. It supports alternative co-evaluations to meet different developers needs. Our work is evaluated on three Eclipse SL implementations, namely OCL, Modisco, and Papyrus over several evolved versions of metamodels and code. In response to five different evolved metamodels, we co-evolved 976 impacts over 18 projects. A comparison of our co-evolved code with the versioned ones shows the usefulness of our approach. Our approach was able to reach a weighted average of 87.4% and 88.9% respectively of precision and recall while supporting useful alternative co-evolution that developers have manually performed.

**On the Power of Abstraction: a Model-Driven Co-evolution Approach of Software Code** Model-driven software engineering fosters abstraction through the use of models and then automation by transforming them into various artefacts, in particular to code, for example: 1) from architectural models to code, 2) from metamodels to API code (with EMF in Eclipse), 3) from entity models to front-end and back-end code in Web stack application (with JHispter), etc. In all these examples, the generated code is usually enriched by developers with additional code implementing advanced functionalities (e.g. checkers,

recommenders, etc.) to build a full coherent system. When the system must evolve, so must the models to regenerate the code. As a result, the developers' enriched code may be impacted and thus need to co-evolve accordingly. Many approaches support the co-evolution of various artifacts, but not the co-evolution of code. Our work [49] sheds light on this issue and envisions to fill this gap. We formulate the hypothesis that code co-evolution can be driven by model changes using change propagation. To investigate this hypothesis, we implemented a prototype for the case of metamodels and their accompanying code in EMF Eclipse. As a preliminary evaluation, we considered the case of the OCL Pivot metamodel evolution and its code co-evolution in two projects from version 3.2.2 to 3.4.4. Preliminary results confirm our hypothesis that model-driven evolution changes can effectively drive the code co-evolution. On 562 impacts in two projects' code by 221 metamodel changes, our approach was able to reach the average of 89% and 92,5% of precision and recall respectively.

**Consistent Change Propagation within Models.** Developers change models with clear intentions, e.g. for refactoring, defects removal, or evolution. However, in doing so developers are often unaware of the consequences of their changes. Changes to one part of a model may affect other parts of the same model and/or even other models, possibly created and maintained by other developers. The consequences are incomplete changes, and with it inconsistencies within or across models. Extensive works exist on detecting and repairing inconsistencies. However, literature tends to focus on inconsistencies as errors in need of repairs rather than on incomplete changes in need of further propagation. Many changes are non-trivial and require a series of coordinated model changes. As developers start changing the model, intermittent inconsistencies arise with other parts of the model that developers have not yet changed. These inconsistencies are cues for incomplete change propagation. Resolving these inconsistencies should be done in a manner that is consistent with the original changes. We define this property as consistent change propagation. Paper [32] leverages classical inconsistency repair mechanisms to explore the vast search space of change propagation. Our approach not only suggests changes to repair a given inconsistency but also changes to repair inconsistencies caused by the aforementioned repair. In doing so, our approach follows the developer's intent where subsequent changes may not contradict or backtrack earlier changes. We argue that consistent change propagation is essential for effective model driven engineering. Our approach and its tool implementation were empirically assessed on 18 case studies from industry, academia, and GitHub to demonstrate its feasibility and scalability. A comparison with two versioned models shows that our approach identifies actual repair sequences that developers had chosen. Furthermore, an experiment involving 22 participants shows that our change propagation approach meets the workflow of how developers handle changes by always computing the sequence of repairs resulting from the change propagation.

**Transforming Abstract to Concrete Repairs with a Generative Approach of Repair Values** Software models, often comprised of interconnected diagrams, change continuously, and developers often fail to keep these diagrams consistent. Detecting inconsistencies quickly and efficiently is state of the art. However, repairing them is not trivial, because there are typically multiple model elements that need to be repaired, leading to an exponentially growing space of combinations of repair choices. Despite extensive research on consistency checking, existing approaches provide abstract repairs only (i.e. identifying the model element but failing to describe the change), which is not satisfactory. Our work proposes a novel approach that provides concrete repair choices based on values from the inconsistent models [31]. More precisely, our approach first retrieves repair values from the model, turns them into repair choices, and groups them based on their effects. This grouping lets our approach explore the repair space in its entirety, providing quick example-like feedback for all possible repairs. Our approach and its tool implementation have been empirically assessed on 10 case studies from industry, academia, and GitHub to demonstrate its feasibility and scalability. A comparison with three versioned models shows that our approach identifies useful repair values that developers have chosen.

**An Approach and Benchmark to Detect Behavioral Changes of Commits in Continuous Integration** When a developer pushes a change to an application's codebase, a good practice is to have a test case specifying this behavioral change. Thanks to continuous integration (CI), the test is run on subsequent commits to check that they do no introduce a regression for that behavior. In our work [28], we propose

an approach that detects behavioral changes in commits. Its inputs are a program, its test suite, and a commit. Its output is a set of test methods that captures the behavioral difference between the pre-commit and post-commit versions of the program. We call our approach DCI (Detecting behavioral changes in CI). It relies on the generation of variations of the existing test cases through (i) assertion amplification and (ii) a search-based exploration of the input space. We evaluate our approach on a curated set of 60 commits from 6 open source Java projects. To our knowledge, this is the first ever curated dataset of real-world behavioral changes. Our evaluation shows that DCI is able to generate test methods that detect behavioral changes. Our approach is fully automated and can be integrated into current development processes.

### 7.4.2 Privacy and Security

**SE-PAC: A Self-Evolving PAcker Classifier against rapid packers evolution** Packers are widespread tools used by malware authors to hinder static malware detection and analysis. Identifying the packer used to pack a malware is essential to properly unpack and analyze the malware, be it manually or automatically. While many well-known packers are used, there is a growing trend for new custom packers that make malware analysis and detection harder. Research works have been very effective in identifying known packers or their variants, with signature-based, supervised machine learning or similarity-based techniques. However, identifying new packer classes remains an open problem. This work presents a self-evolving packer classifier that provides an effective, incremental, and robust solution to cope with the rapid evolution of packers [55]. We propose a composite pairwise distance metric combining different types of packer features. We derive an incremental clustering approach able to identify both (variants of) known packer classes and new ones, as well as to update clusters automatically and efficiently. Our system thus continuously enhances, integrates, adapts and evolves packer knowledge. Moreover, to optimize post clustering packer processing costs, we introduce a new post clustering strategy for selecting small subsets of relevant samples from the clusters. Our approach effectiveness and time-resilience are assessed with: 1) a real-world malware feed dataset composed of 16k packed binaries, comprising 29 unique packers, and 2) a synthetic dataset composed of 19k manually crafted packed binaries, comprising 31 unique packers (including custom ones).

**An Asset-Based Assistance for Secure by Design** With the growing numbers of security attacks causing more and more serious damages in software systems, security cannot be added as an afterthought in software development. It has to be built in from the early development phases such as requirement and design. The role responsible for designing a software system is termed an "architect", knowledgeable about the system architecture design, but not always well-trained in security. Moreover, involving other security experts into the system design is not always possible due to time-to-market and budget constraints. To address these challenges, we propose to define an asset-based security assistance [52], to help architects design secure systems even if these architects have limited knowledge in security. This assistance helps to detect threats, and integrates security controls over vulnerable parts of system into the architecture model. The central concept enabling this assistance is an *asset*. We apply our proposal to a telemonitoring case study to show that automating such an assistance is feasible.

**Asset-Oriented Threat Modeling** Threat modeling is recognized as one of the most important activities in software security. It helps to address security issues in software development. Several threat modeling processes are widely used in the industry such as Microsoft's SDL. In threat modeling, it is essential to first identify assets before enumerating threats, in order to diagnose the threat targets and spot the protection mechanisms. Asset identification and threat enumeration are collaborative activities involving many actors such as security experts and software architects. These activities are traditionally carried out in brainstorming sessions. Due to the lack of guidance, the lack of a sufficiently formalized process, the high dependence on actors' knowledge, and the variety of actors' background, these actors often have difficulties collaborating with each other. Brainstorming sessions are thus often conducted sub-optimally and require significant effort. To address this problem, we aim at structuring the asset identification phase by proposing a systematic asset identification process, which is based on a reference model [53]. This process structures and identifies relevant assets, facilitating the threat enumeration during brainstorming.

We illustrate the proposed process with a case study and show the usefulness of our process in supporting threat enumeration and improving existing threat modeling processes such as the Microsoft SDL one.

**Browser Profiling: State of the Art and Countermeasures** This work presents the result of a litterature review of all the techniques used to track a user navigating on the Internet though a web browser and the main countermeasures available [60].

### 7.4.3 Software Verification

**A Language Agnostic Approach to Modeling Requirements: Specification and Verification** Modeling is a complex and error prone activity, which can result in ambiguous models containing omissions and inconsistencies. Many works have addressed the problem of checking models' consistency. However, most of these works express consistency requirements for a specific modeling language. On the contrary, we argue that in some contexts those requirements should be expressed independently from the modeling language of the models to be checked. In our work [42], we identify a set of modeling requirements in the context of embedded systems design that are expressed independently from any modeling language concrete syntax. We propose a dedicated semantic domain to support them and give a formal characterization of those requirements that is modeling language agnostic.

**Misconfiguration Discovery with Principal Component Analysis for Cloud-Native Services** Native cloud applications and services have significantly increased the importance of system and service configuration activities. These activities include updating (i) these services, (ii) their dependencies on third parties, (iii) their configurations, (iv) the configuration of the execution environment, (v) network configurations. The high frequency of updates results in significant configuration complexity that can lead to failures or performance drops. To mitigate these risks, service providers extensively rely on testing techniques, such as metamorphic testing, to detect these failures before moving to production. However, the development and maintenance of these tests are costly, especially the oracle, which must determine whether a system's performance remains within acceptable boundaries. In [57], we explores the use of a learning method called Principal Component Analysis (PCA) to learn about acceptable performance metrics on cloud-native services and identify a metamorphic relationship between the nominal service behavior and the value of these metrics.We investigate the following research question: Is it possible to combine the metamorphic testing technique with unsupervised learning methods on service monitoring data to detect error-prone reconfigurations before moving to production? We remove the developers' burden to define a specific oracle in detecting these configuration issues. For validation, we applied this proposal on a distributed media streaming application whose authentication was managed by an external identity and access management services. This application illustrates both the heterogeneity of the technologies used to build this type of service and its large configuration space. As a result, our proposal demonstrated the ability to identify error-prone reconfigurations using PCA.

# 8 Bilateral contracts and grants with industry

## 8.1 Bilateral contracts with industry

**ADR Nokia**

- Coordinator: Inria
- Dates: 2017-2021
- Abstract: The goal of this project is to integrate a chaos engineering principles to IoT Services frameworks to improve the robustness of the software-defined network services using this approach and to explore the concept of equivalence for software-defined network services and propose an approach to constantly evolve the attack surface of the network services.

**BCOM**

- Coordinator: UR1
- Dates: 2018-2024
- Abstract: The aim of the Falcon project is to investigate how to improve the resale of available resources in private clouds to third parties. In this context, the collaboration with DiverSE mainly aims at working on efficient techniques for the design of consumption models and resource consumption forecasting models. These models are then used as a knowledge base in a classical autonomous loop.

**GLOSE**

- Partners: Inria/CNRS/Safran
- Dates: 2017-2021
- Abstract: The GLOSE project develops new techniques for heterogeneous modeling and simulation in the context of systems engineering. It aims to provide formal and operational tools and methods to formalize the behavioral semantics of the various modeling languages used at system-level. These semantics will be used to extract behavioral language interfaces supporting the definition of coordination patterns. These patterns, in turn, can systematically be used to drive the coordination of any model conforming to these languages. The project is structured according to the following tasks: concurrent xDSML engineering, coordination of discrete models, and coordination of discrete/continuous models. The project is funded in the context of the network DESIR, and supported by the GEMOC initiative.

**GLOSE Demonstrator**

- Partners: Inria/Safran
- Dates: 2019-2020
- Abstract: Demonstrator illustrating the technologies involved in the WP5 off the GLOSE project. The use case chosen for the demonstrator is the high-level description of a remote control drone system, whose the main objective is to illustrate the design and simulation of the main functional chains, the possible interactivity with the model in order to raise the level of understanding over the models built, and possibly the exploration of the design space.

**Debug4Science**

- Partners: Inria/CEA DAM
- Dates: 2020-2022
- Abstract: Debug4Science aims to propose a disciplined approach to develop domain-specific debugging facilities for Domain-Specific Languages within the context of scientific computing and numerical analysis. Debug4Science is a bilateral collaboration (2020-2022), between the CEA DAM/DIF and the DiverSE team at Inria.

**Orange**

- Partners: UR1/Orange
- Dates: 2020-2023
- Abstract: Context aware adaptive authentification, Anne Bumiller's PhD Cifre project.

**Obeo**

- Partners: Inria/Obéo
- Dates: 2017-2020
- Abstract: Web engineering for domain-specific modeling languages, Fabien Coulon's PhD Cifre project.

**OKWind**

- Partners: UR1/OKWind
- Dates: 2017-2020
- Abstract: Models@runtime to improve self-consumption of renewable energies, Alexandre Rio's PhD Cifre project.

**Keolis**

- Partners: UR1/Keolis
- Dates: 2018-2021
- Abstract: Urban mobility: machine learning for building simulators using large amounts of data, Gauthier LYAN's PhD Cifre project.

**FaberNovel**

- Partners: UR1/FaberNovel
- Dates: 2018-2021
- Abstract: Abstractions for linked data and the programmable web, Antoine Cheron's PhD Cifre project.

# 9 Partnerships and cooperations

## 9.1 International initiatives

### 9.1.1 Inria International Labs

**ALE**

**Title:** *Agile Language Engineering*

**Duration:** 2020 - 2022

**Coordinator:** Benoit Combemale

**Partners:**

- *Inria (France)*, CWI (Netherlands)

**Inria contact:** Benoit Combemale

**Summary:** Software engineering faces new challenges with the advent of modern software-intensive systems such as complex critical embedded systems, cyber-physical systems and the Internet of things. Application domains range from robotics, transportation systems, defense to home automation, smart cities, and energy management, among others. Software is more and more pervasive, integrated into large and distributed systems, and dynamically adaptable in response to a complex and open environment. As a major consequence, the engineering of such systems involves multiple stakeholders, each with some form of domain-specific knowledge, and with the increased use of software as an integration layer. Hence more and more organizations are adopting Domain-Specific Languages (DSLs) to allow domain experts to express solutions directly in terms of relevant domain concepts. This new trend raises new challenges about designing DSLs, evolving a set of DSLs and coordinating the use of multiple DSLs for both DSL designers and DSL users. ALE will contribute to the field of Software Language Engineering, aiming to provide more agility to both language designers and language users. The main objective is twofold. First, we aim to help language designers to leverage previous DSL implementation efforts by reusing and

combining existing language modules, while automating the deployment of distributed, elastic and collaborative modeling environments. Second, we aim to provide more flexibility to language users by ensuring interoperability between different DSLs, offering live feedback about how the model or program behaves while it is being edited (aka. live programming/modeling), and combining with interactive environments like Jupiter Notebook for literate programming.

## 9.2 International research visitors

### 9.2.1 Visits of international scientists

- Nelly Bencomo, Aston University, UK
- Gunter Mussbacher, McGill University, Canada

## 9.3 European initiatives

### 9.3.1 Collaborations with major European organizations

- **Vipo Project**. Vipo is an innovation project from EIT Digital. This year, we bring our expertise on native cloud architecture and adaptable architectures for this project.

## 9.4 National initiatives

### 9.4.1 ANR

**VaryVary ANR JCJC**

- Coordinator: Mathieu Acher
- DiverSE, Inria/IRISA Rennes
- Dates: 2017-2021
- Abstract: Most modern software systems (operating systems like Linux, Web browsers like Firefox or Chrome, video encoders like x264 or ffmpeg, servers, mobile applications, etc.) are subject to variation or come in many variants. Hundreds of configuration options, features, or plugins can be combined, each potentially with distinct functionality and effects on execution time, memory footprint, etc. Among configurations, some of them are chosen and do not compile, crash at run time, do not pass a test suite, or do not reach a certain performance quality (e.g., energy consumption, security). In this JCJC ANR project, we follow a thought-provocative and unexplored direction: We consider that the variability boundary of a software system can be specialized and should vary when needs be. The goal of this project is to provide theories, methods and techniques to make vary variability. Specifically, we consider machine learning and software engineering techniques for narrowing the space of possible configurations to a good approximation of those satisfying the needs of users. Based on an oracle (e.g., a runtime test) that tells us whether a given configuration meets the requirements (e.g., speed or memory footprint), we leverage machine learning to retrofit the acquired constraints into a variability that can be used to automatically specialize the configurable system. Based on a relative small number of configuration samples, we expect to reach high accuracy for many different kinds of oracles and subject systems. Our preliminary experiments suggest that varying variability can be practically useful and effective. However, much more work is needed to investigate sampling, testing, and learning techniques within a variety of cases and application scenarios. We plan to further collect large experimental data and apply our techniques on popular, open-source, configurable software (like Linux, Firefox, ffmpeg, VLC, Apache or JHipster) and generators for media content (like videos, models for 3D printing, or technical papers written in LaTeX).

#### 9.4.2 DGA

**LangComponent (CYBERDEFENSE)**

- Coordinator: DGA
- Partners: DGA MI, INRIA
- Dates: 2019-2022
- Abstract: in the context of this project, DGA-MI and the INRIA team DiverSE explore the existing approaches to ease the development of formal specifications of domain-Specific Languages (DSLs) dedicated to paquet filtering, while guaranteeing expressiveness, precision and safety. In the long term, this work is part of the trend to provide to DGA-MI and its partners a tooling to design and develop formal DSLs which ease the use while ensuring a high level of reasoning.

# 10 Dissemination

## 10.1 Promoting scientific activities

### 10.1.1 Scientific events: organisation

**General chair, scientific chair** We organize ICPE 2021. Johann Bourcier will serve as general chair.

**Member of the organizing committees** Arnaud Blouin:

- JDev 2020

Stéphanie Challita:

- Proceedings chair, ICSA 2021
- Publicity co-chair, ECSA 2020

Djamel Khelladi:

- Publicity co-chair, ICPE 2021

**Chair of conference program committees** Mathieu Acher:

- Program committee co-chair, VaMoS 2020
- Workshop co-chair MODELS 2020
- Organizer/co-chairs of REVE and MODEVAR workshops

Stéphanie Challita:

- Co-organizer/program committee co-chair, FAACS workshop @ECSA 2020

Benoit Combemale:

- PC co-chair for ICT4S 2020
- Tools & Demos co-chair for MODELS'20

Djamel Khelladi:

- Poster Track co-chair, MODELS 2020
- Doctoral Track co-chair, ICSSP/ICGSE 2021

**Member of the conference program committees** Arnaud Blouin:

- ACM/SIGAPP Symposium on Applied Computing (SAC), software engineering track, 2020
- ACM SIGCHI symposium on Engineering interactive computing systems (EICS 2020), 2020

Mathieu Acher:

- SPLC 2020
- VaMoS 2020
- DocSymp@MODELS2020
- ICSE SEIP 2020

Jean-Marc Jézéquel:

- ICSE 2020
- SPLC 2020

Stéphanie Challita:

- ICSA 2021, ECR track
- ICSA 2021, Artifacts Evaluation track
- FormaliSE 2021
- SAC 2021, SA-TTA track
- MODELS 2020, Demo Track

Benoit Combemale:

- ACM SRC for MODELS'20
- MODELS'20 (program board)
- ECMFA'20
- FDL'20
- CBI'20
- QUATIC'20
- MLE'20 workshop at MODELS'20
- DevOps@MODELS'20 workshop at MODELS'20
- CoMoDiTy'20 workshop at ER'20
- PNSE'20 workshop
- MoSC'20 workshop

Olivier Barais:

- ICSR'20
- COMPAS'2020
- SecureMDE'2020 at MODELS'20

Djamel Khelladi:

- MSR 2021
- VSC at WETICE 2019, 2020, 2021
- ME workshop at MODELS 2020

**Reviewer** Arnaud Blouin:

- ICSE'20

### 10.1.2 Journal

**Member of the editorial boards** Jean-Marc Jézéquel:

- Journal of Software and Systems Modeling: SoSyM (Associate Editor in Chief)
- IEEE Computer (Associate Editor in Chief)
- Journal of Systems and Software: JSS
- Journal of Object Technology: JOT

Benoit Combemale:

- Journal of Object Technology: JOT (Deputy Editors-in-Chief)
- Journal of Software and Systems Modeling: SoSyM
- Software Quality Journal (SQJ)
- Journal of Computer Languages (COLA)
- Journal on Science of Computer Programming (SCP, Advisory Board of the Software Section)

**Reviewer - reviewing activities** Arnaud Blouin:

- Journal Of System and Software

Stéphanie Challita:

- Journal of Object Technology: JOT
- IEEE Access

Olivier Barais:

- Journal Of System and Software
- Journal of Software and Systems Modeling: SoSyM

Djamel Khelladi:

- Journal of Empirical Software Engineering: EMSE
- Journal of Object Technology: JOT
- Journal of Software and Systems Modeling: SoSyM
- Software Quality Journal (SQJ)

### 10.1.3 Leadership within the scientific community

Arnaud Blouin:

- Founding member and co-organiser of the French GDR-GPL research action on Software Engineering and Human-Computer Interaction (GL-IHM).

Jean-Marc Jézéquel:

- Board member of Informatics Europe
- Member of the Scientific Committee of the GDR GPL of CNRS
- Member of the Advisory Board of the GEMOC Initiative: On the Globalization of Modeling Languages

Benoit Combemale:

- Chair of the steering committee of the ACM SIGPLAN Intl. Conference on Software Language Engineering (SLE)
- Founding member of the steerng committee of the Modeling Language Engineering and Execution (MLE) workshop series
- Funding Member of the Advisory Board of the GEMOC Initiative: On the Globalization of Modeling Languages

Djamel Khelladi:

- Co-organiser of the French GDR-GPL research action on Software Velocity - software engineering.

Olivier Zendra:

- Scientific Coordinattor of EU H2020 TeamPlay project
- Founder and member of the Steering Committee of the International Workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems (ICOOOLPS)
- Member of the EU HiPEAC CSA project Steering Committee
- Member of the HiPEAC Vision Editorial Board

### 10.1.4 Scientific expertise

Olivier Barais:

- Advisory board member for the H2020 ENACT european Project
- Expert for the evaluation of the project provided by *Ministères des affaires étrangères*

Olivier Zendra:

- Scientific expert for CIR/JEI for *Ministère de l'enseignement supérieur, de la recherche et de l'innovation*

### 10.1.5 Research administration

Jean-Marc Jézéquel:

- Director of UMR6074 IRISA (850 persons)
- Director of EIT Digital in Rennes
- Coordinator of the Acaemic research of Pole d'Excellence Cyber

Olivier Barais:

- Vice Dean of ISTIC (1800 students, 75 associate profs or profs)

Olivier Zendra:

- Member of Inria Evaluation Committee.

## 10.2 Teaching - Supervision - Juries

### 10.2.1 Teaching

The DIVERSE team bears the bulk of the teaching on Software Engineering at the University of Rennes 1 and at INSA Rennes, for the first year of the Master of Computer Science (Project Management, Object-Oriented Analysis and Design with UML, Design Patterns, Component Architectures and Frameworks, Validation & Verification, Human-Computer Interaction) and for the second year of the MSc in software engineering (Model driven Engineering, Aspect-Oriented Software Development, Software Product Lines, Component Based Software Development, Validation & Verification, *etc.*).

Each of Jean-Marc Jézéquel, Noël Plouzeau, Olivier Barais, Benoit Combemale, Johann Bourcier, Arnaud Blouin, Stéphanie Challita and Mathieu Acher teaches about 250h in these domains for a grand total of about 2000 hours, including several courses at ENSTB, IMT, ENS Rennes and ENSAI Rennes engineering school.

Olivier Barais is deputy director of the electronics and computer science teaching department of the University of Rennes 1. Olivier Barais is the head of the Master in Computer Science at the University of Rennes 1. Johann Bourcier is the head of the Information Technology department and member of the management board at the ESIR engineering school in Rennes. Arnaud Blouin is in charge of industrial relationships for the computer science department at INSA Rennes and elected member of this CS department council.

The DIVERSE team also hosts several MSc and summer trainees every year.

### 10.2.2 Supervision

- PhD defense in 2020: Jean-Émile Dartois, *Efficient resources management for hybrid cloud computing*, 2016, O. Barais, Jalil Boukhobza.
- PhD defense in 2020: Tristan Ninet, *Formal verification of the Internet Key Exchange (IKEv2) security protocol*, 2016, O. Zendra.
- PhD in progress: Alexandre Rio, *Demand Side Management A model driven approach to promote energy self-consumption* , 2016, O. Barais, Y. Morel
- PhD in progress: Dorian Leroy, *A generic and generative white-box testing framework for model transformations*, 2017, B. Combemale.
- PhD in progress: Fabien Coulon, *Web engineering for domain-specific modeling languages*, 2017, B. Combemale, S. Begaudeau.
- PhD in progress: June Benvegnu-Sallou, Decision support for the assessment of risks associated with the operation of underground environments, 2018, J-R. De Dreuzy, B. Combemale, J. Bourcier.
- PhD in progress: Pierre JeanJean, *Refining simulators by analyzing execution traces of complex systems*, 2018, O. Barais, B. Combemale.
- PhD in progress: Alif Akbar-pranata, *Chaos Engineering for IoT and Network Services*, 2018, O. Barais, J. Bourcier.
- PhD in progress: Antoine Cheron, *Abstractions for linked data and the programmable web*, 2018, O. Barais, J. Bourcier.
- PhD in progress: Gauthier Lyan, *Urban mobility: machine learning for building simulators using large amounts of data*, 2018, J-M. Jééquel, D. Gross Amblard.
- PhD in progress: Hugo Martin, *Learning variability*, 2018, M. Acher.
- PhD in progress: Lamine Noureddine, *Developing New Techniques for Packing Detection and Unpacking to Stop Malware Propagation*, 2018, O. Zendra.
- PhD in progress: Cassius De Oliveira Puodzius, *Machine learning for malware detection by aggregation of multiple threat feeds*, 2019, O. Zendra.

- PhD in progress: Luc Lesoil, *Deep variability in large-scaled systems*, 2019, M. Acher. A. Blouin, JM Jézéquel.
- PhD in progress: Emmanuel Chebbi, *Domain-Specific Language Reuse*, 2019, B. Combemale, O. Barais, G. Leguernic
- PhD in progress: Mohamed Handaoui, *Scheduling Big Data applications in the cloud: the case of machine learning algorithms*, 2019, Jalil Boukhobza. Olivier Barais.
- PhD in progress: Quentin Le Dilavrec, *Co-evolution between code, tests, and third-part libraries*, 2020, Arnaud Blouin, Jean-Marc Jézéquel, Djamel Eddine Khelladi.
- PhD in progress: Gwendal Jouneaux, *Self-Adaptive Language*, 2020, B. Combemale, O. Barais, G. Mussbacher.
- PhD in progress: Anne Bumiller, *Contextual Modeling of Adaptive Authentication Systems*, 2020, Olivier Barais, Benoit Combemale, Stéphanie Challita.

#### 10.2.3 Juries

**Olivier Barais** was in the examination committee of the following PhD and HDR thesis:

- HDR: Sophie Ebersold, Univ Toulouse, (Mars 2021), (Reviewer)
- PhD: Sami Lazreg Univ Nice (December 2020) (Reviewer)
- Ajay Muroor, Univ Grenoble, (December 2020) (examiner)
- Julien Delplanque Univ Lille (December 2020) (Reviewer)

**Benoit Combemale** was in the examination committee of the following PhD thesis:

- Valentin Besnard, ENSTA Bretagne (December 2020), (Reviewer)

**Olivier Zendra** was in the examination committee of the following PhD thesis:

- Anthony Ferrand, LIRMM / Université de Montpellier (September 2020), (Examiner)

# 11 Scientific production

## 11.1 Major publications

[1] M. Acher, R. E. Lopez-Herrejon and R. Rabiser. 'Teaching Software Product Lines: A Snapshot of Current Practices and Challenges'. In: *ACM Transactions of Computing Education* (May 2017). URL: https://hal.inria.fr/hal-01522779.

[2] B. Baudry and M. Monperrus. 'The Multiple Facets of Software Diversity: Recent Developments in Year 2000 and Beyond'. In: *ACM Computing Surveys* 48.1 (2015), 16:1–16:26. URL: https://hal.inria.fr/hal-01182103.

[3] G. Bécan, M. Acher, B. Baudry and S. Ben Nasr. 'Breathing Ontological Knowledge Into Feature Model Synthesis: An Empirical Study'. In: *Empirical Software Engineering* 21.4 (2015), pp. 1794–1841. DOI: 10.1007/s10664-014-9357-1. URL: https://hal.inria.fr/hal-01096969.

[4] A. Blouin, V. Lelli, B. Baudry and F. Coulon. 'User Interface Design Smell: Automatic Detection and Refactoring of Blob Listeners'. In: *Information and Software Technology* 102 (May 2018), pp. 49–64. DOI: 10.1016/j.infsof.2018.05.005. URL: https://hal.inria.fr/hal-01499106.

[5] M. Boussaa, O. Barais, G. Sunyé and B. Baudry. 'Leveraging metamorphic testing to automatically detect inconsistencies in code generator families'. In: *Software Testing, Verification and Reliability* (Dec. 2019). DOI: 10.1002/stvr.1721. URL: https://hal.inria.fr/hal-02422437.

[6] E. Bousse, D. Leroy, B. Combemale, M. Wimmer and B. Baudry. 'Omniscient Debugging for Executable DSLs'. In: *Journal of Systems and Software* 137 (Mar. 2018), pp. 261–288. DOI: 10.1016/j.jss.2017.11.025. URL: https://hal.inria.fr/hal-01662336.

[7] B. Combemale, J. Deantoni, B. Baudry, R. B. France, J.-M. Jézéquel and J. Gray. 'Globalizing Modeling Languages'. In: *IEEE Computer* (June 2014), pp. 10–13. URL: https://hal.inria.fr/hal-00994551.

[8] K. Corre, O. Barais, G. Sunyé, V. Frey and J.-M. Crom. 'Why can't users choose their identity providers on the web?' In: *Proceedings on Privacy Enhancing Technologies* 2017.3 (Jan. 2017), pp. 72–86. DOI: 10.1515/popets-2017-0029. URL: https://hal.archives-ouvertes.fr/hal-01611048.

[9] J.-E. Dartois, J. Boukhobza, A. Knefati and O. Barais. 'Investigating Machine Learning Algorithms for Modeling SSD I/O Performance for Container-based Virtualization'. In: *IEEE transactions on cloud computing* 14 (2019), pp. 1–14. DOI: 10.1109/TCC.2019.2898192. URL: https://hal.inria.fr/hal-02013421.

[10] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Clelang-Huang and P. Heymans. 'Feature Model Extraction from Large Collections of Informal Product Descriptions'. In: *Proc. of the Europ. Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC/FSE)*. Sept. 2013, pp. 290–300. DOI: 10.1145/2491411.2491455. URL: https://hal.inria.fr/hal-00859475.

[11] T. Degueule, B. Combemale, A. Blouin, O. Barais and J.-M. Jézéquel. 'Melange: A Meta-language for Modular and Reusable Development of DSLs'. In: *Proc. of the Int. Conf. on Software Language Engineering (SLE)*. Oct. 2015. URL: https://hal.inria.fr/hal-01197038.

[12] J. A. Galindo Duarte, M. Alférez, M. Acher, B. Baudry and D. Benavides. 'A Variability-Based Testing Approach for Synthesizing Video Sequences'. In: *Proc. of the Int. Symp. on Software Testing and Analysis (ISSTA)*. July 2014. URL: https://hal.inria.fr/hal-01003148.

[13] I. Gonzalez-Herrera, J. Bourcier, E. Daubert, W. Rudametkin, O. Barais, F. Fouquet, J.-M. Jézéquel and B. Baudry. 'ScapeGoat: Spotting abnormal resource usage in component-based reconfigurable software systems'. In: *Journal of Systems and Software* (2016). DOI: 10.1016/j.jss.2016.02.027. URL: https://hal.inria.fr/hal-01354999.

[14] A. Halin, A. Nuttinck, M. Acher, X. Devroey, G. Perrouin and B. Baudry. 'Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack'. In: *Empirical Software Engineering* (July 2018), pp. 1–44. DOI: 10.1007/s10664-018-9635-4. URL: https://hal.inria.fr/hal-01829928.

[15] J.-M. Jézéquel, B. Combemale, O. Barais, M. Monperrus and F. Fouquet. 'Mashup of Meta-Languages and its Implementation in the Kermeta Language Workbench'. In: *Software and Systems Modeling* 14.2 (2015), pp. 905–920. URL: https://hal.inria.fr/hal-00829839.

[16] D. E. Khelladi, B. Combemale, M. Acher and O. Barais. 'On the Power of Abstraction: a Model-Driven Co-evolution Approach of Software Code'. In: *42nd International Conference on Software Engineering, New Ideas and Emerging Results*. Séoul, South Korea, May 2020. URL: https://hal.inria.fr/hal-03029426.

[17] P. Laperdrix, W. Rudametkin and B. Baudry. 'Beauty and the Beast: Diverting modern web browsers to build unique browser fingerprints'. In: *Proc. of the Symp. on Security and Privacy (S&P)*. May 2016. URL: https://hal.inria.fr/hal-01285470.

[18] M. Leduc, T. Degueule, E. Van Wyk and B. Combemale. 'The Software Language Extension Problem'. In: *Software and Systems Modeling* (2019), pp. 1–4. URL: https://hal.inria.fr/hal-02399166.

[19] M. Rodriguez-Cancio, B. Combemale and B. Baudry. 'Automatic Microbenchmark Generation to Prevent Dead Code Elimination and Constant Folding'. In: *Proc. of the Int. Conf. on Automated Software Engineering (ASE)*. Sept. 2016. URL: https://hal.inria.fr/hal-01343818.

[20] P. Temple, M. Acher, J.-M. Jezequel and O. Barais. 'Learning-Contextual Variability Models'. In: *IEEE Software* 34.6 (Nov. 2017), pp. 64–70. DOI: 10.1109/MS.2017.4121211. URL: https://hal.inria.fr/hal-01659137.

[21] P. Temple, M. Acher and J.-M. Jézéquel. 'Empirical Assessment of Multimorphic Testing'. In: *IEEE Transactions on Software Engineering* (July 2019), pp. 1–21. DOI: 10.1109/TSE.2019.2926971. URL: https://hal.inria.fr/hal-02177158.

[22] P. Temple, G. Perrouin, M. Acher, B. Biggio, J.-M. Jézéquel and F. Roli. 'Empirical Assessment of Generating Adversarial Configurations for Software Product Lines'. In: *Empirical Software Engineering* (Dec. 2020), pp. 1–57. URL: https://hal.inria.fr/hal-03045797.

[23] O. L. Vera-Pérez, B. Danglot, M. Monperrus and B. Baudry. 'A Comprehensive Study of Pseudo-tested Methods'. In: *Empirical Software Engineering* (2018), pp. 1–33. DOI: 10.1007/s10664-018-9653-2. URL: https://hal.inria.fr/hal-01867423.

## 11.2 Publications of the year

**International journals**

[24] M. Bagherzadeh, K. Jahed, B. Combemale and J. Dingel. 'Live Modeling in the Context of State Machine Models and Code Generation'. In: *Software and Systems Modeling* (2020), pp. 1–44. DOI: 10.1007/s10270-020-00829-y. URL: https://hal.inria.fr/hal-02942374.

[25] J.-M. Bruel, B. Combemale, E. M. Guerra, J.-M. Jézéquel, J. Kienzle, J. De Lara, G. Mussbacher, E. Syriani and H. Vangheluwe. 'Comparing and Classifying Model Transformation Reuse Approaches across Metamodels'. In: *Software and Systems Modeling* 19.2 (2020), pp. 441–465. DOI: 10.1007/s10270-019-00762-9. URL: https://hal.inria.fr/hal-02317864.

[26] S. Challita, F. Korte, J. Erbel, F. Zalila, J. Grabowski and P. Merle. 'Model-Based Cloud Resource Management with TOSCA and OCCI'. In: *Software and Systems Modeling* (2021). URL: https://hal.archives-ouvertes.fr/hal-03122452.

[27] B. Combemale, J. Kienzle, G. Mussbacher, H. Ali, D. Amyot, M. Bagherzadeh, E. Batot, N. Bencomo, B. Benni, J.-M. Bruel, J. Cabot, B. H. C. Cheng, P. Collet, G. Engels, R. Heinrich, J.-M. Jézéquel, A. Koziolek, S. Mosser, R. Reussner, H. Sahraoui, R. Saini, J. Sallou, S. Stinckwich, E. Syriani and M. Wimmer. 'A Hitchhiker's Guide to Model-Driven Engineering for Data-Centric Systems'. In: *IEEE Software* (2020), pp. 1–9. DOI: 10.1109/MS.2020.2995125. URL: https://hal.inria.fr/hal-02612087.

[28] B. Danglot, M. Monperrus, W. Rudametkin and B. Baudry. 'An approach and benchmark to detect behavioral changes of commits in continuous integration'. In: *Empirical Software Engineering* 25.4 (July 2020), pp. 2379–2415. DOI: 10.1007/s10664-019-09794-7. URL: https://hal.inria.fr/hal-03121735.

[29] F. A. Fontana, G. Perrouin, A. Ampatzoglou, M. Acher, B. Walter, M. Cordy, F. Palomba and X. Devroey. 'MALTESQUE 2019 Workshop Summary'. In: *Software Engineering Notes* (24th Jan. 2020). DOI: 10.1145/3375572.3375582. URL: https://hal.inria.fr/hal-02471302.

[30] J. Kienzle, G. Mussbacher, B. Combemale, L. Bastin, N. Bencomo, J.-M. Bruel, C. Becker, S. Betz, R. Chitchyan, B. Cheng, S. Klingert, R. Paige, B. Penzenstadler, N. Seyff, E. Syriani and C. C. Venters. 'Towards Model-Driven Sustainability Evaluation'. In: *Communications of the ACM* 63.3 (2020), pp. 80–91. URL: https://hal.inria.fr/hal-02146543.

[31] R. Kretschmer, D. E. Khelladi and A. Egyed. 'Transforming Abstract to Concrete Repairs with a Generative Approach of Repair Values'. In: *Journal of Systems and Software* (2021). URL: https://hal.inria.fr/hal-03127118.

[32] R. Kretschmer, D. E. Khelladi, R. E. Lopez-Herrejon and A. Egyed. 'Consistent Change Propagation within Models'. In: *Software and Systems Modeling* (6th July 2020). DOI: 10.1007/s10270-020-00823-4. URL: https://hal.inria.fr/hal-03029432.

[33] M. Leduc, T. Degueule, E. Van Wyk and B. Combemale. 'The Software Language Extension Problem'. In: *Software and Systems Modeling* 19.2 (2020), pp. 263–267. URL: https://hal.inria.fr/hal-02399166.

[34] M. Leduc, G. Jouneaux, T. Degueule, G. Le Guernic, O. Barais and B. Combemale. 'Automatic generation of Truffle-based interpreters for Domain-Specific Languages'. In: *The Journal of Object Technology* 19.2 (2020), pp. 1–21. DOI: 10.5381/jot.2020.19.2.a1. URL: https://hal.inria.fr/hal-02395867.

[35] D. Leroy, E. Bousse, M. Wimmer, T. Mayerhofer, B. Combemale and W. Schwinger. 'Behavioral interfaces for executable DSLs'. In: *Software and Systems Modeling* (23rd Apr. 2020). DOI: 10.1007/s10270-020-00798-2. URL: https://hal.archives-ouvertes.fr/hal-02565549.

[36] D. Leroy, P. Jeanjean, E. Bousse, M. Wimmer and B. Combemale. 'Runtime Monitoring for Executable DSLs'. In: *The Journal of Object Technology* 19.2 (2020), pp. 1–23. DOI: 10.5381/jot.2020.19.2.a6. URL: https://hal.inria.fr/hal-03109992.

[37] G. Mussbacher, B. Combemale, J. Kienzle, S. Abrahão, H. Ali, N. Bencomo, M. Búr, L. Burgueño, G. Engels, P. Jeanjean, J.-M. Jézéquel, T. Kühn, S. Mosser, H. Sahraoui, E. Syriani, D. Varró and M. Weyssow. 'Opportunities in Intelligent Modeling Assistance'. In: *Software and Systems Modeling* (2020), pp. 1–7. DOI: 10.1007/s10270-020-00814-5. URL: https://hal.inria.fr/hal-02876536.

[38] A. Pierantonio, M. V. D. Brand and B. Combemale. 'Open Access: all you wanted to know and never dared to ask'. In: *The Journal of Object Technology* (2020). URL: https://hal.inria.fr/hal-03137877.

[39] P. Temple, G. Perrouin, M. Acher, B. Biggio, J.-M. Jézéquel and F. Roli. 'Empirical Assessment of Generating Adversarial Configurations for Software Product Lines'. In: *Empirical Software Engineering* (8th Dec. 2020), pp. 1–57. URL: https://hal.inria.fr/hal-03045797.

[40] X. Tërnava and P. Collet. 'A framework for managing the imperfect modularity of variability implementations'. In: *Journal of Computer Languages* (Sept. 2020), pp. 1–39. DOI: 10.1016/j.cola.2020.100998. URL: https://hal.archives-ouvertes.fr/hal-02951745.

[41] A. Wortmann, O. Barais, B. Combemale and M. Wimmer. 'Modeling Languages in Industry 4.0: An Extended Systematic Mapping Study'. In: *Software and Systems Modeling* 19.1 (2020), pp. 67–94. DOI: 10.1007/s10270-019-00757-6. URL: https://hal.inria.fr/hal-02282028.

**International peer-reviewed conferences**

[42] A. Alidra, A. Beugnard, H. Godfroy, P. Kimmel and G. Le Guernic. 'A Language Agnostic Approach to Modeling Requirements: Specification and Verification'. In: MODELS '20 Companion. Virtual Event, Canada, 18th Oct. 2020. DOI: 10.1145/3417990.3419224. URL: https://hal.inria.fr/hal-02924645.

[43] J. Alves Pereira, M. Acher, H. Martin and J.-M. Jézéquel. 'Sampling Effect on Performance Prediction of Configurable Systems: A Case Study'. In: ICPE 2020 - 11th ACM/SPEC International Conference on Performance Engineering. Edmonton, Canada, 20th Apr. 2020, pp. 1–13. URL: https://hal.inria.fr/hal-02356290.

[44] F. Bordeleau, B. Combemale, R. Eramo, M. Van Den Brand and M. Wimmer. 'Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges'. In: ICSMM 2020 - International Conference on Systems Modelling and Management. Bergen, Norway, 25th June 2020. URL: https://hal.inria.fr/hal-02946949.

[45] F. Coulon, A. Auvolat, B. Combemale, Y.-D. Bromberg, F. Taïani, O. Barais and N. Plouzeau. 'Modular and Distributed IDE'. In: SLE 2020 - 13th ACM SIGPLAN International Conference on Software Language Engineering. Virtual, United States, Nov. 2020, pp. 270–282. DOI: 10.1145/3426425.3426947. URL: https://hal.archives-ouvertes.fr/hal-02964806.

[46] M. Handaoui, J.-E. Dartois, J. Boukhobza, O. Barais and L. D'orazio. 'ReLeaSER: A Reinforcement Learning Strategy for Optimizing Utilization Of Ephemeral Cloud Resources'. In: CloudCom 2020 - 12th IEEE International Conference on Cloud Computing Technology and Science. Bangkok, Thailand, 14th Dec. 2020, pp. 1–9. URL: https://hal.archives-ouvertes.fr/hal-02989286.

[47] M. Handaoui, J.-E. Dartois, L. Lemarchand and J. Boukhobza. 'Salamander: a Holistic Scheduling of MapReduce Jobs on Ephemeral Cloud Resources'. In: CCGRID 2020 - 20th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Melbourne, Australia, 2nd Nov. 2020, pp. 1–10. URL: https://hal.archives-ouvertes.fr/hal-02497029.

[48] P. Jeanjean, B. Combemale and O. Barais. 'IDE as Code: Reifying Language Protocols as First-Class Citizens'. In: ISEC 2021 - Innovations in Software Engineering Conference. Bhubaneswar / Virtual, India, 25th Feb. 2021. URL: https://hal.inria.fr/hal-03107122.

[49] D. E. Khelladi, B. Combemale, M. Acher and O. Barais. 'On the Power of Abstraction: a Model-Driven Co-evolution Approach of Software Code'. In: 42nd International Conference on Software Engineering, New Ideas and Emerging Results. Séoul, South Korea, 27th May 2020. URL: https://hal.inria.fr/hal-03029426.

[50] D. E. Khelladi, B. Combemale, M. Acher, O. Barais and J.-M. Jézéquel. 'Co-Evolving Code with Evolving Metamodels'. In: 42nd International Conference on Software Engineering. Séoul, South Korea, 27th May 2020. URL: https://hal.inria.fr/hal-03029429.

[51] L. Lesoil, M. Acher, A. Blouin and J.-M. Jézéquel. 'Deep Software Variability: Towards Handling Cross-Layer Configuration'. In: VaMoS 2021 - 15th International Working Conference on Variability Modelling of Software-Intensive Systems. Krems / Virtual, Austria: https://vamos2021.fh-krems.ac.at/, 9th Feb. 2021. URL: https://hal.inria.fr/hal-03084276.

[52] N. Messe, N. Belloir, V. Chiprianov, J. El-Hachem, R. Fleurquin and S. Sadou. 'An Asset-Based Assistance for Secure by Design'. In: APSEC 2020 - 27th Asia-Pacific Software Engineering Conference. Singapore, Singapore, 1st Dec. 2020, pp. 1–10. URL: https://hal.archives-ouvertes.fr/hal-02990897.

[53] N. Messe, V. Chiprianov, N. Belloir, J. El-Hachem, R. Fleurquin and S. Sadou. 'Asset-Oriented Threat Modeling'. In: TrustCom 2020 - 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. Guangzhou, China, 31st Dec. 2020, pp. 1–11. URL: https://hal.archives-ouvertes.fr/hal-02990919.

[54] G. Mussbacher, B. Combemale, S. Abrahão, N. Bencomo, L. Burgueño, G. Engels, J. Kienzle, T. Kühn, S. Mosser, H. Sahraoui and M. Weyssow. 'Towards an Assessment Grid for Intelligent Modeling Assistance'. In: MDE Intelligence 2020 - 2nd Workshop on Artificial Intelligence and Model-driven Engineering. Vol. 10. Montreal, Canada, 18th Oct. 2020, pp. 1–10. URL: https://hal.inria.fr/hal-02925142.

[55] L. Noureddine, A. Heuser, C. Puodzius and O. Zendra. 'SE-PAC: A Self-Evolving PAcker Classifier against rapid packers evolution'. In: CODASPY '21: Eleventh ACM Conference on Data and Application Security and Privacy. Virtual Event, United States, 26th Apr. 2021. DOI: 10.1145/3422337.3447848. URL: https://hal.inria.fr/hal-03149211.

[56] J. A. Pereira, H. Martin, P. Temple and M. Acher. 'Machine Learning and Configurable Systems: A Gentle Introduction'. In: SPLC 2020 - 24th ACM International Systems and Software Product Line Conference. Montreal, Canada, 19th Oct. 2020, p. 1. DOI: 10.1145/3382025.3414976. URL: https://hal.inria.fr/hal-03020125.

[57] A. A. Pranata, O. Barais, J. Bourcier and L. Noirie. 'Misconfiguration Discovery with Principal Component Analysis for Cloud-Native Services'. In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. UCC 2020 - 13th IEEE/ACM International Conference on Utility and Cloud Computing. Leicester / Virtual, United Kingdom, 30th Dec. 2020, pp. 269–278. URL: https://hal.archives-ouvertes.fr/hal-03137874.

[58] J. Sallou, A. Gauvain, J. Bourcier, B. Combemale and J.-R. De Dreuzy. 'Loop Aggregation for Approximate Scientific Computing'. In: *Computational Science – ICCS 2020*. International Conference on Computational Science. Lecture Notes in Computer Science book series. Amsterdam, Netherlands, June 2020, pp. 141–155. DOI: 10.1007/978-3-030-50417-5_11. URL: https://hal.archives-ouvertes.fr/hal-02545875.

[59] L. Thomas Van Binsbergen, M. V. Merino, P. Jeanjean, T. Van Der Storm, B. Combemale and O. Barais. 'A principled approach to REPL interpreters'. In: SPLASH 2020 - ACM SIGPLAN conference on Systems, Programming, Languages, and Applications: Software for Humanity. Chicago / Virtual, United States, 15th Nov. 2020, pp. 1–17. DOI: 10.1145/3426428.3426917. URL: https://hal.inria.fr/hal-02968938.

**Scientific book chapters**

[60] B. Baudry, Y.-D. Bromberg, D. Frey, A. Gómez-Boix, P. Laperdrix and F. Taïani. 'Profilage de navigateurs : état de l'art et contre-mesures'. In: *Le profilage en ligne : entre libéralisme et régulation*. 15th Oct. 2020. URL: https://hal.inria.fr/hal-03043187.

**Doctoral dissertations and habilitation theses**

[61] J.-E. Dartois. 'Leveraging Cloud unused heterogeneous resources for applications with SLA guarantees'. Univ-Rennes1, 4th Sept. 2020. URL: https://hal.inria.fr/tel-03009816.

## 11.3 Cited publications

[62] A. Arcuri and L. C. Briand. 'A practical guide for using statistical tests to assess randomized algorithms in software engineering'. In: *ICSE*. 2011, pp. 1–10.

[63] A. Avizienis. 'The N-version approach to fault-tolerant software'. In: *Software Engineering, IEEE Transactions on* 12 (1985), pp. 1491–1501.

[64] F. Bachmann and L. Bass. 'Managing variability in software architectures'. In: *SIGSOFT Softw. Eng. Notes* 26 (3 May 2001), pp. 126–132. DOI: http://doi.acm.org/10.1145/379377.375274. URL: http://doi.acm.org/10.1145/379377.375274.

[65] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone and A. Sangiovanni-Vincentelli. 'Metropolis: An integrated electronic system design environment'. In: *Computer* 36.4 (2003), pp. 45–52.

[66] E. Baniassad and S. Clarke. 'Theme: an approach for aspect-oriented analysis and design'. In: *26th International Conference on Software Engineering (ICSE)*. 2004, pp. 158–167.

[67] E. G. Barrantes, D. H. Ackley, S. Forrest and D. Stefanović. 'Randomized instruction set emulation'. In: *ACM Transactions on Information and System Security (TISSEC)* 8.1 (2005), pp. 3–40.

[68] D. Batory, R. E. Lopez-Herrejon and J.-P. Martin. 'Generating Product-Lines of Product-Families'. In: *ASE '02: Automated software engineering*. IEEE, 2002, pp. 81–92.

[69] S. Becker, H. Koziolek and R. Reussner. 'The Palladio component model for model-driven performance prediction'. In: *Journal of Systems and Software* 82.1 (Jan. 2009), pp. 3–22.

[70] N. Bencomo. 'On the use of software models during software execution'. In: *MISE '09: Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering*. IEEE Computer Society, May 2009.

[71] A. Beugnard, J.-M. Jézéquel and N. Plouzeau. 'Contract Aware Components, 10 years after'. In: *WCSI*. 2010, pp. 1–11.

[72] J. Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.

[73] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. H. Obbink and K. Pohl. 'Variability Issues in Software Product Lines'. In: *PFE '01: Revised Papers from the 4th International Workshop on Software Product-Family Engineering*. London, UK: Springer-Verlag, 2002, pp. 13–21.

[74] L. C. Briand, E. Arisholm, S. Counsell, F. Houdek and P. Thévenod–Fosse. 'Empirical studies of object-oriented artifacts, methods, and processes: state of the art and future directions'. In: *Empirical Software Engineering* 4.4 (1999), pp. 387–404.

[75] J. T. Buck, S. Ha, E. A. Lee and D. G. Messerschmitt. 'Ptolemy: A framework for simulating and prototyping heterogeneous systems'. In: *Int. Journal of Computer Simulation* (1994).

[76] T. Bures, P. Hnetynka and F. Plasil. 'Sofa 2.0: Balancing advanced features in a hierarchical component model'. In: *Software Engineering Research, Management and Applications, 2006. Fourth International Conference on*. IEEE. 2006, pp. 40–48.

[77] B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns and J. Whittle. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Ed. by D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi and J. Magee. Vol. 5525. Betty H. C. Cheng, Rogério de Lemos, Holger Giese, Paola Inverardi, and Jeff Magee. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[78] J. Coplien, D. Hoffman and D. Weiss. 'Commonality and Variability in Software Engineering'. In: *IEEE Software* 15.6 (1998), pp. 37–45.

[79] I. Crnkovic, S. Sentilles, A. Vulgarakis and M. R. Chaudron. 'A classification framework for software component models'. In: *Software Engineering, IEEE Transactions on* 37.5 (2011), pp. 593–615.

[80] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. 'A fast and elitist multiobjective genetic algorithm: NSGA-II'. In: *Evolutionary Computation, IEEE Transactions on* 6.2 (2002), pp. 182–197.

[81] R. DeMilli and A. J. Offutt. 'Constraint-based automatic test data generation'. In: *Software Engineering, IEEE Transactions on* 17.9 (1991), pp. 900–910.

[82] S. Forrest, A. Somayaji and D. H. Ackley. 'Building diverse computer systems'. In: *Operating Systems, 1997., The Sixth Workshop on Hot Topics in*. IEEE. 1997, pp. 67–72.

[83] R. B. France and B. Rumpe. 'Model-driven Development of Complex Software: A Research Roadmap'. In: *Proceedings of the Future of Software Engineering Symposium (FOSE '07)*. Ed. by L. C. Briand and A. L. Wolf. IEEE, 2007, pp. 37–54.

[84] S. Frey, F. Fittkau and W. Hasselbring. 'Search-based genetic optimization for deployment and reconfiguration of software in the cloud'. In: *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press. 2013, pp. 512–521.

[85] G. Halmans and K. Pohl. 'Communicating the Variability of a Software-Product Family to Customers'. In: *Software and System Modeling* 2.1 (2003), pp. 15–36.

[86] C. Hardebolle and F. Boulanger. 'ModHel'X: A component-oriented approach to multi-formalism modeling'. In: *Models in Software Engineering*. Springer, 2008, pp. 247–258.

[87] M. Harman and B. F. Jones. 'Search-based software engineering'. In: *Information and Software Technology* 43.14 (2001), pp. 833–839.

[88] H. Hemmati, L. C. Briand, A. Arcuri and S. Ali. 'An enhanced test case selection approach for model-based testing: an industrial case study'. In: *SIGSOFT FSE*. 2010, pp. 267–276.

[89] J. Hutchinson, J. Whittle, M. Rouncefield and S. Kristoffersen. 'Empirical assessment of MDE in industry'. In: *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. Ed. by R. N. Taylor, H. Gall and N. Medvidovic. ACM, 2011, pp. 471–480.

[90] J.-M. Jézéquel. 'Model Driven Design and Aspect Weaving'. In: *Journal of Software and Systems Modeling (SoSyM)* 7.2 (May 2008), pp. 209–218. URL: http://www.irisa.fr/triskell/publis/2008/Jezequel08a.pdf.

[91] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak and A. S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. Carnegie-Mellon University Software Engineering Institute, Nov. 1990.

[92] J. Kramer and J. Magee. 'Self-Managed Systems: an Architectural Challenge'. In: *Future of Software Engineering*. IEEE, 2007, pp. 259–268.

[93] K.-K. Lau, P. V. Elizondo and Z. Wang. 'Exogenous connectors for software components'. In: *Component-Based Software Engineering*. Springer, 2005, pp. 90–106.

[94] P. McMinn. 'Search-based software test data generation: a survey'. In: *Software Testing, Verification and Reliability* 14.2 (2004), pp. 105–156.

[95] J. Meekel, T. B. Horton and C. Mellone. 'Architecting for Domain Variability'. In: *ESPRIT ARES Workshop*. 1998, pp. 205–213.

[96] R. Mélisson, P. Merle, D. Romero, R. Rouvoy and L. Seinturier. 'Reconfigurable run-time support for distributed service component architectures'. In: *the IEEE/ACM international conference*. New York, New York, USA: ACM Press, 2010, p. 171.

[97] A. M. Memon. 'An event-flow model of GUI-based applications for testing'. In: *Software Testing, Verification and Reliability* 17.3 (2007), pp. 137–157.

[98] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey and A. Solberg. 'Models at Runtime to Support Dynamic Adaptation'. In: *IEEE Computer* (Oct. 2009), pp. 46–53. URL: http://www.irisa.fr/triskell/publis/2009/Morin09f.pdf.

[99] P.-A. Muller, F. Fleurey and J.-M. Jézéquel. 'Weaving Executability into Object-Oriented Meta-Languages'. In: *Proc. of MODELS/UML'2005*. LNCS. Jamaica: Springer, 2005.

[100] C. Nebut, Y. Le Traon and J.-M. Jézéquel. 'System Testing of Product Families: from Requirements to Test Cases'. In: *Software Product Lines*. Springer Verlag, 2006, pp. 447–478. URL: http://www.irisa.fr/triskell/publis/2006/Nebut06b.pdf.

[101] C. Nebut, S. Pickin, Y. Le Traon and J.-M. Jézéquel. 'Automated Requirements-based Generation of Test Cases for Product Families'. In: *Proc. of the 18th IEEE International Conference on Automated Software Engineering (ASE'03)*. 2003. URL: http://www.irisa.fr/triskell/publis/2003/nebut03b.pdf.

[102] L. M. Northrop. 'A Framework for Software Product Line Practice'. In: *Proceedings of the Workshop on Object-Oriented Technology*. London, UK: Springer-Verlag, 1999, pp. 365–366.

[103] L. M. Northrop. 'SEI's Software Product Line Tenets'. In: *IEEE Softw.* 19.4 (2002), pp. 32–40.

[104] I. Ober, S. Graf and I. Ober. 'Validating timed UML models by simulation and verification'. In: *International Journal on Software Tools for Technology Transfer* 8.2 (2006), pp. 128–145.

[105] D. L. Parnas. 'On the Design and Development of Program Families'. In: *IEEE Trans. Softw. Eng.* 2.1 (1976), pp. 1–9.

[106] S. Pickin, C. Jard, T. Jéron, J.-M. Jézéquel and Y. Le Traon. 'Test Synthesis from UML Models of Distributed Software'. In: *IEEE Transactions on Software Engineering* 33.4 (Apr. 2007), pp. 252–268.

[107] K. Pohl, G. Böckle and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[108] B. Randell. 'System structure for software fault tolerance'. In: *Software Engineering, IEEE Transactions on* 2 (1975), pp. 220–232.

[109] M. Rinard. 'Obtaining and reasoning about good enough software'. In: *Proceedings of Annual Design Automation Conference (DAC)*. 2012, pp. 930–935.

[110] J. Rothenberg, L. E. Widman, K. A. Loparo and N. R. Nielsen. 'The Nature of Modeling'. In: *in Artificial Intelligence, Simulation and Modeling*. John Wiley & Sons, 1989, pp. 75–92.

[111] P. Runeson and M. Höst. 'Guidelines for conducting and reporting case study research in software engineering'. In: *Empirical Software Engineering* 14.2 (2009), pp. 131–164.

[112] D. Schmidt. 'Guest Editor's Introduction: Model-Driven Engineering'. In: *IEEE Computer* 39.2 (2006), pp. 25–31.

[113] F. Shull, J. Singer and D. I. Sjberg. *Guide to advanced empirical software engineering*. Springer, 2008.

[114] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann and M. Rinard. 'Managing performance vs. accuracy trade-offs with loop perforation'. In: *Proc. of the Symp. on Foundations of software engineering*. ESEC/FSE '11. Szeged, Hungary: ACM, 2011, pp. 124–134.

[115] J. Steel and J.-M. Jézéquel. 'On Model Typing'. In: *Journal of Software and Systems Modeling (SoSyM)* 6.4 (Dec. 2007), pp. 401–414. URL: http://www.irisa.fr/triskell/publis/2007/Steel07a.pdf.

[116] C. Szyperski, D. Gruntz and S. Murer. *Component software: beyond object-oriented programming*. Addison-Wesley, 2002.

[117] J.-C. Trigaux and P. Heymans. *Modelling variability requirements in Software Product Lines: a comparative survey*. Tech. rep. FUNDP Namur, 2003.

[118] M. Utting and B. Legeard. *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2010.

[119] P. Vromant, D. Weyns, S. Malek and J. Andersson. 'On interacting control loops in self-adaptive systems'. In: *SEAMS 2011*. ACM, 2011, pp. 202–207.

[120] C. Yilmaz, M. B. Cohen and A. A. Porter. 'Covering arrays for efficient fault characterization in complex configuration spaces'. In: *Software Engineering, IEEE Transactions on* 32.1 (2006), pp. 20–34.

[121] Z. A. Zhu, S. Misailovic, J. A. Kelner and M. Rinard. 'Randomized accuracy-aware program transformations for efficient approximate computations'. In: *Proc. of the Symp. on Principles of Programming Languages (POPL)*. 2012, pp. 441–454.

[122] T. Ziadi and J.-M. Jézéquel. 'Product Line Engineering with the UML: Deriving Products'. In: Springer Verlag, 2006, pp. 557–586.