UMR IRISA

Activity Report 2021

Team DIVERSE

Diversity-centric Software Engineering

Joint team with Inria Rennes – Bretagne Atlantique

D4 – Language and Software Engineering

Contents

Project-Team DIVERSE

Creation of the 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]
- Gunter Mussbacher [INRIA, Chair, Professor at McGill]
- Olivier Zendra [Inria, Researcher]

Faculty Members

- Olivier Barais [Team leader, Univ de Rennes I, Professor, HDR]
- Mathieu Acher [Univ de Rennes I, Associate Professor]
- Raounak Benabidallah [Univ de Rennes I, ATER, until Aug 2021]
- Reda Bendraou [Univ de Nanterre, Associate Professor, from Sep 2021, HDR]
- Arnaud Blouin [INSA Rennes, Associate Professor, HDR]
- Johann Bourcier [Univ de Rennes I, Associate Professor, HDR]
- Stéphanie Challita [Univ de Rennes I, Associate Professor]
- Elyes Cherfa [Univ de Rennes I, from Sep 2021]
- 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]
- Oscar Luis Vera Perez [Univ de Rennes I, ATER, until Feb 2021]
- Nan Zhang Messe [Univ de Rennes I, ATER, until Aug 2021]

Post-Doctoral Fellow

- Xhevahire Ternava [Univ de Rennes I]

PhD Students

- June Benvegnu-Sallou [Univ de Rennes I]
- Anne Bumiller [Orange, CIFRE]
- Emmanuel Chebbi [Inria]
- Antoine Cheron [Zengularity SAS, CIFRE, until Oct 2021]
- Cassius De Oliveira Puodzius [Inria]
- Pierre Jeanjean [Inria]
- Gwendal Jouneaux [Univ de Rennes I]
- Piergiorgio Ladisa [SAP, CIFRE]
- Quentin Le Dilavrec [Univ de Rennes I]
- Luc Lesoil [Univ de Rennes I]

- Gauthier Lyan [Keolis, CIFRE, until Jul 2021]
- Hugo Martin [Univ de Rennes I, until Oct 2021]
- Lamine Noureddine [Inria, until Feb 2021]
- Alif Akbar Pranata [Inria, until Nov 2021]
- Georges Aaron Randrianaina [Univ de Rennes I, from Jul 2021]

Technical Staff

- Florian Badie [Inria, Engineer, from Nov 2021]
- Romain Belafia [Univ de Rennes I, Engineer, from Sep 2021]
- Theo Giraudet [Univ de Rennes I, Engineer, from Sep 2021]
- Bruno Lebon [Inria, Engineer, from Jul 2021 until Oct 2021]
- Romain Lefeuvre [Inria, Engineer, from Sep 2021]
- Dorian Leroy [Inria, Engineer]
- Didier Vojtisek [Inria, Engineer]

Interns and Apprentices

- Romain Belafia [Inria, from Feb 2021 until Jul 2021]
- Antoine Bouffard [Inria, from May 2021 until Aug 2021]
- Remi Daniel [INSA Rennes, from May 2021 until Aug 2021]
- Willy Fortin [Inria, from May 2021 until Aug 2021]
- Theo Giraudet [Univ de Rennes I, from May 2021 until Jul 2021]
- Antoine Gratia [Univ de Rennes I, from Mar 2021 until Jun 2021]
- Mouna Hissane [Univ de Rennes I, from May 2021 until Aug 2021]
- Philemon Houdaille [Inria, from May 2021 until Jul 2021]
- Zohra Kaouter Kebaili [INSA Rennes, from Mar 2021 until Sep 2021]
- Meziane Khodja [Univ de Rennes I, from Jun 2021 until Aug 2021]
- Alexandre Le Balanger [Univ de Rennes I, from Jun 2021 until Aug 2021]
- Marius Lumbroso [Univ de Rennes I, from May 2021 until Aug 2021]
- Quentin Mazouni [Inria, from Oct 2021]
- Jean Loup Moll [Univ de Rennes I, from Jun 2021 until Jul 2021]
- Koffi Ismael Ouattara [Inria, from Apr 2021 until Aug 2021]
- Georges Aaron Randrianaina [Univ de Rennes I, from Feb 2021 until Jul 2021]
- Leo Rolland [Univ de Rennes I, from Jun 2021 until Aug 2021]

Administrative Assistant

- Sophie Maupile [CNRS]

External Collaborator

- Gurvan Le Guernic [DGA]

2 Overall objectives

DIVERSE's research agenda targets core values of software engineering. In this fundamental domain we focus on 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 borne 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 services). Today these systems seem to be all radically 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**, where software has to run and adapted; **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 as well as in its environment. Conversely, explicitly handling diversity is a great opportunity to allow software to spontaneously explore alternative design solutions, and to mitigate security risks.

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 envelope.

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 Context

Applications are becoming more complex and the demand for faster development is increasing. In order to better adapt to the unbridled evolution of requirements in markets where software plays an essential role, companies are changing the way they design, develop, secure and deploy applications, by relying on:

- A massive use of reusable libraries from a rich but fragmented eco-system;

- An increasing configurability of most of the produced software;
- A strongly increase in evolution frequency;
- Cloud-native architectures based on containers, naturally leading to a diversity of programming languages used, and to the emergence of infrastructure, dependency, project and deployment descriptors (models);
- Implementations of fully automated software supply chains;
- The use of lowcode/nocode platforms;
- The use of ever richer integrated development environments (IDEs), more and more deployed in SaaS mode;
- The massive use of data and artificial intelligence techniques in software production chains.

These trends are set to continue, all the while with a strong concern about the security properties of the produced and distributed software.
The numbers in the examples below help to understand why this evolution of modern software engineering brings a **change of dimension**:

- When designing a simple kitchen sink (*hello world*) with the `angular` framework, more than 1600 dependencies of JavaScript libraries are pulled.
- The numbers revealed by Google in 2018 showed that over 500 million tests are run *per day* inside Google's systems, leading to over 4 millions daily builds.
- Also at Google, they reported 86 TB of data, including two billion lines of code in nine million source files [105]. Their software also rapidly evolves both in terms of frequency and in terms of size. Again, at Google, 25,000 developers typically commit 16,000 changes to the codebase on a single workday. This is also the case for most of software code, including open source software.
- x264, a highly popular and configurable video encoder, provides 100+ options that can take boolean, integer or string values. There are different ways of compiling x264, and it is well-known that the compiler options (e.g., -O1 –O2 –O3 of gcc) can influence the performance of a software; the widely used gcc compiler, for example, offers more than 200 options. The x264 encoder can be executed on different configurations of the Linux operating system, whose options may in turn influence x264 execution time; in recent versions (> 5), there are 16000+ options to the Linux kernel. Last but not least, x264 should be able to encode many different videos, in different formats and with different visual properties, implying a huge variability of the input space. Overall, the variability space is enormous, and ideally x264 should be run and tested in all these settings. But a rough estimation shows that the number of possible configurations, resulting from the combination of the different variability layers, is 10^{6000}.

The DIVERSE research project is working and evolving in the context of this acceleration. We are active at all stages of the **software supply chain**. Software supply chain covers all the activities and all the stakeholders that relate to software production and delivery. All these activities and stakeholders have to be smartly managed together as part of an overall strategy. The goal of supply chain management (SCM) is to meet customer demands with the most efficient use of resources possible.

In this context, DIVERSE is particularly interested in the following research questions:

- How to engineer tool-based abstractions for a given set of experts in order to foster their socio-technical collaboration;
- How to generate and exploit useful data for the optimization of this supply chain, in particular for the control of variability and the management of the co-evolution of the various software artifacts;
- How to increase the confidence in the produced software, by working on the resilience and security of the artifacts produced throughout this supply chain.

3.2 Scientific background

3.2.1 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) [109]. It provides DIVERSE with solid foundations to specify, analyze and reason about the different forms of diversity that occur throughout the development life cycle. 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 [81]. 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 [65]. 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*, including a unified typing theory that integrates models as first class entities [111].

Incorporating domain-specific concepts and a 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 (programming) field. Domains in which DSMLs have been developed and used include, among others, automotive, avionics, and cyber-physical systems. A study performed by Hutchinson et al. [86] 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 the 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 the 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 [95].

3.2.2 Variability modeling

While the basic vision underlying *Software Product Lines* (SPL) can probably be traced back to David Parnas' seminal article [102] on the Design and Development of Program Families, it is only quite recently that SPLs have started emerging as a paradigm shift towards modeling and developing software system families rather than individual systems [99]. 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* [100]. Bosch provides a different definition [71]: *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 [77]. 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 [83]. 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 [72]. Variability management is thus increasingly seen as the cornerstone of SPL development, covering the entire development life cycle, from requirements elicitation [113] to product derivation [117] to product testing [98, 97].

Halmans *et al.* [83] distinguish between *essential* and *technical* variability, especially at the 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.* [104]: 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.* [92] (feature, hardware platform, performance and attributes variability) or Bass *et al.* [63] 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* [88]. 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 [72]. 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 [67]. Model Driven Engineering (MDE) makes it possible to automate this weaving process [87]. 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 fact an executable meta-model [96]) manipulating these models to produce for instance a detailed design that can ultimately be transformed to code, or to test suites [103], or other software artifacts.

3.2.3 Component-based software development

Component-based software development [112] 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 [78]: description of requirements on services required from the other components; indirect connections between components thanks to ports and connectors constructs [90]; hierarchical definition of components (assemblies of components can define new component types); connectors supporting various communication semantics [75]; quantitative properties on the services [70].

In recent years component-based architectures have evolved from static designs to dynamic, adaptive designs (e.g., SOFA [75], Palladio [68], Frascati [93]). 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 [89]. 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 [76, 115]. 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 [69], 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 [107]". 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 systems. 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 envelope at run-time. 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 [82]. Multi Objectives Search based techniques [79] 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.

3.2.4 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 envelope. 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 envelope 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 [114] mainly relies on model analysis, constraint solving [80] and search-based reasoning [91]. DIVERSE leverages in particular the applications of model-based testing in the context of highly-configurable systems and [116] interactive systems [94] as well as recent advances based on diversity for test cases selection [85].

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 [101], or Xholon. 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 [74] 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 [64] provides modeling elements amenable to semantically equivalent mathematical models. Metropolis offers a precise semantics flexible enough to support different models of computation. ModHel'X [84] studies the composition of multi-paradigm models relying on different 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 [62], recovery blocks [106] and code randomization [66], 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.

3.2.5 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 [110, 108]. Such methods have been used for example to understand the impact of new software development paradigms [73]. 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 [61].

3.3 Research axis

DIVERSE explore *Software Diversity*. Leveraging our strong background on Model-Driven Engineering, and our large expertise on several related fields (programming languages, distributed systems, GUI, machine learning, security...), *we explore tools and methods to embrace the inherent diversity in software engineering*, from the stakeholders and underlying tool-supported languages involved in the software system life cycle, to the configuration and evolution space of the modern software systems, and the heterogeneity of the targeted execution platforms. Hence, we organize our research directions according to three axes (cf. Fig. 1):

- **Axis #1: Software Language Engineering.** We explore the future engineering and scientific environments to support the socio-technical coordination among the various stakeholders involved across modern software system life cycles.

- **Axis #2: Spatio-temporal Variability in Software and Systems.** We explore systematic and automatic approaches to cope with software variability, both in space (software variants) and time (software maintenance and evolution).

- **Axis #3: DevSecOps and Resilience Engineering for Software and Systems.** We explore smart continuous integration and deployment pipelines to ensure the delivery of secure and resilient software systems on heterogeneous execution platforms (cloud, IoT...).

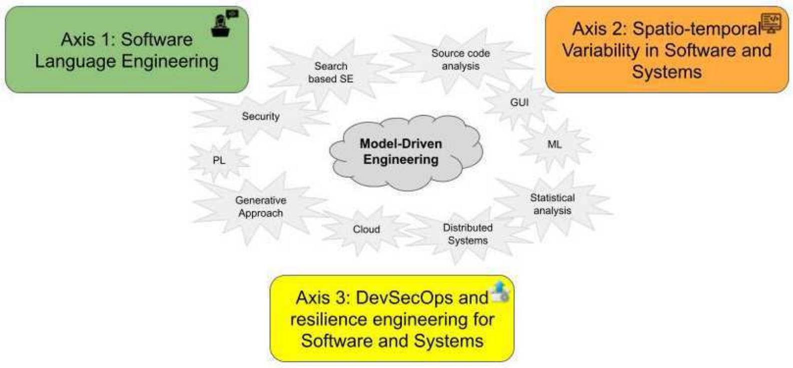
Axis 1: Software Language Engineering
Axis 2: Spatio-temporal Variability in Software and Systems
Axis 3: DevSecOps and resilience engineering for Software and Systems
Model-Driven Engineering
Search based SE
Source code analysis
GUI
ML
Statistical analysis
Distributed Systems
Cloud
Generative Approach
PL
Security

Figure 1: The three research axes of DIVERSE, relying on model driven engineering scientific background and leveraging several related fields

3.3.1 Axis #1: Software Language Engineering

Overall objective. The disruptive design of new, complex systems requires a high degree of flexibility in the communication between many stakeholders, often limited by the silo-like structure of the organization itself (cf. Conway's law). To overcome this constraint, modern engineering environments aim to: (i) better manage the necessary exchanges between the different stakeholders; (ii) provide a unique and usable place for information sharing; and (iii) ensure the consistency of the many points of view. Software languages are the key pivot between the *diverse* stakeholders involved, and the software systems they have to implement. Domain-Specific (Modeling) Languages enable stakeholders to address the *diverse* concerns through specific points of view, and their coordinated use is essential to support the socio-technical coordination across the overall software system life cycle.

Our perspectives on Software Language Engineering over the next period is presented in Figure 2 and detailed in the following paragraphs.

DSL Executability. Providing rich and adequate environments is key to the adoption of domain-specific languages. In particular, we focus on tools that support model and program execution. We explore the foundations to define the required concerns in language specification, and systematic approaches to derive environments (*e.g.,* IDE, notebook, design labs) including debuggers, animators, simulators, loggers, monitors, trade-off analysis, etc.

Modular & Distributed IDE. IDEs are indispensable companions to software languages. They are increasingly turning towards Web-based platforms, heavily relying on cloud infrastructures and forges. Since all language services require different computing capacities and response times (to guarantee a user-friendly experience within the IDE) and use shared resources (*e.g.,* the program), we explore new architectures for their modularization and systematic approaches for their individual deployment and dynamic adaptation within an IDE. To cope with the ever-growing number of programming languages, manufacturers of Integrated Development Environments (IDE) have recently defined protocols as a way to use and share multiple language services in language-agnostic environments. These protocols rely on

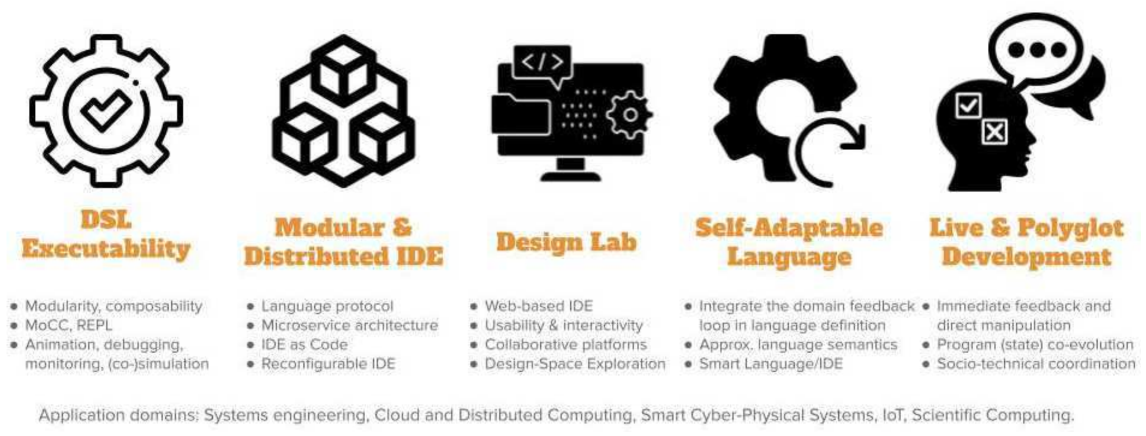
DSL Executability
• Modularity, composability
• MoCC, REPL
• Animation, debugging, monitoring, (co-)simulation
Modular & Distributed IDE
• Language protocol
• Microservice architecture
• IDE as Code
• Reconfigurable IDE
Design Lab
• Web-based IDE
• Usability & interactivity
• Collaborative platforms
• Design-Space Exploration
Self-Adaptable Language
• Integrate the domain feedback loop in language definition
• Approx. language semantics
• Smart Language/IDE
Live & Polyglot Development
• Immediate feedback and direct manipulation
• Program (state) co-evolution
• Socio-technical coordination
Application domains: Systems engineering, Cloud and Distributed Computing, Smart Cyber-Physical Systems, IoT, Scientific Computing.

Figure 2: Perspectives on Software Language Engineering (axis #1)

a proper specification of the services that are commonly found in the tool support of general-purpose languages, and define a fixed set of capabilities to offer in the IDE. However, new languages regularly appear offering unique constructs (e.g., DSLs), and which are supported by dedicated services to be offered as new capabilities in IDEs. This trend leads to the multiplication of new protocols, hard to combine and possibly incompatible (e.g., overlap, different technological stacks). Beyond the proposition of specific protocols, we will explore an original approach to be able to specify language protocols and to offer IDEs to be configured with such protocol specifications. IDEs went from directly supporting languages to protocols, and we envision the next step: *IDE as code*, where language protocols are created or inferred on demand and serve as support of an adaptation loop taking in charge of the (re)configuration of the IDE.

Design Lab. Web-based and cloud-native IDEs open new opportunities to bridge the gap between the IDE and collaborative platforms, *e.g.,* forges. In the complex world of software systems, we explore new approaches to reduce the distance between the various stakeholders (e.g., systems engineers and all those involved in specialty engineering) and to improve the interactions between them through an adapted tool chain. We aim to improve the usability of development cycles with efficiency, affordance and satisfaction. We also explore new approaches to explore and interact with the design space or other concerns such as software and systems human values or security, and provide facilities for trade-off analysis and decision making.

Live & Polyglot Development. As of today, polyglot development is massively popular and virtually all software systems put multiple languages to use, which not only complexifies their development, but also their evolution and maintenance. Moreover, as software grows more critical in new application domains (e.g., data analytics, health or scientific computing), it is crucial to ease the participation of scientists, decision-makers, and more generally non-software experts. Live programming makes it possible to change a program while it is running, by propagating changes on a program code to its run-time state. This effectively bridges the gulf of evaluation between program writing and program execution: the effects a change has on the running system are immediately visible, and the developer can take immediate action. The challenges at the intersection of polyglot and live programming have received little attention so far, and we envision a language design and implementation approach to specify domain-specific languages and their coordination, and automatically provide interactive domain-specific environments for live and polyglot programming.

Self-Adaptable Language. Over recent years, self-adaptation has become a concern for many software systems that operate in complex and changing environments. At the core of self-adaptation lies a feedback loop and its associated trade-off reasoning, to decide on the best course of action. However, existing software languages do not abstract the development and execution of such feedback loops for self-adaptable systems. Developers have to fall back to ad-hoc solutions to implement self-adaptable systems, often with wide-ranging design implications (e.g., explicit MAPE-K loop). Furthermore, existing software languages do not capitalize on monitored usage data of a language and its modeling environment. This hinders the continuous and automatic evolution of a software language based on feedback loops from the modeling environment and runtime software system. To address the aforementioned issues, we will explore the concept of Self-Adaptable Language (SAL) to abstract the feedback loops at both system and language levels.

3.3.2 Axis #2: Spatio-temporal Variability in Software and Systems

Overall objective. Leveraging our longstanding activity on variability management for software product lines and configurable systems covering *diverse* scenarios of use, we will investigate over the next period the impact of such a variability across the *diverse* layers, incl. source code, input/output data, compilation chain, operating systems and underlying execution platforms. We envision a better support and assistance for the configuration and optimisation (e.g., non-functional properties) of software systems according to this deep variability. Moreover, as software systems involve *diverse* artefacts (*e.g.,* APIs, tests, models, scripts, data, cloud services, documentation, deployment descriptors...), we will investigate their continuous co-evolution during the overall lifecycle, including maintenance and evolution. Our perspectives on spatio-temporal variability over the next period is presented in Figure 3 and is detailed in the following paragraphs.

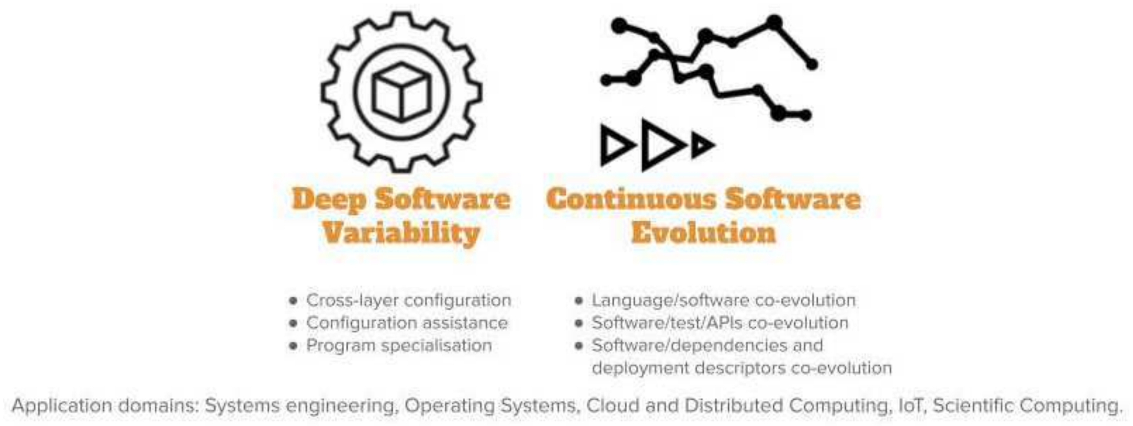
Deep Software Variability
Continuous Software Evolution
• Cross-layer configuration
• Configuration assistance
• Program specialisation
• Language/software co-evolution
• Software/test/APIs co-evolution
• Software/dependencies and deployment descriptors co-evolution
Application domains: Systems engineering, Operating Systems, Cloud and Distributed Computing, IoT, Scientific Computing.

Figure 3: Perspectives on Spatio-temporal Variability in Software and Systems (axis #2)

Deep Software Variability. Software systems can be configured to reach specific functional goals and non-functional performance, either statically at compile time or through the choice of command line options at runtime. We observed that considering the software layer only might be a naive approach to tune the performance of the system or to test its functional correctness. In fact, many layers (hardware, operating system, input data, etc.), which are themselves subject to variability, can alter the performance or functionalities of software configurations. We call *deep software variability* the interaction of all variability layers that could modify the behavior or non-functional properties of a software. Deep software variability calls to investigate how to systematically handle cross-layer configuration. The diversification of the different layers is also an opportunity to test the robustness and resilience of the software layer in multiple environments. Another interesting challenge is to tune the software for one specific executing environment. In essence, deep software variability questions the generalization of the configuration knowledge.

Continuous Software Evolution. Nowadays, software development has become more and more complex, involving various artefacts, such as APIs, tests, models, scripts, data, cloud services, documentation, etc., and embedding millions of lines of code (LOC). Recent evidence highlights continuous software evolution based on thousands of commits, hundreds of releases, all done by thousands of developers. We focus on the following essential backbone dimensions in software engineering: languages, models, APIs, tests and deployment descriptors, all revolving around software code implementation. We will explore the foundations of a multidimensional and polyglot co-evolution platform, and will provide a better understanding with new empirical evidence and knowledge.

3.3.3 Axis #3: DevSecOps and Resilience Engineering for Software and Systems

Overall objective. The production and delivery of modern software systems involves the integration of *diverse* dependencies and continuous deployment on *diverse* execution platforms in the form of large distributed socio-technical systems. This leads to new software architectures and programming models, as well as complex supply chains for final delivery to system users. In order to boost cybersecurity, we want to provide strong support to software engineers and IT teams in the development and delivery of secure and resilient software systems, ie. systems able to resist or recover from cyberattacks. Our perspectives on DevSecOps and Resilience Engineering over the next period are presented in Figure 4 and detailed in the following paragraphs.

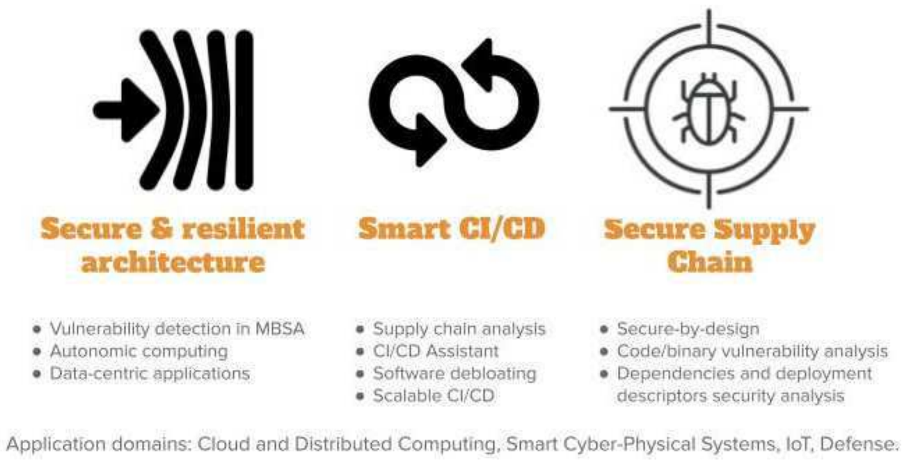
Secure & resilient architecture
Smart CI/CD
Secure Supply Chain
• Vulnerability detection in MBSA
• Autonomic computing
• Data-centric applications
• Supply chain analysis
• CI/CD Assistant
• Software debloating
• Scalable CI/CD
• Secure-by-design
• Code/binary vulnerability analysis
• Dependencies and deployment descriptors security analysis
Application domains: Cloud and Distributed Computing, Smart Cyber-Physical Systems, IoT, Defense.

Figure 4: Perspectives on DevSecOps and Resilience Eng. for Software and Systems (axis #3)

Secure & Resilient Architecture. Continuous integration and deployment pipelines are processes implementing complex software supply chains. We envision an explicit and early consideration of security properties in such pipelines to help in detecting vulnerabilities. In particular, we integrate the security concern in Model-Based System Analysis (MBSA) approaches, and explore guidelines, tools and methods to drive the definition of secure and resilient architectures. We also investigate resilience at runtime through frameworks for autonomic computing and data-centric applications, both for the software systems and the associated deployment descriptors.

Smart CI/CD. Dependencies management, Infrastructure as Code (IaC) and DevOps practices open opportunities to analyze complex supply chains. We aim at providing relevant metrics to evaluate and ensure the security of such supply chains, advanced assistants to help in specifying corresponding pipelines, and new approaches to optimize them (*e.g.,* software debloating, scalability...). We study how supply chains can actively leverage software variability and diversity to increase cybersecurity and resilience.

Secure Supply Chain. In order to produce secure and resilient software systems, we explore new secure-by-design foundations that integrate security concerns as first class entities through a seamless continuum from the design to the continuous integration and deployment. We explore new models, architectures, inter-relations, and static and dynamic analyses that rely on explicitly expressed security concerns to ensure a secure and resilient supply chain. We lead research on automatic vulnerability and malware detection in modern supply chains, considering the various artefacts either as white boxes enabling source code analysis (to avoid accidental vulnerabilities or intentional ones or code poisoning), or as black boxes requiring binary analysis (to find malware or vulnerabilities). We also conduct research activities in dependencies and deployment descriptors security analysis.

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 domains 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 to use software engineering approach, in particular, the use of SLE and approximate computing techniques.
- the field of large software systems such as the Linus kernel 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 Social and environmental responsibility

Not applicable

6 Highlights of the year

- Mathieu Acher junior member Institut Universitaire de France (IUF).

6.1 Awards

Best paper award at SPLC 2021 for Hugo Martin, Mathieu Acher, Juliana Alves Pereira, and Jean-Marc Jézéquel. A comparison of performance specialization learning for configurable systems (2021) [47]

Best paper award for Cassius Puodzius, Olivier Zendra, Annelie Heuser, Lamine Noureddine for "Accurate and Robust Malware Analysis through Similarity of External Calls Dependency Graphs (ECDG)" in ARES-IWCC 2021 [51]

7 New software and platforms

Not applicable

7.1 New software

7.1.1 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

7.1.2 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.
- Monilog, an extension for monitoring and logging executable domain-specific 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

Contact: Benoît Combemale

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

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

7.1.3 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/

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

7.1.4 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

Contact: Benoît Combemale

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

7.1.5 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

Contact: Benjamin Danglot

Participants: Benoit Baudry, Martin Monperrus, Benjamin Danglot

Partner: KTH Royal Institute of Technology

7.1.6 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

7.1.7 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

7.1.8 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

Contact: Benoit Baudry

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

Partner: KTH Royal Institute of Technology

7.1.9 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

7.1.10 SE-PAC

Name: Self-Evolving-PAcker Classifier

Keywords: Packers, Malware, Obfuscation, Classification, Incremental clustering

Functional Description: SE-PAC is a Self-Evolving PAcker Classifier software that identifies the class of the packer used to pack a (malicious) Win32 executable file. It has the the ability to identify both known and unknown packer classes, as well as self-updating its packer knowledge with the predicted packer classes.

SE-PAC relies on incremental clustering in a semi-supervised fashion. It is composed of an offline phase and an online phase. In the offline phase, a set of features is extracted from a collection of packed binaries provided with their ground truth labels, then a density-based clustering algorithm (DBSCAN) is used to group similar packers together into clusters with respect to a distance measure. In this step, the similarity threshold is tuned in order to form the clusters that fit the best with the set of labels provided. In the online phase, SE-PAC reproduces the same operations of feature extraction and distance calculation, then uses a customized version of the incremental clustering algorithm DBSCAN in order to classify incoming packed samples, either in known packer classes by extending existing clusters or new packer classes by forming new clusters, or temporarily leave them unclassified (see notion of noise of DBSCAN). The clusters formed after each update serve as a baseline for SE-PAC to self-evolve.

Finally, SE-PAC includes a post-clustering selection strategy that extracts a set of relevant samples from the clusters found in order to reduce the cost of the post-clustering packer processing.

Release Contributions: Version 1.1 brings: - Implementation of so-called SRPs (Scattered Representative Points) for reducing the complexity of the incremental process. - Implementation of a post-grouping strategy for the selection of relevant samples from the constituted groups. - Better modularity of the code.

Publication: hal-03149211

Contact: Olivier Zendra

7.2 New platforms

Not applicable

Participants no one in the team.

8 New results

8.1 Results for Axis #1: Software Language Engineering

Participants Olivier Barais, Johann Bourcier, Benoit Combemale, Jean-Marc Jézéquel, Gurvan Leguernic, Gunter Mussbacher, Noel Plouzeau, Didier Vojtisek.

8.1.1 Foundations of Software Language Engineering

Self-Adaptable Languages Over recent years, self-adaptation has become a concern for many software systems that have to operate in complex and changing environments. At the core of self-adaptation, there is a feedback loop and associated trade-off reasoning to decide on the best course of action. However, existing software languages do not abstract the development and execution of such feedback loops for self-adaptable systems. Developers have to fall back to ad-hoc solutions to implement self-adaptable systems, often with wide-ranging design implications (e.g., explicit MAPE-K loop). Furthermore, existing software languages do not capitalize on monitored usage data of a language and its modeling environment. This hinders the continuous and automatic evolution of a software language based on feedback loops from the modeling environment and runtime software system. To address the aforementioned issues, in [39] we introduce the concept of Self-Adaptable Language (SAL) to abstract the feedback loops at both system and language levels. We propose L-MODA (Language, Models, and Data) as a conceptual reference framework that characterizes the possible feedback loops abstracted into a SAL. To demonstrate SALs, we present emerging results on the abstraction of the system feedback loop into the language semantics. We report on the concept of Self-Adaptable Virtual Machines as an example of semantic adaptation in a language interpreter and present a roadmap for SALs.

As a first contribution in this vision, in [38] we present SEALS, a framework for building self-adaptive virtual machines for domain specific languages. This framework provides first-class entities for the language engineer to promote domain-specific feedback loops in the definition of the DSL operational semantics. In particular, the framework supports the definition of (i) the abstract syntax and the semantics of the language as well as the correctness envelope defining the acceptable semantics for a domain concept, (ii) the feedback loop and associated trade-off reasoning, and (iii) the adaptations and the predictive model of their impact on the trade-off. We use this framework to build three languages with self-adaptive virtual machines and discuss the relevance of the abstractions, effectiveness of correctness envelopes, and compare their code size and performance results to their manually implemented counterparts. We show that the framework provides suitable abstractions for the implementation of self-adaptive operational semantics while introducing little performance overhead compared to a manual implementation.

IDE as Code: Reifying Language Protocols as First-Class Citizens To cope with the ever-growing number of programming languages, manufacturers of Integrated Development Environments (IDE) have recently defined protocols as a way to use and share multiple language services (e.g., auto-completion, type checker, language runtime) in language-agnostic environments (i.e., the user interface provided by the IDE): the most notable are the Language Server Protocol (LSP) for textual editors, and the Debug Adapter Protocol (DAP) for debugging facilities. These protocols rely on a proper specification of the services that are commonly found in the tool support of general-purpose languages, and define a fixed set of capabilities to offer in the IDE. However, new languages appear regularly, offering unique constructs (e.g., Domain-Specific Languages), and supported by dedicated services to be offered as new capabilities in IDEs. This trend leads to the multiplication of new protocols, hard to combine and possibly incompatible (e.g., overlap, use of different technological stacks). Beyond the proposition of specific protocols, the goal of [37] is to highlight the importance of being able to specify language protocols and to offer IDEs to be configured with such protocol specifications. We present our vision by discussing the main concepts for the specification of language protocols, and an approach that can make use of these specifications in order to deploy an IDE as a set of coordinated, individually deployed, language capabilities (e.g., microservice choreography). IDEs went from directly supporting languages to protocols, and we envision the next step in [37]: IDE as Code, where language protocols are created or inferred on demand and serve as support of an adaptation loop taking charge of the (re)configuration of the IDE.

DataTime: a Framework to smoothly Integrate Past, Present and Future into Models Models at runtime have been initially investigated for adaptive systems. Models are used as a reflective layer of the current state of the system to support the implementation of a feedback loop. More recently, models at runtime have also been identified as key for supporting the development of full-fledged digital twins. However, this use of models at runtime raises new challenges, such as the ability to seamlessly interact with the past, present and future states of the system. In [46], we propose a framework called DataTime to implement models at runtime which capture the state of the system according to the dimensions of both time and space, here modeled as a directed graph where both nodes and edges bear local states (ie. values of properties of interest). DataTime provides a unifying interface to query the past, present and future (predicted) states of the system. This unifying interface provides (i) an optimized structure of the time series that capture the past states of the system, possibly evolving over time, (ii) the ability to get the last available value provided by the system's sensors, and (iii) a continuous micro-learning over graph edges of a predictive model to make it possible to query future states, either locally or more globally, thanks to a composition law. The framework has been developed and evaluated in the context of the Intelligent Public Transportation Systems of the city of Rennes (France). This experimentation has demonstrated how DataTime can deprecate the use of heterogeneous tools for managing data from the past, the present and the future, and facilitate the development of digital twins.

A Hitchhiker's Guide to Model-Driven Engineering for Data-Centric Systems In [27], we introduce the MODA framework as a conceptual reference framework that provides the foundations for identifying the various models and their respective roles within any model-based system development life-cycle. It is intended to be a guide to organize the various models in datacentric socio-technical systems.

8.1.2 DSL for Scientific Computing

When Scientific Software Meets Software Engineering The development of scientific software relies on the collaboration of various stakeholders for the scientific computing and software engineering activities. Computer languages have an impact on both activities and related concerns, as well as on the engineering principles required to ensure the development of reliable scientific software. The more general-purpose the language is — with low-level, computing-related, system abstractions — the more flexibility it will provide, but also the more rigorous engineering principles and Validation & Verification (V&V) activities it will require from the language user. In [32], we investigate the different levels of abstraction, linked to the diverse artifacts of the scientific software development process, a software language can propose, and the V&V facilities associated to the corresponding level of abstraction the language can provide to the user. We aim to raise awareness among scientists, engineers and language providers on their shared responsibility in developing reliable scientific software.

Monilogging for Executable Domain-Specific Languages Runtime monitoring and logging are fundamental techniques for analyzing and supervising the behavior of computer programs. However, supporting these techniques for a given language induces significant development costs that can hold language engineers back from providing adequate logging and monitoring tooling for new domain-specific modeling languages. Moreover, runtime monitoring and logging are generally considered as two different techniques: they are thus implemented separately which makes users prone to overlooking their potentially beneficial mutual interactions. In [41], we propose a language-agnostic, unifying framework for runtime monitoring and logging and demonstrate how it can be used to define loggers, runtime monitors and combinations of the two, aka. moniloggers. We provide an implementation of the framework that can be used with Java-based executable languages, and evaluate it on two implementations of the NabLab interpreter, leveraging in turn the instrumentation facilities offered by Truffle, and those offered by AspectJ.

8.1.3 Design lab

In the context of the research on the design lab axis, we worked this year on new abstractions of interaction and we also studied how to build a simulation tool combining modeling techniques and data analysis in the field of transportation with an industrial partner.

Interacto: A Modern User Interaction Processing Model Since most software systems provide their users with interactive features, building user interfaces (UI) is one of the core software engineering tasks. It consists in designing, implementing and testing ever more sophisticated and versatile ways for users to interact with software systems, and safely connecting these interactions with commands querying or modifying their state. However, most UI frameworks still rely on a low level model, the bare bone UI event processing model. This model was suitable for the rather simple UIs of the early 80's (menus, buttons, keyboards, mouse clicks), but now exhibits major software engineering flaws for modern, highly interactive UIs. These flaws include lack of separation of concerns, weak modularity and thus low reusability of code for advanced interactions, as well as low testability. To mitigate these flaws, we propose Interacto as a high level user interaction processing model [25]. By reifying the concept of user interaction, Interacto makes it easy to design, implement and test modular and reusable advanced user interactions, and to connect them to commands with built-in undo/redo support. To demonstrate its applicability and generality, we briefly present two open source implementations of Interacto for Java/JavaFX and TypeScript/Angular. We evaluate Interacto interest (1) on a real world case study, where it has been used since 2013, and with (2) a controlled experiment with 44 master students, comparing it with traditionnal UI frameworks.

Impact of Data Cleansing for Urban Bus Commercial Speed Prediction Public Transportation Information Systems (PTIS) are widely used for public bus services amongst cities in the world. These systems gather information about trips, bus stops, bus speeds, ridership, etc. This massive data is an inviting source of information for machine learning predictive tools. However, it most often suffers from quality deficiencies, due to multiple data sets with multiple structures, to different infrastructures using incompatible technologies, to human errors or hardware failures. In this work [33, 45], we consider the impact of data cleansing on a classical machine-learning task: predicting urban bus commercial speed. We show that simple, transport specific business and quality rules can drastically enhance data quality, whereas more sophisticated rules may offer little improvements despite a high computational cost.

8.2 Results for Axis #2: Spatio-temporal Variability in Software and Systems

Participants Mathieu Acher, Arnaud Blouin, Jean-Marc Jezequel, Djamel Eddine Khelladi, Olivier Zendra
.

In [24] we perform a systematic review of the rich literature related to the learning of software configuration spaces. We report on the different application objectives (e.g., performance prediction, configuration optimization, constraint mining), use cases, targeted software systems and application domains. We review the various strategies employed to gather a representative and cost-effective sample. We describe automated software techniques used to measure functional and non-functional properties of configurations. We classify machine learning algorithms and how they relate to the pursued application. Finally, we also describe how researchers evaluate the quality of the learning process.

In [34], we apply learning techniques to the Linux kernel. With now more than 15,000 configuration options, including more than 9,000 just for the x86 architecture, the Linux kernel is one of the most complex configurable open-source systems ever developed. If all these options were binary and independent, that would indeed yield 2^{15000} possible variants of the kernel. Of course not all options are independent (leading to fewer possible variants), but some of them have tri-states values: yes, no, or module instead of simply boolean values (leading to more possible variants). The Linux kernel is mentioned in numerous papers about configurable systems and machine learning for motivating the problem and the underlying approach. However, only a few works truly explore such a huge configuration space. In this line of work, we take up the Linux challenge either for configurations' bug prevention or for predicting the binary size of a configured kernel. We also design a learning technique capable of transferring a prediction model among **variants and versions** of Linux [34].

In [47] (**Best paper award at SPLC 2021**), we compare six learning techniques: three variants of decision trees (including a novel algorithm) with and without the use of model-based feature selection. We first perform a study on 8 configurable systems considered in previous related works and show that the accuracy reaches more than 90%, and that feature selection can improve the results in the majority of cases. We then perform a study on the Linux kernel and show that these techniques perform as well as on the other systems. Overall, our results show that there is no one-size-fits-all learning variant (though high accuracy can be achieved): we present guidelines and discuss tradeoffs.

In [43, 44], we start from the fact that many software projects are configurable through compile-time options (e.g., using ./configure) and also through run-time options (e.g., command-line parameters, fed to the software at execution time). Several works have shown how to predict the effect of run-time options on performance. However it is yet to be studied how these prediction models behave when the software is built with different compile-time options. For instance, is the best run-time configuration always the best w.r.t. the chosen compilation options? We investigate the effect of compile-time options on the performance distributions of 4 software systems. There are cases where the compiler layer effect is linear which is an opportunity to generalize performance models or to tune and measure runtime performance at lower cost. We also prove there can exist an interplay by exhibiting a case where compile-time options significantly alter the performance distributions of a configurable system.

In [42] we notice that many layers (hardware, operating system, input data, etc.), themselves subject to variability, can alter performances of software configurations. For instance, the configurations' 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. 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 debugging, and the testing 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.

In [40] we explored the challenge of code and tests co-evolution. Version Control Systems are key elements of modern software development. They provide the history of software systems, serialized as lists of commits. Practitioners may rely on this history to understand and study the evolution of software systems, including the co-evolution amongst strongly coupled development artifacts such as production code and their tests. However, a precise identification of code and test co-evolutions requires practitioners to manually untangle a spaghetti of evolutions. In this work, we propose an automated approach for detecting co-evolutions between code and tests, independently of the commit history. The approach creates a sound knowledge base of code and test co-evolutions that practitioners can use for various purposes in their projects. We conducted an empirical study on a curated set of 45 open-source systems having Git histories. Our approach exhibits a precision of 100 % and an underestimated recall of 37.5 % in detecting the code and test co-evolutions. Our approach also spotted different kinds of code and test co-evolutions, including some of those researchers manually identified in previous work.

In [31] we explore the direction of change propagation for models repair. 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 work exists 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 speak of consistent change propagation. This work 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 work of how developers handle changes by always computing the sequence of repairs resulting from the change propagation.

In [30] we explore the challenge of computing concrete executable repairs for models. Software models, often comprised of interconnected diagrams, change continuously, and developers often fail in keeping these diagrams consistent. Detecting inconsistencies quickly and efficiently is now 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. This work presents a novel approach that provides concrete repair choices based on values from the inconsistent models. Thus, 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.

8.3 Results for Axis #3: DevSecOps and Resilience Engineering for Software and Systems

Participants	Mathieu Acher, Olivier Barais, Arnaud Blouin, Stephanie Challita, Benoit Combemale, Jean-Marc Jezequel, Olivier Zendra.

The work done in 2021 for axis #3 draws from previous works in software engineering and security from both DiverSE and the former TAMIS team. The first area of investigation deals with microservices, seen as a mean to increase maintainability, security and resilience.

In [36], we explore going **From Monolithic to Microservice Architecture: The Case of Extensible and Domain-Specific IDEs**. Integrated Development Environments (IDEs) are evolving towards cloud-native applications with the aim to relocate the language services provided by an IDE on distant servers. Existing research works focus on the overall migration process to handle more efficiently their specific requirements. However, the microservicization of legacy monolithic applications is still highly dependent on the specific properties of the application of interest. In this work, we report our experiment on the microservicization process of the Cloud-Based graphical modeling workbench Sirius Web. We aim at identifying the technical challenges related to applications with similar properties, and we provide insights for practitioners to migrate their similar applications towards microservices. We discuss the main lessons learned and identify the underlying challenges to be further addressed by the community.

Further works pertain to program analysis and program behavioral characterization, to detect malwares or packed malwares.

In [51] **(Best Paper Award at IWCC 2021)**, we addressed **Accurate and Robust Malware Analysis through Similarity of External Calls Dependency Graphs (ECDG)**. Malware is a primary concern in cybersecurity, being one of the attacker's favorite cyberweapons. Over time, malware evolves not only in complexity but also in diversity and quantity. Malware analysis automation is thus crucial. In this work we present ECDGs, a shorter call graph representation, and a new similarity function that is accurate and robust. Toward this goal, we revisit some principles of malware analysis research to define basic primitives and an evaluation paradigm addressed for the setup of more reliable experiments. Our benchmark shows that our similarity function is very efficient in practice, achieving speedup rates of 3.30x and 354, 11x wrt. radiff2 for the standard and the cache-enhanced implementations, respectively. Our evaluations generate clusters that produce almost unerring results-homogeneity score of 0.983 for the accuracy phase-and marginal information loss for a highly polluted dataset-NMI score of 0.974 between initial and final clusters of the robustness phase. Overall, ECDGs and our similarity function enable autonomous frameworks for malware search and clustering that can assist human-based analysis or improve classification models for malware analysis.

In [49], we present **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. 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).

On the later topic, Lamine Noureddine successfully defended on 21/12/2021 his PhD thesis in computer science at Université Rennes 1 on "**Packing detection and classification relying on machine learning to stop malware propagation**" [60]

In [28], we investigating Machine Learning Algorithms for Modeling SSD I/O Performance for Container-based Virtualization. It is used create resources-usage finger print and detect security issue in Cloud computing domain. Indeed, one of the cornerstones of the cloud provider business is to reduce hardware resources cost by maximizing their utilization. This is done through smartly sharing processor, memory, network and storage, while fully satisfying SLOs negotiated with customers. For the storage part, while SSDs are increasingly deployed in data centers mainly for their performance and energy efficiency, their internal mechanisms may cause a dramatic SLO violation. In effect, we measured that I/O interference may induce a 10x performance drop. We are building a framework based on autonomic computing which aims to achieve intelligent container placement on storage systems by preventing bad I/O interference scenarios. One prerequisite to such a framework is to design SSD performance models that take into account interactions between running processes/containers, the operating system and the SSD. These interactions are complex. In this work, we investigate the use of machine learning for building such models in a container based Cloud environment. We have investigated five popular machine learning algorithms along with six different I/O intensive applications and benchmarks. We analyzed the prediction accuracy, the learning curve, the feature importance and the training time of the tested algorithms on four different SSD models. Beyond describing modeling component of our framework, this work aims to provide insights for cloud providers to implement SLO compliant container placement algorithms on SSDs. Our machine learning-based framework succeeded in modeling I/O interference with a median Normalized Root-Mean-Square Error (NRMSE) of 2.5%

In [50], we explore testing techinques to improve distributed software resilience. Indeed, detecting faults in distributed network systems is challenging because of their complexity, but this is required to evaluate and improve their reliability. We propose ChaT, a testing and evaluation methodology under system reconfigurations and perturbations for distributed network systems, to evaluate QoS reliability by discriminating safe and failure-prone behaviors from different testing scenarios. Motivated by metamorphic testing technique that removes the burden of defining software oracles, we propose some metamorphic relationships that correlate system inputs and outputs to find patterns in executions. Classification techniques based on machine learning (principal component analysis and support vector machine) are used to identify system states and validate the proposed metamorphic relationships. These metamorphic relationships are also used to help anomaly detection. We verify this with several anomaly detection techniques (isolation forest, one-class SVM, local outlier factor, and robust covariance) that categorize experiments belonging to either safe or failure-prone states. We apply ChaT to a video streaming application use case. The simulation results show the effectiveness of ChaT to achieve our goals: identifying execution classes and detecting failure-prone experiments based on metamorphic relationships with high level of statistical scores.

Digital twins are a very promising avenue to realize secure and resilient architectures and systems.

In [29], we study **Conceptualizing Digital Twins**. Digital Twins are an emerging concept which is gaining importance in several fields. It refers to a comprehensive software representation of an actual system, which includes structures, properties, conditions, behaviours, history and possible futures of that system through models and data to be continuously synchronized. Digital Twins can be built for different purposes, such as for the design, development, analysis, simulation, and operations of non-digital systems in order to understand, monitor, and/or optimize the actual system. To realize Digital Twins, data and models originated from diverse engineering disciplines have to be integrated, synchronized, and managed to leverage the benefits provided by software (digital) technologies. However, properly arranging the different models, data sources, and their relations to engineer Digital Twins is challenging. We, therefore, propose a conceptual modeling framework for Digital Twins that captures the combined usage of heterogeneous models and their respective evolving data for the etwin's entire life-cycle.

In [26], we explore the use and OCCI and TOSCA, two standard in the domain of cloud computing as a building block for buildng a digital twin of modern applications. 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 Infrastructureas-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. 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.

These achievements constitute basic blocks upon which we will continue building our research and systems, extending for example the applicability to secure supply chains.

9 Bilateral contracts and grants with industry

Participants all the team.

9.1 Bilateral contracts with industry

ADR Nokia

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

SLIMFAST

- Partners: DGA
- Dates: 2021-2022
- Abstract: Debloating software variability for improving non-functional properties (e.g. security)

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: 2019-2021
- Abstract: Demonstrator illustrates 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: 2017-2021
- Abstract: Web engineering for domain-specific modeling languages, Fabien Coulon's PhD Cifre project.

OKWind

- Partners: UR1/OKWind
- Dates: 2017-2021
- 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.

10 Partnerships and cooperations

Participants all the team.

10.1 International initiatives

10.1.1 Associate Teams in the framework of an Inria International Lab or in the framework of an Inria International Program

ALE

Title: Agile Language Engineering

Duration: 2017 ->

Coordinator: Tijs van der Storm (storm@cwi.nl)

Partners:

- CWI

Inria contact: Benoît Combemale

Summary: ALE contributes 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. Second, we aim to provide more flexibility to language users by ensuring interoperability between different DSLs and offering live feedback about how the model or program behaves while it is being edited (aka. live programming/modeling)

RESIST-EA

Title: Resilient Software Science

Duration: 2021 ->

Coordinator: Mathieu Acher and Arnaud Gotlieb (arnaud@simula.no)

Partners:

- SIMULA

Inria contact: Mathieu Acher

Summary: The goal of the RESIST associate team is to explore the science of resilient software by foundational work on advanced a priori testing methods such as metamorphic testing and a posteriori continuous improvements through digital twins. resist web site

10.2 International research visitors

10.2.1 Visits of international scientists

Inria International Chair We are glad to host Prof. Gunter Mussbacher (McGill University) on Inria International Chair in the DiverSE team.

Other international visits to the team

10.2.2 Visits to international teams

Sabbatical programme

Research stays abroad

10.3 European initiatives

10.3.1 FP7 & H2020 projects

TeamPlay (653)

Title: Title: TeamPlay: Time, Energy and security Analysis for Multi/Many-core heterogeneous PLAtforms

Duration: 01/2018 - 06/2021

Coordinator: Olivier Zendra (Olivier.Zendra@inria.fr), Inria

Partners:
- Absint Angewandte Informatik Gmbh (Germany)
- Institut National De Recherche en Informatique et Automatique (France)
- Sky-Watch A/S (Danemark)
- Syddansk Universitet (Danemark)
- Systhmata Ypologistikis Orashs Irida Labs Ae (Greece)
- Technische Universität Hamburg-Harburg (Germany)
- Thales Alenia Space Espana (Spain)
- Universiteit Van Amsterdam (Netherlands)
- University Of Bristol (UK)
- University Of St Andrews (UK)
- Secure-Ic Sas (France) untill 13/10/2020.

Inria contact: Olivier Zendra (Olivier.Zendra@inria.fr)

Summary: As mobile applications, the Internet of Things, and cyber-physical systems become more prevalent, so there is an increasing focus on energy efficiency of multicore computing applications. At the same time, traditional performance issues remain equally important. Increasingly, software designs need to find the best performance within some energy budget — often while also respecting real-time or other constraints, which may include security, data locality or system criticality — and while simultaneously optimising the usage of the available hardware resources. While parallel multi- core/manycore hardware can, in principle, ameliorate energy problems, and heterogeneous systems can help to find a good balance between performance and energy usage, there were no effective analyses beyond user-guided simulations that could reliably predict energy usage for parallel systems, whether alone or in combination with timing information. In order to create energy-efficient parallel software, programmers need to be actively engaged in decisions about energy usage, performance etc. rather than passively informed about their effects. This extends to design-time as well as to implementation-time and run-time. In order to address this fundamental challenge, TeamPlay took a radically new approach: by exploiting new and emerging ideas that allow extra-functional properties to be deeply embedded within their programs, programmers can be empowered to directly treat energy usage, time, and other key system properties, as first-class citizens in their parallel software.
The TeamPlay project aimed to develop new, formally-motivated, techniques that allow execution time, energy usage, security (ETS), and other important non-functional properties (NFP) of parallel software to be treated effectively, and as first- class citizens. We build this into a toolbox to develop parallel software for low-energy systems, as required by the internet of things, cyber-physical systems etc. The TeamPlay approach allows programs to reflect directly on their own ETS, etc., as well as enables the developer to reason about both the functional and the NFP of their software at source code level. Our success ensures significant progress on a pressing problem of major industrial importance: how to effectively manage energy consumption for parallel systems while maintaining the right balance with other important software metrics, including time, security, code size/complexity, etc. The project brought together leading industrial and academic experts in parallelism, energy modeling/transparency, worst-case execution time analysis, non-functional property analysis, compilation, security, and task coordination. Results were evaluated with industrial use cases taken from the computer vision, satellites, flying drones, medical domains.

10.4 National initiatives

10.4.1 ANR

MC-Evo2 ANR JCJC

- Coordinator: Djamel E. Khelladi
- DiverSE, CNRS/IRISA Rennes
- Dates: 2021-2025
- Abstract: Software maintenance represents 40% to 80% of the total cost of developing software. On 65 projects, an IT company reported a cost of several million dollars, with a 25% higher cost on complex projects. Nowadays, software evolves frequently with the philosophy “Release early, release often” embraced by IT giants like the GAFAM, thus making software maintenance difficult and costly. Developing complex software inevitably requires developers to handle multiple dimensions, such as APIs to use, tests to write, models to reason with, etc. When software evolves, a co-evolution is usually necessary as a follow-up, to resolve the impacts caused by the evolution changes. For example, when APIs evolve, code must be co-evolved, or when code evolves, its tests must be co-evolved. The goals of this project are to: 1) address these challenges from a novel perspective, namely a multidimensional co-evolution approach, 2) investigate empirically the multidimensional co-evolution in practice in GitHub, Maven, and Eclipse, 3) automate and propagate the multidimensional co-evolution between the software code, APIs, tests, and models.

VaryVary ANR JCJC

- Coordinator: Mathieu Acher
- DiverSE, Inria/IRISA Rennes
- Dates: 2017-2021
- Abstract: Most modern software systems (operating systems such as Linux, Web browsers such as Firefox or Chrome, video encoders such as 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 variability vary. 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).

10.4.2 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 packet 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.

10.4.3 DGAC

OneWay

- Coordinator: Airbus
- Partners: Airbus, Dassault Aviation, Liebherr Aerospace, Safran Electrical Power, Safran Aerotechnics, Thales, Altran Technologies, Cap Gemini, Sopra Steria, CIMPA, IMT Mines Ales, University of Rennes 1, ENSTA Bretagne, and PragmaDev.
- Dates: 2021-2022
- Abstract: The ONEWAY project aims at maturing digital functional bricks for the following capacities: 1) Digitalization, MBSE modeling and synthetic analysis by substitution model, of all the information and under all the points of view necessary for the design and validation across an extended enterprise of the complete aircraft system and at all its levels of decomposition, 2) Generic and instantiable configuration management throughout the life cycle, on products and their support systems, in product lines or on aircraft programs, interactively in the context of an extended enterprise, 3) Decision support for launching, then controlling and steering a Product Development Plan interactively in the context of an extended enterprise, and 4) Helping the efficiency of IVVQ activities: its operations, its testing and data processing resources, its ability to perform massive testing.

10.5 Regional initiatives

IPSCO

- Coordinator: IPSCO
- Partners: UR1, Jamespot, Logpickr.
- Dates: 2021-2023
- Abstract: The IPSCO (Intelligent Support Processes and Communities) project aims to develop a new customer support platform for digital companies and public services. Both by implementing intelligent mechanisms for filtering and processing public requests (customers and partners) and by providing analysis on the organization of the response to these requests. In addition, in order to provide a robust product for small teams, the solution will allow the effective management of expert user communities to foster their autonomy and the emergence of best practices. The integration of this product into an enterprise collaborative platform redefines support by placing it at the heart of its activity. This paradigm shift is part of a fundamental phenomenon that is currently sweeping through companies.

11 Dissemination

Participants all the team.

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

General chair, scientific chair

Member of the organizing committees

- The DIVERSE group organized ICPE'21 (ACM/SPEC International Conference on Performance Engineering), 2021. Most of the permanent staff was involved.
- Olivier Barais organized the french national days on cloud computing Journées Cloud 2021

11.1.2 Scientific events: selection

Chair of conference program committees

Member of the conference program committees

- Mathieu Acher:
  - PC member for SPLC 2021 Research Track and VaMoS 2021.
  - Member of the Prix de thèse Gilles Khan 2021
- Arnaud Blouin:
  - ACM/SIGAPP Symposium on Applied Computing (SAC), software engineering track, 2021;
  - ACM SIGCHI symposium on Engineering interactive computing systems (EICS 2021), 2021;
  - International Workshop on Human Factors in Modeling at MODELS'2021(HuFaMo), 2021
- Jean-Marc Jézéquel:
  - 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2021, Madrid, Spain
  - SPLC 2021 Industry Track
- Benoit Combemale:
  - Program board member for MODELS'21
  - PC member for RE'21
  - PC member for ISEC'21
  - PC member for ECMFA'21
  - PC member for EASE'21
  - PC member for EduSymp at MODELS'21
  - PC member for DevOps4CPS-Testing at ICST'21
  - PC member for the Eclipse SAAM Mobility 2021 conference
- Olivier Zendra

  - 16th ACM International Workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems (ICOOOLPS 2021)

- Olivier Barais
  - MODELS 2021: ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)

11.1.3 Journal

Member of the editorial boards

Jean-Marc Jézéquel

- Associate **Editor in Chief** of *IEEE Computer*
- Associate **Editor in Chief** of the *Journal on Software and System Modeling*
- Member of the editorial board of *Journal on Software and Systems*

Benoit Combemale

- Deputy **Editor in Chief** of *Journal of Object Technology* (JOT)
- Member of the editorial board of the *Journal on Software and System Modeling* (SoSyM)
- Member of the editorial board of the *Software Quality Journal* (SQJ)
- Member of the editorial board of the *Journal of Computer Languages* (COLA)
- Member of the editorial board of the *Journal on Science of Computer Programming* (SCP, Advisory Board of the Software Section)

Reviewer - reviewing activities
Team members regularly review for the main journals in the field, namely TSE, Sosym, JSS, Jot, SPE, IEEE Software, . . .

11.1.4 Invited talks

Jean-Marc Jézéquel:

- Keynote at 17th European Conference on Modelling Foundations and Applications, Co-located with STAF 2021. Bergen, Norway, June 2021.
- Keynote at ICPE workshop on Performance Modeling. Rennes, France, April 2021

11.1.5 Leadership within the scientific community

- Mathieu Acher:
  - Member of the Steering committee of SPLC conference
  - Member of the Steering committee of VaMoS conference
- Arnaud Blouin:
  - Founding member and co-organiser of the French GDR-GPL research action on Software Engineering and Human-Computer Interaction (GL-IHM).
- Benoit Combemale:
  - Chair of the Steering Committee of the ACM SIGPLAN Intl. Conference on Software Language Engineering

  - Funding Member of the Advisory Board of the GEMOC Initiative: On the Globalization of Modeling Languages
  - Steering Committee member of the Modeling Language Engineering and Execution (MLE) workshop
  - Founding member of the International Workshop on Model-Driven Engineering of Digital Twins (ModDiT'21)

- Jean-Marc Jézéquel:
  - Vice-President Informatics Europe
  - Executive Board of the GDR GPL of CNRS
  - Chair of the Software Product Line Most Influencial Paper Award Committee
  - Scientific Advisory Board of Center for Cyber Defense and Information Security, KTH, Sweden

- Olivier Zendra:
  - founder and a member of the Steering Committee of ICOOOLPS (International Workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems).
  - Scientific Coordinator of EU H2020 TeamPlay project
  - Member of the EU HiPEAC CSA project Steering Committee
  - Member of the HiPEAC Vision Editorial Board

11.1.6 Scientific expertise

- Arnaud Blouin: reviewer for the "Initiatives de Recherche à Grenoble Alpes" (IRGA) french research project call.
- Olivier Zendra: reviewer for the PhD thesis grants 2021 call of the "Pôle de recherche cyber" (French Cybersecurity Research Center); reviewer of the "Appel à projets génériques" 2021 of ANR (generic research projets call from French National Research Agency); reviewer for the HiPEAC collaboration Grants 2022; scientific CIR/JEI expert for the MESRI
- Olivier Barais:
  - scientific reviewer for WASP NEST Sweden research program
  - member of the scientific board of Pole de compétitivité Image et Réseau
  - scientific reviewer for FRQNT- Programme Samuel-de-Champlain (Quebec)
  - external expert for the H2020 ENACT project
  - scientific expertise for DGRI international program (20 proposals per year)
  - AutoActive board of expert (Norvegian project)

11.1.7 Research administration

Olivier Zendra is a Member of Inria Evaluation Committee.

Olivier Barais lead the jury for hiring a full professor position at University of Rennes 1.

11.2 Teaching - Supervision - Juries

11.2.1 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 Computer Science department and member of the management board at the ESIR engineering school in Rennes until 08/2021, and Benoit Combemale took the responsability afterward. 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.

11.2.2 Supervision

PhD defenses

Hugo Martin successfully defended his PhD thesis entitled “Machine Learning for Performance Modelling on Colossal Software Configuration Space”. Mathieu Acher and Jean-Marc Jézéquel were the supervisors of the thesis.

Lamine Noureddine successfully defended on 21/12/2021 his PhD thesis in computer science at Université Rennes 1 on “Packing detection and classification relying on machine learning to stop malware propagation” [60]. Olivier Zendra has been co-director of this thesis.

Antoine Cheron successfully defended on 17/12/2021 his PhD thesis in computer science at Université Rennes 1 on “Conception, maintenance et évolution non-cassante des API REST”. Olivier Barais and Johann Bourcier have been co-director of this thesis.

Alexandre Rio successfully defended on 26/02/2021 his PhD thesis in computer science at Université Rennes 1 on “Optimisation de l’utilisation des énergies renouvelables : un jumeau numérique pour les microgrilles”. Olivier Barais has been co-director of this thesis.

11.2.3 Juries

- Mathieu Acher was in the jury (reviewer) of Marcel Heinz (University of Koblenz-Landau) PhD thesis "Knowledge Engineering for Software Languages and Software Technologies"

- Jean-Marc Jézéquel was in the jury of Mathieu Acher’s HDR of University of Rennes on "Modelling, Reverse Engineering, and Learning Software Variability"

- Olivier Zendra was in the jury of Inès BEN EL OUAHMA PhD thesis of SORBONNE UNIVERSITE on Robustness analysis and assembly codes securisation against physical attacks.

- Olivier Barais was in the jury (reviewer) of Sophie Ebersold’s HDR of University of Toulouse on: "Modélisation des systèmes complexes et Points de vue : l’Ingénierie des Modèles centrée utilisateur pour L’Ingénierie Système"

- Olivier Barais was in the jury (president) of Mulugeta Ayalew Tamiru’s PhD of Universite of Rennes on: "Automatic resource management in geo-distributed multi-cluster environments"

- Olivier Barais was in the jury (reviewer) of Nikolaos Antoniadis’s PhD of Universite of Luxembourg on: "Enhancing Smart Grid Resilience and Reliability by Using and Combining Simulation and Optimization Methods"

- Olivier Barais was in the jury (president) of Elyes Cherfa’s PhD of Universite of South Brittany on: "Assistance à la spécification de contraintes OCL dans les métamodèles"

11.3 Popularization

11.3.1 Articles and contents

Olivier Zendra, as a member of its editorial board, contributed to the **HiPEAC Vision 2021** [57]. Our world is evolving very rapidly, both from the technological point of view — with impressive advances in artificial intelligence and new hardware challenging longstanding PC hardware traditions, for example — and as a result of unexpected events. The year 2020 was quite exceptional, an annus horribilis, according to some. It is hard to disagree with this statement, but every dark cloud has a silver lining. 2020 was also the year that accelerated digital transformation beyond what could have been imagined in 2019. Vaccine development happened faster than would ever have been conceivable a year ago, digital payment became the norm for many people and e-commerce and online sales threatened brick and mortar shops. Employees were encouraged to work from home – with its advantages and disadvantages, videoconferencing became the de facto way to interact with both family and colleagues, schools were forced to experiment with distance learning. The list goes on. After living for over a year in an online world, most people will not return completely to the "old normal". They will go for a combination of the "old normal" and things they discovered and experimented with in the circumstances forced upon us by COVID-19; they might keep their home office on some days, and be in the workplace on other days. Higher education will certainly also continue to offer online teaching. The rapidly evolving digital world has also had an impact on the HiPEAC Vision: updating it every two years no longer seems quite in keeping with the speed of the evolution of computing systems. Therefore, we decided to move from producing a large roadmap document every other year, to an agile, rapidly evolving electronic magazine-like set of articles. The HiPEAC Vision 2021 has two main parts: 1) A set of recommendations for the HiPEAC community. It will also introduce the second part of the Vision and will be updated periodically, or on particular occasions. 2) A set of "articles", such as in magazines, that will be regularly updated, the purpose of which is to support the set of recommendations, or to introduce new topics or situations. This will guarantee that the HiPEAC Vision keeps evolving and remains up to date. These articles are intended to be self-sufficient and can be read independently. They are grouped into four themes or dimensions: technical, business, societal and European. New articles will be added over the course of the years (and outdated ones might be removed). A further element of this new approach to the Vision is that the editorial board asked and will ask various authors to contribute to those articles. "is adds heterogeneity and a diversity of point of view that could be helpful for better analysis of the computing systems landscape as well as improve the quality of the recommendations.

In [55], we advocate that **cybersecurity must come to IT systems now**. After decades of apparently low-intensity cyber attacks, during which security was not really thought of on most IT systems, recent years have brought a flurry of well-organized, larger-scale attacks that have caused billions of Euros of damage. This was made possible by the plethora of IT systems that have been produced with no or low security, a trend that has further increased with the rise of ubiquitous computing, with smartphones, IoT and smart-* being everywhere with extremely low control. However, although the current situation in IT systems can still be considered as critical and very much working in favour of cyber attackers, there are definitely paths to massive but doable technical improvements that can lead us to a much more secure and sovereign IT ecosystem, along with strong business opportunities in Europe.

In [54], we claim that **whether you're aware of it or not, privacy does matter**. While privacy used to be a concern of only a limited number of people, in recent years awareness of it has been growing. This has been for a number of reasons including the enactment of the GDPR, the growing impact of data leaks, data logging by governments and companies, and even the recent discussions about COVID-19 contact tracing. At the same time, most of us are knowingly or unknowingly sending more and more private data to the cloud, which increases the risk of it being leaked or abused in some way. In order to try and reconcile these two opposing directions, consumers and companies alike should increase their usage of privacy-enhancing technologies, and businesses should integrate privacy by design into their development.

In [56], we advocate **taming the IT systems complexity hydra**. Although most people remain unaware of its presence in the background, the ever-increasing complexity of IT, with its multiple sources, has been an ongoing issue for quite some time. It can even be qualified as a crisis, in both hardware and software. Indeed, this complexity has reached the point where systems are no longer fully understandable by human beings, which raises the question of how we can continue being in full control of their functioning. It is of course a matter of cost for the IT industry. But a number of incidents caused by bugs or a misunderstanding of some part of an IT system have already occurred. With an IT world that is permanently connected on a worldwide scale, the risk of damage caused by the lack of control of IT systems is both real and growing, with errors and malevolent attacks the most likely culprits.Taming the IT complexity hydra is thus more necessary than ever. Fortunately, various solutions can be proposed to tackle the various heads of the hydra (i.e. the various aspects of complexity); these are solutions based on existing methodologies, tools and resources or extensions thereof.

12 Scientific production

12.1 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.

12.2 Publications of the year

International journals

[24] J. Alves Pereira, H. Martin, M. Acher, J.-M. Jézéquel, G. Botterweck and A. Ventresque. 'Learning Software Configuration Spaces: A Systematic Literature Review'. In: *Journal of Systems and Software* (6th Aug. 2021). DOI: 10.1016/j.jss.2021.111044. URL: https://hal.inria.fr/hal-02148791.

[25] A. Blouin and J.-M. Jézéquel. 'Interacto: A Modern User Interaction Processing Model'. In: *IEEE Transactions on Software Engineering* (2021), pp. 1–20. DOI: 10.1109/TSE.2021.3083321. URL: https://hal.inria.fr/hal-03231669.

[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* (26th Feb. 2021), pp. 1–23. DOI: 10.1007/s10270-021-00869-y. 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* 38.4 (2021). DOI: 10.1109/MS.2020.2995125. URL: https://hal.inria.fr/hal-02612087.

[28] 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* 9.3 (July 2021), pp. 1103–1116. DOI: 10.1109/TCC.2019.2898192. URL: https://hal.inria.fr/hal-02013421.

[29] R. Eramo, F. Bordeleau, B. Combemale, M. Van Den Brand, M. Wimmer and A. Wortmann. 'Conceptualizing Digital Twins'. In: *IEEE Software* (2021), pp. 1–7. URL: https://hal.inria.fr/hal-03466396.

[30] 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* 175 (2021), p. 19. DOI: 10.1016/j.jss.2020.110889. URL: https://hal.inria.fr/hal-03127118.

[31] R. Kretschmer, D. E. Khelladi, R. E. Lopez-Herrejon and A. Egyed. 'Consistent Change Propagation within Models'. In: *Software and Systems Modeling* 20.2 (Apr. 2021), pp. 539–555. DOI: 10.1007/s10270-020-00823-4. URL: https://hal.inria.fr/hal-03029432.

[32] D. Leroy, J. Sallou, J. Bourcier and B. Combemale. 'When Scientific Software Meets Software Engineering'. In: *Computer* (2021), pp. 1–11. URL: https://hal.inria.fr/hal-03318348.

[33] G. Lyan, D. Gross-Amblard, J.-M. Jézéquel and S. Malinowski. 'Impact of Data Cleansing for Urban Bus Commercial Speed Prediction'. In: *Springer Nature Computer Science* (23rd Nov. 2021), pp. 1–11. URL: https://hal.inria.fr/hal-03220449.

[34] H. Martin, M. Acher, J. A. Pereira, L. Lesoil, J.-M. Jézéquel and D. E. Khelladi. 'Transfer Learning Across Variants and Versions: The Case of Linux Kernel Size'. In: *IEEE Transactions on Software Engineering* (2021), pp. 1–17. URL: https://hal.inria.fr/hal-03358817.

International peer-reviewed conferences

[35] M. Acher. 'Reproducible Science and Deep Software Variability'. In: VaMoS 2022 - 16th International Working Conference on Variability Modelling of Software-Intensive Systems. Florence, Italy, 23rd Feb. 2022. URL: https://hal.inria.fr/hal-03528889.

[36] R. Belafia, P. Jeanjean, O. Barais, G. Le Guernic and B. Combemale. 'From Monolithic to Microservice Architecture: The Case of Extensible and Domain-Specific IDEs'. In: MODELS 2021: ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems. Virtual, Japan: IEEE, 10th Oct. 2021, pp. 1–10. URL: https://hal.inria.fr/hal-03342678.

[37] 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, pp. 1–5. URL: https://hal.inria.fr/hal-03107122.

[38] G. Jouneaux, O. Barais, B. Combemale and G. Mussbacher. 'SEALS: A framework for building Self-Adaptive Virtual Machines'. In: SLE 2021 - 14th ACM SIGPLAN International Conference on Software Language Engineering. Chicago, United States: ACM, 17th Oct. 2021, pp. 1–14. DOI: 10.1145/3486608.3486912. URL: https://hal.inria.fr/hal-03355253.

[39] G. Jouneaux, O. Barais, B. Combemale and G. Mussbacher. 'Towards Self-Adaptable Languages'. In: Onward! 2021 - ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. Chicago, United States: ACM, 17th Oct. 2021, pp. 1–16. URL: https://hal.inria.fr/hal-03318816.

[40] Q. Le Dilavrec, D. E. Khelladi, A. Blouin and J.-M. Jézéquel. 'Untangling Spaghetti of Evolutions in Software Histories to Identify Code and Test Co-evolutions'. In: ICMSE'21 - 37th International Conference on Software Maintenance and Evolution. Virtual Event, Luxembourg, 27th Sept. 2021, pp. 1–11. URL: https://hal.inria.fr/hal-03340174.

[41] D. Leroy, B. Lelandais, M.-P. Oudot and B. Combemale. 'Monilogging for Executable Domain-Specific Languages'. In: SLE 2021 - 13th International Conference on Software Language Engineering. Chicago, United States: ACM, 17th Oct. 2021, pp. 1–14. DOI: 10.1145/3486608.3486906. URL: https://hal.inria.fr/hal-03358061.

[42] 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, 9th Feb. 2021, pp. 1–8. URL: https://hal.inria.fr/hal-03084276.

[43] L. Lesoil, M. Acher, X. Tërnava, A. Blouin and J.-M. Jézéquel. 'The Interplay of Compile-time and Run-time Options for Performance Prediction'. In: SPLC 2021 - 25th ACM International Systems and Software Product Line Conference - Volume A. Leicester, United Kingdom: ACM, 6th Sept. 2021, pp. 1–12. DOI: 10.1145/3461001.3471149. URL: https://hal.archives-ouvertes.fr/hal-03286127.

[44] L. Lesoil, H. Martin, M. Acher, A. Blouin and J.-M. Jézéquel. 'Transferring Performance between Distinct Configurable Systems : A Case Study'. In: 16th International Working Conference on Variability Modelling of Software-Intensive Systems. Florence, Italy, 23rd Feb. 2022. DOI: 10.1145/3510466.3510486. URL: https://hal.inria.fr/hal-03514984.

[45] G. Lyan, D. Gross-Amblard and J.-M. Jézéquel. 'On the Quality of Compositional Prediction for Prospective Analytics on Graphs'. In: *DEXA 2021 - Database and Expert Systems Applications - Workshops*. DaWaK 2021 - 23rd International Conference on Big Data Analytics and Knowledge Discovery. Database and Expert Systems Applications - DEXA 2021 Workshops. Linz, Austria, 20th Sept. 2021, pp. 91–105. DOI: 10.1007/978-3-030-87101-7_10. URL: https://hal.archives-ouvertes.fr/hal-03356199.

[46] G. Lyan, J.-M. Jézéquel, D. Gross-Amblard and B. Combemale. 'DataTime: a Framework to smoothly Integrate Past, Present and Future into Models'. In: MODELS 2021 - ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems. Fukuoka, Japan, 10th Oct. 2021, pp. 1–11. URL: https://hal.inria.fr/hal-03355162.

[47] H. Martin, M. Acher, J. A. Pereira and J.-M. Jézéquel. 'A comparison of performance specialization learning for configurable systems'. In: *SPLC '21: Proceedings of the 25th ACM International Systems and Software Product Line Conference*. SPLC 2021 - 25th ACM International Systems and Software Product Line Conference. Vol. A. Leicester, United Kingdom: ACM, 6th Sept. 2021, pp. 46–57. DOI: 10.1145/3461001.3471155. URL: https://hal.archives-ouvertes.fr/hal-03335263.

[48] J. Mortara, X. Tërnava, P. Collet and A.-M. Dery-Pinna. 'Extending the Identification of Object-Oriented Variability Implementations using Usage Relationships'. In: SPLC 2021 - 25th ACM International Systems and Software Product Line Conference. Vol. Volume B. SPLC 2021 - Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume B. Leicester, United Kingdom: ACM, 6th Sept. 2021, pp. 1–8. DOI: 10.1145/3461002.3473943. URL: https://hal.archives-ouvertes.fr/hal-03284626.

[49] L. Noureddine, A. Heuser, C. Puodzius and O. Zendra. 'SE-PAC: A Self-Evolving PAcker Classifier against rapid packers evolution'. In: CODASPY '21 - 11th ACM Conference on Data and Application Security and Privacy. Virtual Event, United States: ACM, 26th Apr. 2021, pp. 1–12. DOI: 10.1145/3422337.3447848. URL: https://hal.inria.fr/hal-03149211.

[50] A. A. Pranata, O. Barais, J. Bourcier and L. Noirie. 'ChaT: Evaluation of Reconfigurable Distributed Network Systems Using Metamorphic Testing'. In: GLOBCOM 2021 - IEEE Global Communications Conference. Madrid, Spain: IEEE, 7th Dec. 2021, pp. 1–6. URL: https://hal.archives-ouvertes.fr/hal-03379913.

[51] C. Puodzius, O. Zendra, A. Heuser and L. Noureddine. 'Accurate and Robust Malware Analysis through Similarity of External Calls Dependency Graphs (ECDG)'. In: *IWCC 2021 - 10th International Workshop on Cyber Crime (held in conjunction with ARES 2021)*. ARES 2021 - The 16th International Conference on Availability, Reliability and Security. Virtual, Austria: ACM, 17th Aug. 2021, pp. 1–12. DOI: 10.1145/3465481.3470115. URL: https://hal.archives-ouvertes.fr/hal-03328395.

[52] X. Tërnava, L. Lesoil, G. A. Randrianaina, D. E. Khelladi and M. Acher. 'On the Interaction of Feature Toggles'. In: VaMoS 2022 - 16th International Working Conference on Variability Modelling of Software-Intensive Systems. The Proceedings of 16th International Working Conference on Variability Modelling of Software-Intensive Systems. Florence, Italy, 23rd Feb. 2022. DOI: 10.1145/3510466.3510485. URL: https://hal.archives-ouvertes.fr/hal-03527250.

National peer-reviewed Conferences

[53] A. Blouin and J.-M. Jézéquel. 'Programmez vos IHM avec Interacto: une démonstration'. In: GDR GPL 2021 - Journées du Groupement de Recherche du Génie de la Programmation et du Logiciel. Virtuelle, France, 14th June 2021, pp. 1–2. URL: https://hal.inria.fr/hal-03259896.

Scientific book chapters

[54] B. Coppens and O. Zendra. 'Privacy: whether you're aware of it or not, it does matter!' In: *HiPEAC Vision 2021*. Jan. 2021, pp. 1–6. DOI: 10.5281/zenodo.4719402. URL: https://hal.inria.fr/hal-03362809.

[55] O. Zendra and B. Coppens. 'Cybersecurity must come to IT systems now'. In: *HiPEAC Vision 2021*. Jan. 2021, pp. 1–6. URL: https://hal.inria.fr/hal-03362808.

[56] O. Zendra and K. De Bosschere. 'Taming the IT systems complexity hydra'. In: *HiPEAC Vision 2021*. Jan. 2021, pp. 1–8. DOI: 10.5281/zenodo.4719574. URL: https://hal.inria.fr/hal-03362810.

Edition (books, proceedings, special issue of a journal)

[57] M. Duranton, K. De Bosschere, B. Coppens, C. Gamrat, t. hoberg thomas, H. Munk, C. Roderick, T. Vardanega and O. Zendra. *HiPEAC Vision 2021: High Performance Embedded Architecture And Compilation: Vision 2021*. Jan. 2021, pp. 1–228. URL: https://hal.inria.fr/hal-03359519.

Doctoral dissertations and habilitation theses

[58] M. Acher. 'Modelling, Reverse Engineering, and Learning Software Variability'. Université de Rennes 1, 16th Nov. 2021. URL: https://hal.inria.fr/tel-03521806.

[59] G. Lyan. 'Urban mobility : leveraging machine learning and data masses for the building of simulators'. Université Rennes 1, 23rd Sept. 2021. URL: https://tel.archives-ouvertes.fr/tel-03520672.

[60] L. Noureddine. 'Packing detection and classification relying on machine learning to stop malware propagation'. UNIVERSITE DE RENNES 1; INRIA Rennes - Bretagne Atlantique, 21st Dec. 2021. URL: https://hal.inria.fr/tel-03522278.

12.3 Cited publications

[61] 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.

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

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

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

[71] 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.

[72] 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.

[73] 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.

[74] 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).

[75] 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.

[76] 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.

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

[78] 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.

[79] 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.

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

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

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

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

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

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

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] 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.

[98] 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.

[99] 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.

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

[101] 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.

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

[103] 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.

[104] 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.

[105] R. Potvin and J. Levenberg. 'Why Google stores billions of lines of code in a single repository'. In: *Communications of the ACM* 59.7 (2016), pp. 78–87.

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

[107] 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.

[108] 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.

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

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

[111] 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.

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

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

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

[115] 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.

[116] 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.

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