RESEARCH CENTRE

Inria Center at Rennes University

IN PARTNERSHIP WITH:

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

2022

ACTIVITY REPORT

Project-Team

DIVERSE

Diversity-centric Software Engineering

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

DOMAIN

Networks, Systems and Services, Distributed Computing

THEME

Distributed programming and Software engineering

Contents

Project-Team DIVERSE

Creation of the Project-Team: 2014 July 01

Keywords

Computer sciences and digital sciences

A1.2.1. – Dynamic reconfiguration

A1.3.1. – Web

A1.3.5. – Cloud

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

A2.6.4. – Ressource management

A4.1.1. – Malware analysis

A4.4. – Security of equipment and software

A4.6. – Authentication

A4.7. – Access control

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 Khelladi [CNRS, Researcher]
- Gunter Mussbacher [UNIV MCGILL, Advanced Research Position, until Aug 2022]
- Olivier Zendra [INRIA, Researcher]

Faculty Members

- Olivier Barais [Team leader, UNIV RENNES I, Professor, HDR]
- Mathieu Acher [INSA Rennes, Professor, from Sep 2022, IUF, HDR]
- Arnaud Blouin [INSA RENNES, Associate Professor, HDR]
- Johann Bourcier [UNIV RENNES I, Associate Professor, HDR]
- Stéphanie Challita [UNIV RENNES I, Associate Professor]
- Benoît Combemale [UNIV RENNES I, Professor, HDR]
- Jean-Marc Jézéquel [UNIV RENNES I, Professor, HDR]
- Noël Plouzeau [UNIV RENNES I, Associate Professor]
- Walter Rudametkin Ivey [UNIV RENNES I, Associate Professor, from Sep 2022, HDR]
- Paul Temple [UNIV RENNES I, Associate Professor, from Sep 2022]

Post-Doctoral Fellow

- Xhevahire Ternava [UNIV RENNES I]

PhD Students

- Anne Bumiller [ORANGE]
- Cassius De Oliveira Puodzius [INRIA, until Jan 2022]
- Theo Giraudet [UNIV RENNES I, CIFRE, from Sep 2022]
- Gwendal Jouneaux [UNIV RENNES I]
- Zohra Kebaili [CNRS]
- Piergiorgio Ladisa [SAP]
- Leo Laugier [UNIV RENNES I, from Oct 2022]
- Quentin Le Dilavrec [UNIV RENNES I]
- Luc Lesoil [UNIV RENNES I]
- Georges Aaron Randrianaina [UNIV RENNES I]

Technical Staff

- Florian Badie [INRIA, Engineer]
- Romain Belafia [UNIV RENNES I, Engineer]
- Emmanuel Chebbi [INRIA, Engineer]
- Guy De Spiegeleer [UNIV RENNES I, Engineer, from Feb 2022]
- Theo Giraudet [UNIV RENNES I, Engineer, until Feb 2022]
- Pierre Jeanjean [INRIA, Engineer]
- Romain Lefeuvre [INRIA, Engineer]
- Dorian Leroy [INRIA, Engineer]
- Didier Vojtisek [INRIA, Engineer]

Interns and Apprentices

- Benjamin Ramone [UNIV RENNES I, from May 2022]

Administrative Assistant

- Sophie Maupile [CNRS]

Visiting Scientists

- Jessie Galasso-Carbonnel [UNIV MONTREAL, from Nov 2022]
- Mark Van Den Brand [UNIV EINDHOVEN, from Apr 2022 until Apr 2022]

External Collaborator

- Gurvan Le Guernic [DGA, until Nov 2022]

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 correctness 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 [111]. 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) [115]. 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 [86]. 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 [70]. 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 [117].

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. [91] 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 [101].

3.2.2 Variability modeling

While the basic vision underlying *Software Product Lines* (SPL) can probably be traced back to David Parnas' seminal article [108] 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 [105]. 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* [106]. Bosch provides a different definition [76]: *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 [82]. 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 [88]. 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 [77]. Variability management is thus increasingly seen as the cornerstone of SPL development, covering the entire development life cycle, from requirements elicitation [119] to product derivation [123] to product testing [104, 103].

Halmans *et al.* [88] 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.* [110]: 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.* [98] (feature, hardware platform, performance and attributes variability) or Bass *et al.* [68] 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* [93]. 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 [77]. 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 [72]. Model Driven Engineering (MDE) makes it possible to automate this weaving process [92]. 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 [102]) manipulating these models to produce for instance a detailed design that can ultimately be transformed to code, or to test suites [109], or other software artifacts.

3.2.3 Component-based software development

Component-based software development [118] 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 [83]: description of requirements on services required from the other components; indirect connections between components thanks to ports and connectors constructs [96]; hierarchical definition of components (assemblies of components can define new component types); connectors supporting various communication semantics [80]; quantitative properties on the services [75].

In recent years component-based architectures have evolved from static designs to dynamic, adaptive designs (e.g., SOFA [80], Palladio [73], Frascati [99]). 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 [94]. 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 [81, 121]. 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 [74], 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 [113]". 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 [87]. Multi Objectives Search based techniques [84] 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 [120] mainly relies on model analysis, constraint solving [85] and search-based reasoning [97]. DIVERSE leverages in particular the applications of model-based testing in the context of highly-configurable systems and [122] interactive systems [100] as well as recent advances based on diversity for test cases selection [90].

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 [107], 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 [79] 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 [69] provides modeling elements amenable to semantically equivalent mathematical models. Metropolis offers a precise semantics flexible enough to support different models of computation. ModHel'X [89] 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 [67], recovery blocks [112] and code randomization [71], 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 [116, 114]. Such methods have been used for example to understand the impact of new software development paradigms [78]. 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 [66].

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...).

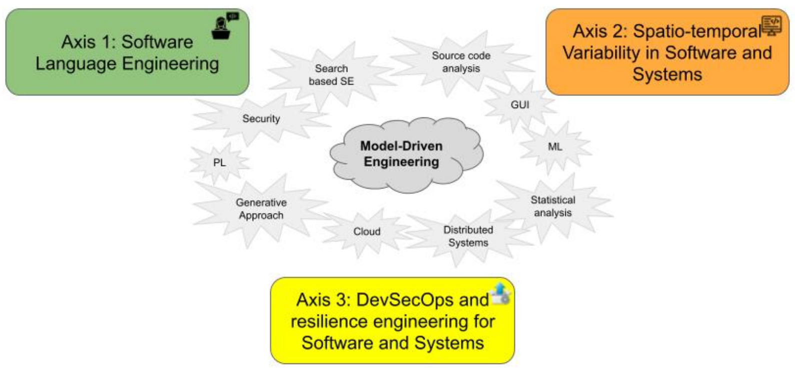
Axis 1: Software Language Engineering
Axis 2: Spatio-temporal Variability in Software and Systems
Search based SE
Source code analysis
GUI
Security
Model-Driven Engineering
ML
PL
Generative Approach
Statistical analysis
Cloud
Distributed Systems
Axis 3: DevSecOps and resilience engineering for Software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

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.

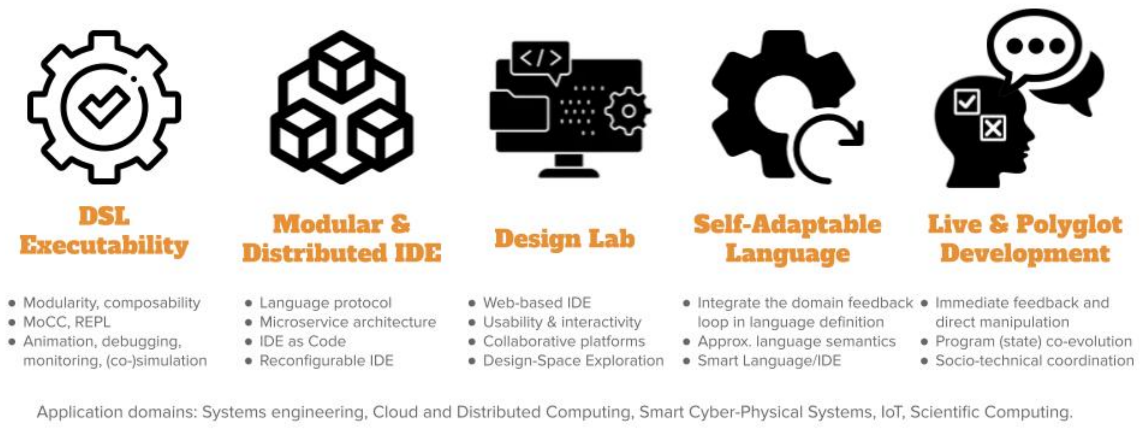
DSL Executability
Modular & Distributed IDE
Design Lab
Self-Adaptable Language
Live & Polyglot Development
• Modularity, composability
• MoCC, REPL
• Animation, debugging, monitoring, (co-)simulation
• Language protocol
• Microservice architecture
• IDE as Code
• Reconfigurable IDE
• Web-based IDE
• Usability & interactivity
• Collaborative platforms
• Design-Space Exploration
• Integrate the domain feedback loop in language definition
• Approx. language semantics
• Smart Language/IDE
• 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)

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 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 human values or security, and provide facilities for trade-off analysis and decision making in the the context of software and system designs.

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 are more used 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. 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

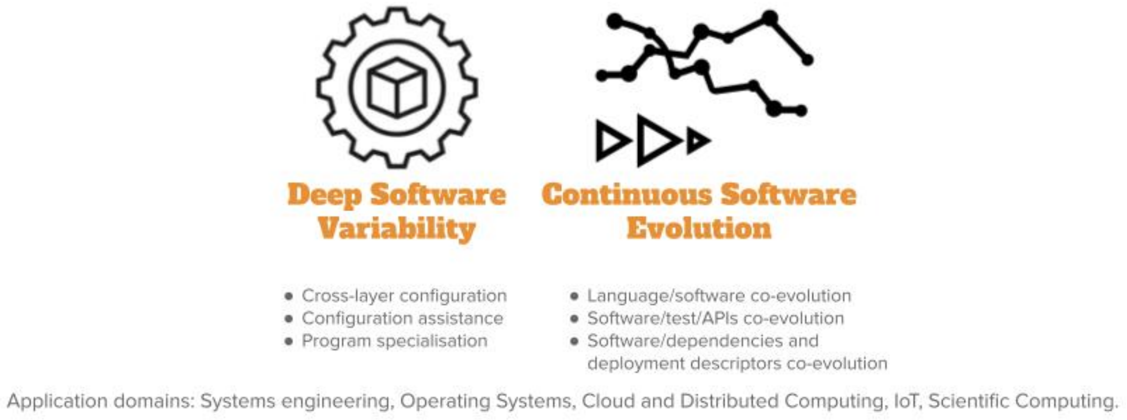
Deep Software Variability
Continuous Software Evolution
• Cross-layer configuration
• Configuration assistance
• Program specialisation
• Language/software co-evolution
• Software/test/APIs co-evolution
• Software/dependencies and deployment descriptors co-evolution
Application domains: Systems engineering,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

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

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. 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

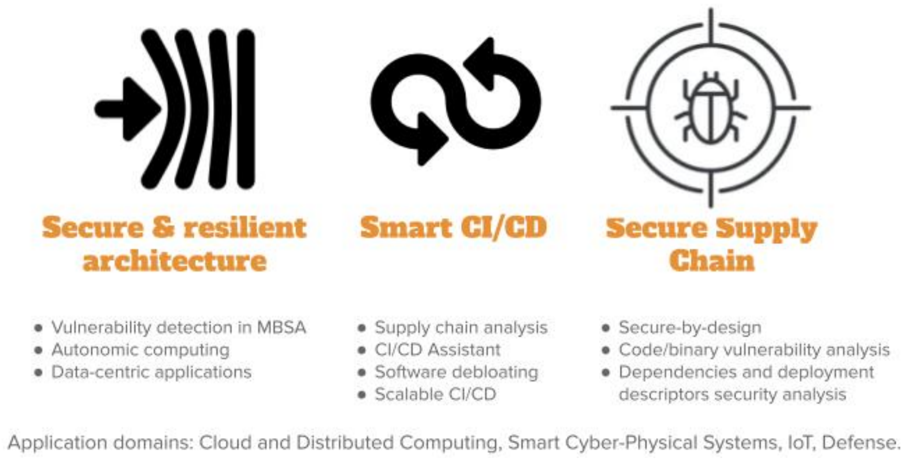
Secure & resilient architecture
• Vulnerability detection in MBSA
• Autonomic computing
• Data-centric applications
Smart CI/CD
• Supply chain analysis
• CI/CD Assistant
• Software debloating
• Scalable CI/CD
Secure Supply Chain
• Secure-by-design
• Code/binary vulnerability analysis
• Dependencies and 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)

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. 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 Linux 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 Highlights of the year

5.1 Impact

- The work on Risk explorer for exploring open-source software supply chain security has been accepted to SnP confenrence in 2023 [95]. This work and the associated tooling got some impact in the field in 2022 [44]:
  - Cited as guidance resource in Microsoft's OSS SSC Framework.
  - Cited in Adam Shostack's Application Security Roundup of September '22.
  - Risk Explorer used internally at SAP and at Citigroup Inc. for threat modelling and development of best-practices. Submission to RSA Conference in forecast.
  - Ongoing discussion to transfer the taxonomy under OpenSSF.

5.2 Awards

- Our paper "HyperAST: Enabling Efficient Analysis of Software Histories at Scale. Quentin Le Dilavrec, Djamel Eddine Khelladi, Arnaud Blouin, Jean-Marc Jézéquel. ASE 2022 - 37th IEEE/ACM International Conference on Automated Software Engineering", received an ACM sigsoft distinguished paper award at ASE 2022.
- Our paper "Scratching the Surface of ./configure: Learning the Effects of Compile-Time Options on Binary Size and Gadgets. Xhevahire Tërnava, Mathieu Acher, Luc Lesoil, Arnaud Blouin, Jean-Marc Jézéquel. ICSR 2022 - 20th International Conference on Software and Systems Reuse", received the best paper award at ICSR 2022.

5.3 New permanent positions within the team

- Mathieu Acher has been promoted as Professor at INSA Rennes.
- Walter Rudametkin has joined the team as Full Professor at University of Rennes 1. He also was awarded an IUF junior position at the same time.
- Paul Temple has joined the team as Associate Professor at University of Rennes 1.

6 New software and platforms

6.1 New software

6.1.1 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

6.1.2 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

Publications: hal-00850770, hal-01355391, hal-01609576, hal-01651801, hal-01152342, hal-03374955, hal-01614561, hal-01616154

Contact: Benoît Combemale

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

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

6.1.3 Interacto

Keywords: GUI (Graphical User Interface), User Interfaces, HCI, Software engineering

Functional Description: Interacto is a framework for developing user interfaces and user interactions. It complements other general graphical framework by providing a fluent API specifically designed to process user interface event and develop complex user interactions. Interacto is currently developped in Java and TypeScript to target both Java desktop applications (JavaFX) and Web applications (Angular).

URL: `https://interacto.github.io`

Publications: hal-03231669, tel-02354530, inria-00590891, inria-00477627

Contact: Arnaud Blouin

Participants: Arnaud Blouin, Olivier Beaudoux

6.1.4 ALE

Name: Action Language for Ecore

Keywords: Meta-modeling, Executable DSML

Functional Description: Main features of ALE include:

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

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

Contact: Benoît Combemale

Partner: OBEO

6.1.5 Melange

Name: Melange

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

7.1 Results for Axis #1: Software Language Engineering

Participants: Olivier Barais, Johann Bourcier, Benoît Combemale, Jean-Marc Jézéquel, Gurvan Leguernic, Gunter Mussbacher, Noël Plouzeau, Didier Vojtisek.

7.1.1 Foundations of Software Language Engineering

Exploratory programming is a software development style in which code is a medium for prototyping ideas and solutions, and in which even the end-goal can evolve over time. Exploratory programming is valuable in various contexts, such as programming education, data science, and end-user programming. However, there is a lack of appropriate tooling and language design principles to support exploratory programming. In [37], we present a host language- and object language-independent protocol for exploratory programming akin to the Language Server Protocol. The protocol serves as a basis to develop novel programming environments (or to extend existing ones) for exploratory programming, such as computational notebooks and command-line REPLs. An architecture is exposed, on top of which prototype environments can be developed with relative ease, because existing (language) components can be reused. Our prototypes demonstrate that the proposed protocol is sufficiently expressive to support exploratory programming scenarios as encountered in literature of the software engineering, human-computer interaction and data science domains.

Recent results in language engineering simplify the development of tool-supported executable domain-specific modelling languages (xDSMLs), including editing (e.g., completion and error checking) and execution analysis tools (e.g., debugging, monitoring and live modelling). However, such frameworks are currently limited to sequential execution traces, and cannot handle execution traces resulting from an execution semantics with a concurrency model supporting parallelism or interleaving. This prevents the development of concurrency analysis tools, like debuggers supporting the exploration of model executions resulting from different interleavings. In [34], we present a generic framework to integrate execution semantics with either implicit or explicit concurrency models, to explore the possible execution traces of conforming models, and to define strategies to help in the exploration of the possible executions. This framework is complemented with a protocol to interact with the resulting executions and hence to build advanced concurrency analysis tools. The approach has been implemented within the GEMOC Studio. We demonstrate how to integrate two representative concurrent meta-programming approaches (MoCCML/Java and Henshin), which use different paradigms and underlying foundations to define an xDSML's concurrency model. We also demonstrate the ability to define an advanced concurrent omniscient debugger with the proposed protocol. Our work, thus, contributes key abstractions and an associated protocol for integrating concurrent meta-programming approaches in a language workbench, and dynamically exploring the possible executions of a model in the modelling workbench.

7.1.2 DSL for Scientific Computing

Scientific software are complex software systems. Their engineering involves various stakeholders using specific computer languages for defining artifacts at different abstraction levels and for different purposes. In [28], we review the overall process leading to the development of scientific software, and discuss the role of computer languages in the definition of the different artifacts. We then provide guidelines to make informed decisions when the time comes to choose the computer languages to use when developing scientific software.

7.1.3 Digital Twins

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

In [26], 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 twin's entire life cycle.

We also created EDT.Community, a programme of seminars on the engineering of digital twins hosting digital twins experts from academia and industry. In [41], we report on the main topics of discussion from the first year of the programme. We contribute by providing (1) a common understanding of open challenges in research and practice of the engineering of digital twins, and (2) an entry point to researchers who aim at closing gaps in the current state of the art.

7.1.4 Reasoning over Time 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 [30], we propose a framework called DataTime to implement models at runtime that 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 offers 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 be used for managing data from the past, the present and the future, and facilitate the development of digital twins.

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

Participants: Mathieu Acher, Arnaud Blouin, Benoît Combemale, Jean-Marc Jézéquel, Djamel Eddine Khelladi, Olivier Zendra
.

7.2.1 Learning at scale

Learning large-scale variability In [36], 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 on configurable systems and machine learning, as motivating example stating 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 [31].

Linux kernels are used in a wide variety of appliances, many of them having strong requirements on the kernel size due to constraints such as limited memory or instant boot. With more than nine thousands of configuration options to choose from, developers and users of Linux actually spend significant effort to document, understand, and eventually tune (combinations of) options for meeting a kernel size. In [36], we describe a large-scale endeavour automating this task and predicting a given Linux kernel binary size out of unmeasured configurations. We first experiment that state-of-the-art solutions specifically made for configurable systems such as performance-influence models cannot cope with that number of options, suggesting that software product line techniques may need to be adapted to such huge configuration spaces. We then show that tree-based feature selection can learn a model achieving low prediction errors over a reduced set of options. The resulting model, trained on 95,854 kernel configurations, is quick to compute, simple to interpret and even outperforms the accuracy of learning without feature selection.

7.2.2 Smart build

Incremental build of configurations and variants Building software is a crucial task to compile, test, and deploy software systems while continuously ensuring quality. As software is more and more configurable, building multiple configurations is a pressing need, yet, costly and challenging to instrument. The common practice is to independently build (a.k.a., clean build) a software for a subset of configurations. While incremental build has been considered for software evolution and relatively small modifications of the source code, it has surprisingly not been considered for software configurations. In this work, we formulate the hypothesis that incremental build can reduce the cost of exploring the configuration space of software systems. In [49], we detail how we apply **incremental build** for two real-world application scenarios and conduct a preliminary evaluation on two case studies, namely x264 and the Linux Kernel. For x264, we found that one can incrementally build configurations in an order such that overall build time is reduced. Nevertheless, we could not find any optimal order with the Linux Kernel, due to a high distance between random configurations. Therefore, we show it is possible to control the process of generating configurations: we could reuse commonality and gain up to 66% of build time compared to only clean builds.

In the exploratory study [50], we examine the benefits and limits of building software configurations incrementally, rather than always building them cleanly. By using five real-life configurable systems as subjects, we explore whether incremental build works, outperforms a sequence of clean builds, is correct w.r.t. clean build, and can be used to find an optimal ordering for building configurations. Our results show that incremental build is feasible in 100% of the times in four subjects and in 78% of the times in one subject. In average, 88.5% of the configurations could be built faster with incremental build while also finding several alternatives faster incremental builds. However, only 60% of faster incremental builds are correct. Still, when considering those correct incremental builds with clean builds, we could always find an optimal order that is faster than just a collection of clean builds with a gain up to 11.76%.

7.2.3 Variability and debloating

Debloating variability In [54], we call for **removing variability**. Indeed, software variability is largely accepted and explored in software engineering and seems to have become a norm and a must, if only in the context of product lines. Yet, the removal of superfluous or unneeded software artefacts and functionalities is an inevitable trend. It is frequently investigated in relation to software bloat. This work is essentially a call to the community on software variability to devise methods and tools that will facilitate the removal of unneeded variability from software systems. The advantages are expected to be numerous in terms of functional and non-functional properties, such as maintainability (lower complexity), security (smaller attack surface), reliability, and performance (smaller binaries).

Feature toggling and variability Feature toggling is a technique for enabling branching-in-code. It is increasingly used during continuous deployment to incrementally test and integrate new features before their release. In principle, feature toggles tend to be light, that is, they are defined as simple Boolean flags and used in conditional statements to condition the activation of some software features. However, there is a lack of knowledge on whether and how they may interact with each other, in that case their enabling and testing become complex. We argue that finding the interactions of feature toggles is valuable for developers to know which of them should be enabled at the same time, which are impacted by a removed toggle, and to avoid their misconfigurations. In [51], we mine feature toggles and their interactions in five open-source projects. We then analyse how they are realized and whether they tend to be multiplied over time. Our results show that 7% of feature toggles interact with each other, 33% of them interact with another code expression, and their interactions tend to increase over time (22%, on average). Further, their interactions are expressed by simple logical operators (i.e., and and or) and nested if statements. We propose to model them into a Feature Toggle Model, and believe that our results are helpful towards robust management approaches of feature toggles.

Several works have already identified the proximity of feature toggles with the notion of Feature found in Software Product Lines. In [42], we propose to go one step further in unifying these concepts to provide a seamless transition between design time and runtime variability resolutions. We show how it can scale to build a configurable authentication system, where a partially resolved feature model can interface with popular feature toggle frameworks such as Togglz.

Gadgets and variability Numerous software systems are configurable through compile-time options and the widely used ./configure. However, the combined effects of these options on binaries' non-functional properties size and attack surface are often not documented, and or not well understood, even by experts. Our goal is to provide automated support for exploring and comprehending the configuration space a. k. a., surface of compile-time options using statistical learning techniques. In [65], we perform an empirical study on four C-based configurable systems. Our results show that, by changing the default configuration, the system's binary size and gadgets vary greatly (roughly -79% to 244% and -77% to 30%, respectively). Then, we found out that identifying the most influential options can be accurately learned with a small training set, while their relative importance varies across size and attack surface for the same system. Practitioners can use our approach and artifacts to explore the effects of compile-time options in order to take informed decisions when configuring a system with ./configure. Our work received the Best paper award at ICSR 2022.

7.2.4 Scaling temporal analysis

Temporal code analysis at scale Syntax Trees (ASTs) are widely used beyond compilers in many tools that measure and improve code quality, such as code analysis, bug detection, mining code metrics, refactoring. With the advent of fast software evolution and multistage releases, the temporal analysis of an AST history is becoming useful to understand and maintain code. However, jointly analyzing thousands of versions of ASTs independently faces scalability issues, mostly combinatorial, both in terms of memory and CPU usage. In [46], we propose a novel type of AST, called HyperAST , that enables efficient temporal code analysis on a given software history by: 1) leveraging code redundancy through space (between code elements) and time (between versions); 2) reusing intermediate computation results. We show how the HyperAST can be built incrementally on a set of commits to capture all multiple ASTs at once in an optimized way. We evaluated the HyperAST on a curated list of large software projects. Compared to Spoon, a state-of-the-art technique, we observed that the HyperAST outperforms it with an order-of-magnitude difference from ×6 up to ×8076 in CPU construction time and from ×12 up to ×1159 in memory footprint. While the HyperAST requires up to 2 h 22 min and 7.2 GB for the largest project, Spoon requires up to 93 h 31 min and 2.2 TB. The gains in construction time varied from 83.4% to 99%.99% and the gains in memory footprint varied from 91.8% to 99.9%. We further compared the task of finding references of declarations with the HyperAST and Spoon. We observed on average 90% precision and 97% recall without a significant difference in search time.

7.2.5 Deep variability

Deep software variability refers to the interaction of all external layers modifying the behavior of software. Configuring software is a powerful means to reach functional and performance goals of a system, but many layers of variability can make this difficult.

Variability in input, version, and software. With commits and releases, hundreds of tests are run on varying conditions (e.g., over different hardware and workloads) that can help to understand evolution and ensure non-regression of software performance. In [47], we hypothesize that performance is not only sensitive to evolution of software, but also to different variability layers of its execution environment, spanning the hardware, the operating system, the build, or the workload processed by the software. Leveraging the MongoDB dataset, our results show that changes in hardware and workload can drastically impact performance evolution and thus should be taken into account when reasoning about evolution. An open problem resulting from this study is how to manage the variability layers in order to efficiently test the performance evolution of a software.

Transferring Performance between Distinct Configurable Systems. Many research studies predict the performance of configurable software using machine learning techniques, thus requiring large amounts of data. Transfer learning aims at reducing the amount of data needed to train these models and has been successfully applied on different executing environments (hardware) or software versions. In [48], we investigate for the first time the idea of applying transfer learning between distinct configurable systems. We design a study involving two video encoders (namely x264 and x265) coming from different code bases. Our results are encouraging since transfer learning outperforms traditional learning for two performance properties (out of three). We discuss the open challenges to overcome for a more general application.

*Global Decision Making Over **Deep Variability** in Feedback-Driven Software Development* To succeed with the development of modern software, organizations must have the agility to adapt faster to constantly evolving environments to deliver more reliable and optimized solutions that can be adapted to the needs and environments of their stakeholders including users, customers, business, development, and IT. However, stakeholders do not have sufficient automated support for global decision making, considering the increasing variability of the solution space, the frequent lack of explicit representation of its associated variability and decision points, and the uncertainty of the impact of decisions on stakeholders and the solution space. This leads to an ad-hoc decision making process that is slow, error-prone, and often favors local knowledge over global, organization-wide objectives. The Multi-Plane Models and Data (MP-MODA) framework introduced in [43] explicitly represents and manages variability, impacts, and decision points. It enables automation and tool support in aid of a multi-criteria decision making process involving different stakeholders within a feedback-driven software development process where feedback cycles aim to reduce uncertainty. We present the conceptual structure of the framework, discuss its potential benefits, and enumerate key challenges related to tool supported automation and analysis within MP-MODA.

Reproducibility We sketch a vision about **reproducible science** and deep software variability in [35].

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

Participants: Mathieu Acher, Olivier Barais, Arnaud Blouin, Stephanie Challita, Benoît Combemale, Jean-Marc Jézéquel, Olivier Zendra.

In this section, we present our achievements for 2022 that draw on our previous works, and that constitute basic blocks upon which we will continue building our research and systems, for example with the aim to extend the applicability to secure supply chains.

7.3.1 Side-channels and source-code vulnerabilities

We also worked on methods and techniques to improve the cybersecurity of code by removing cyber-vulnerabilities from source-codes, especially the ones enabling side-channels attacks.

In [38], we indeed try to address the specific type of cyber attacks known as side channel attacks, where attackers exploit information leakage from the physical execution of a program, e.g. timing or power leakage, to uncover secret information, such as encryption keys or other sensitive data. There have been various attempts at addressing the problem of preventing side-channel attacks, often relying on various measures to decrease the discernibility of several code variants or code paths. Most techniques require a high-degree of expertise by the developer, who often employs ad hoc, hand-crafted code-patching in an attempt to make it more secure. In this work, we take a different approach, building on the idea of ladderisation, inspired by Montgomery Ladders. We present **a semi-automatic tool-supported technique to provide countermeasures to side-channel attacks**. Our technique, aimed at the non-specialised developer, which refactors (a class of) C programs into functionally (and even algorithmically) equivalent counterparts with improved security properties. Our approach provides refactorings that transform the source code into its ladderised equivalent, driven by an underlying verified rewrite system, based on dependent types. Our rewrite system automatically finds rewritings of selected C expressions, facilitating the production of their equivalent ladderised counterparts for a subset of C. We demonstrated our approach on a number of representative examples from the cryptographic domain, showing increased security.

Side-channel attacks are by definition made possible by information leaking from computing systems through nonfunctional properties like execution time, consumed energy, power profiles, etc. These attacks are especially difficult to protect from, since they rely on physical measurements not usually envisioned when designing the functional properties of a program. Furthermore, countermeasures are usually dedicated to protect a particular program against a particular attack, lacking universality. To help fight these threats, we propose in [62] **the Indiscernibility Methodology**, a novel methodology to quantify with no prior knowledge the information leaked from programs, thus providing the developer with valuable security metrics, derived either from topology or from information theory. Our original approach considers the code to be analyzed as a completely black box, only the public inputs and leakages being observed. It can be applied to various types of side-channel leakages: time, energy, power, EM, etc. In this work, we first present our Indiscernibility Methodology, including channels of information and our threat model. We then detail the computation of our novel metrics, with strong formal foundations based both on topological security (with distances defined between secret-dependent observations) and on information theory (quantifying the remaining secret information after observation by the attacker). Then we demonstrate the applicability of our approach by providing experimental results for both time and power leakages, studying both average case, worst case, and indiscernible information metrics.

7.3.2 Malware analysis and classification

Historically, malware (MW) analysis has heavily resorted to human savvy for manual signature creation to detect and classify malware. This procedure is very costly and time consuming, thus unable to cope with modern cyber threat scenario. The solution is to widely automate malware analysis. Toward this goal, malware classification allows optimizing the handling of large malware corpora by identifying resemblances across similar instances. Consequently, malware classification figures as a key activity related to malware analysis, which is paramount in the operation of computer security as a whole. In this line of research work, the PhD thesis [60] addresses the problem of malware classification taking an approach in which human intervention is spared as much as possible. There, we steer clear of subjectivity inherent to human analysis by designing malware classification solely on data directly extracted from malware analysis, thus taking a data-driven approach. Our objective was to improve the automation of malware analysis and to combine it with machine learning methods that are able to autonomously spot and reveal unwitting commonalities within data. This worked was phased in three stages. Initially we focused on improving malware analysis and its automation, studying new ways of leveraging symbolic execution in malware analysis and developing a distributed framework to scale up our computational power. Then we focused on the representation of malware behavior, with painstaking attention to its accuracy and robustness. Finally, we fixed attention on malware clustering, devising a methodology that has no restriction in the combination of syntactical and behavioral features and remains scalable in practice. The main contributions of this work are: revamping the use of symbolic execution for malware analysis with special attention to the optimal use of SMT solver tactics and hyperparameter settings; conceiving a new evaluation paradigm for malware analysis systems; formulating a compact graph representation of behavior, along with a corresponding function for pairwise similarity computation, which is accurate and robust; and elaborating a new malware clustering strategy based on ensemble clustering that is flexible with respect to the combination of syntactical and behavioral features.

7.3.3 Open-source software supply chain security

Open-source software supply chain attacks aim at infecting downstream users by poisoning open-source packages. The common way of consuming such artifacts is through package repositories and the development of vetting strategies to detect such attacks is ongoing research. Despite its popularity, the Java ecosystem is the less explored one in the context of supply chain attacks. In this work [45], we study simple-yet-effective indicators of malicious behavior that can be observed statically through the analysis of Java bytecode. Then we evaluate how such indicators and their combinations perform when detecting malicious code injections. We do so by injecting three malicious payloads taken from real-world examples into the Top-10 most popular Java libraries from libraries.io. We found that the analysis of strings in the constant pool and of sensitive APIs in the bytecode instructions aids in the task of detecting malicious Java packages by significantly reducing the information, thus, making also manual triage possible.

In this context of Supply chain attacks on open-source projects, recent work systematized the knowledge about such attacks and proposed a taxonomy in the form of an attack tree [95]. We propose a visualization tool called Risk Explorer [44] for Software Supply Chains, which allows inspecting the taxonomy of attack vectors, their descriptions, references to real-world incidents and other literature, as well as information about associated safeguards. Being open-source itself, the community can easily reference new attacks, accommodate for entirely new attack vectors or reflect the development of new safeguards. This tool is also available online [1]

7.3.4 A Context-Driven Modelling Framework for Dynamic Authentication Decisions

Nowadays, many mechanisms exist to perform authentication, such as text passwords and biometrics. However, reasoning about their relevance (e.g., the appropriateness for security and usability) regarding the contextual situation is challenging for authentication system designers. In [40], we present a Context-driven Modelling Framework for dynamic Authentication decisions (COFRA), where the context information specifies the relevance of authentication mechanisms. COFRA is based on a precise metamodel that reveals framework abstractions and a set of constraints that specify their meaning. Therefore, it provides a language to determine the relevant authentication mechanisms (characterized by properties that ensure their appropriateness) in a given context. The framework supports the adaptive authentication system designers in the complex trade-off analysis between context information, risks and authentication mechanisms, according to usability, deployability, security, and privacy. We validate the proposed framework through case studies and extensive exchanges with authentication and modelling experts. We show that model instances describing real-world use cases and authentication approaches proposed in the literature can be instantiated validly according to our metamodel. This validation highlights the necessity, sufficiency, and soundness of our framework.

In many situations, it is of interest for authentication systems to adapt to context (e.g., when the user's behavior differs from the previous behavior). Hence, during authentication events, it is common to use contextually available features to calculate an impersonation risk score. This work proposes an explainability model [39] that can be used for authentication decisions and, in particular, to explain the impersonation risks that arise during suspicious authentication events (e.g., at unusual times or locations). The model applies Shapley values to understand the context behind the risks. Through a case study on 30,000 real world authentication events, we show that risky and non risky authentication events can be grouped according to similar contextual features, which can explain the risk of impersonation differently and specifically for each authentication event. Hence, explainability models can effectively improve our understanding of impersonation risks. The risky authentication events can be classified according to attack types. The contextual explanations of the impersonation risk can help authentication policymakers and regulators who attempt to provide the right authentication mechanisms, to understand the suspiciousness of an authentication event and the attack type, and hence to choose the suitable authentication mechanism.

[1] Risk explorer web site

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

ADR Nokia

Participants: Olivier Barais, Johann Bourcier.

- 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

Participants: Mathieu Acher.

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

BCOM

Participants: Olivier Barais.

- 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.

Debug4Science

Participants: Benoît Combemale.

- 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

Participants: Olivier Barais, Benoît Combemale, Stéphanie Chalita.

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

Obeo

Participants: Benoît Combemale, Arnaud Blouin.

- Partners: UR1/Obéo
- Dates: 2022-2025
- Abstract: Low-code language workbench, Theo Giraudet's PhD Cifre project.

SAP

Participants: Olivier Barais.

- Partners: UR1/SAP
- Dates: 2021-2024
- Abstract: Research focusing on Open-source software Supply Chain security. Piergiorgio Ladisa's PhD Cifre project.

9 Partnerships and cooperations

9.1 International initiatives

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

ALE

Participants: Benoît Combemale, Didier Vojtisek, Olivier Barais, Djamel Eddine Khelladi, Gunter Mussbacher.

Title: Agile Language Engineering

Duration: 2020 ->

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

Partners:

- CWI Amsterdam (Pays-Bas)

Inria contact: Benoît Combemale

Summary: Software engineering faces new challenges with the advent of modern software-intensive systems such as complex critical embedded systems, cyber-physical systems and the Internet of things. Application domains range from robotics, transportation systems, defense to home automation, smart cities, and energy management, among others. Software is more and more pervasive, integrated into large and distributed systems, and dynamically adaptable in response to a complex and open environment. As a major consequence, the engineering of such systems involves multiple stakeholders, each with some form of domain-specific knowledge, and with the increased use of software as an integration layer. Hence more and more organizations are adopting Domain-Specific Languages (DSLs) to allow domain experts to express solutions directly in terms of relevant domain concepts. This new trend raises new challenges about designing DSLs, evolving a set of DSLs and coordinating the use of multiple DSLs for both DSL designers and DSL users. ALE will contribute to the field of Software Language Engineering, aiming to provide more agility to both language designers and language users. The main objective is twofold. First, we aim to help language designers to leverage previous DSL implementation efforts by reusing and combining existing language modules, while automating the deployment of distributed, elastic and collaborative modeling environments. Second, we aim to provide more flexibility to language users by ensuring interoperability between different DSLs, offering live feedback about how the model or program behaves while it is being edited (aka. live programming/modeling), and combining with interactive environments like Jupiter Notebook for literate programming.

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

RESIST_EA

Participants: Mathieu Acher, Benoît Combemale, Djamel Eddine Khelladi, Didier Vojtisek.

Title: Resilient Software Science

Duration: 2021 ->

Coordinator: Arnaud Gotlieb (arnaud@simula.no)

Partners:

- SIMULA (Norvège)

Inria contact: Mathieu Acher

Summary: The Science of Resilient Software (RESIST_EA) intends to create software-systems which can resist failures without significantly degrading their functionality. For several years, creating resilient software-systems has become extremely important in various application domains. For example, in robotics, the deployment of advanced collaborative robots which have to cope with uncertainty and unexpected behaviors while being able to recover from their failures has led to new research challenges. A recent area where these challenges have become pregnant is industrial robotics for car manufacturing where major issues faced by an “excessive automation” have surfaced. For instance, Tesla has struggled with painting, welding, assembling industrial robots in its advanced California car factory since 2018. Generally speaking, Autonomous Software-Systems (AS) such as self-driving cars, autonomous ships or industrial robots require the development of resilient software-systems as they have to manage unexpected events, such as faults or hazards. The goal of the Associate Team “Resilient Software Science” (and the main innovation of this project) is to explore the Science of resilient software by laying the ground to foundational work on advanced a priori testing methods such as metamorphic testing and a posteriori continuous improvements through digital twins.

9.1.3 Inria International Partners

Informal International Partners

- School of computer science, University of St Andrews (United Kingdom): program analysis for security, security properties, program refactoring
- UAS - Unmanned Arial Systems Center at SDU - University of Southern Denmark (Denmark): program analyses and transformations for security
- Institute of Embedded Systems at TUHH - Hamburg University of Technology (Germany): program analyses and transformations for security
- University of Luxembourg (Luxembourg): program analyses and transformations for security, information leakage
- SNE - System and Networking Engineering lab at the University of Amsterdam (The Netherlands): task scheduling for security
- The Bristol Microelectronics Research Group, University of Bristol (United Kingdom): program analyses and transformations for security
- UNIMORE - The University of Modena and Reggio Emilia (Italy): program analyses and transformations for security

9.2 International research visitors

9.2.1 Visits of international scientists

Inria International Chair

- Gunter Mussbacher has an Inria International Chair, and he is visiting the DiverSE team 4 months per year.

9.3 European initiatives

9.3.1 Horizon Europe

HiPEAC

Participants: Olivier Zendra, Jean-Marc Jézéquel.

HiPEAC project on cordis.europa.eu

Title: High Performance, Edge And Cloud computing

Duration: From December 1, 2022 to May 31, 2025

Partners:

- INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE (INRIA), France
- ECLIPSE FOUNDATION EUROPE GMBH (EFE GMBH), Germany
- INSIDE, Netherlands
- UNIVERSITEIT GENT (UGent), Belgium
- RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN (RWTH AACHEN), Germany
- COMMISSARIAT A L ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES (CEA), France

- SINTEF AS (SINTEF), Norway
- IDC ITALIA SRL, Italy
- THALES (THALES), France
- CLOUDFERRO SP ZOO, Poland
- BARCELONA SUPERCOMPUTING CENTER CENTRO NACIONAL DE SUPERCOMPUTACION (BSC CNS), Spain

Inria contact: Olivier Zendra

Coordinator:

Summary: The objective of HiPEAC is to stimulate and reinforce the development of the dynamic European computing ecosystem that supports the digital transformation of Europe. It does so by guiding the future research and innovation of key digital, enabling, and emerging technologies, sectors, and value chains. The longer term goal is to strengthen European leadership in the global data economy and to accelerate and steer the digital and green transitions through human-centred technologies and innovations. This will be achieved via mobilising and connecting European partnerships and stakeholders to be involved in the research, innovation and development of computing and systems technologies. They will provide roadmaps supporting the creation of next-generation computing technologies, infrastructures, and service platforms.

The key aim is to support and contribute to rapid technological development, market uptake and digital autonomy for Europe in advanced digital technology (hardware and software) and applications across the whole European digital value chain. HiPEAC will do this by connecting and upscaling existing initiatives and efforts, by involving the key stakeholders, and by improving the conditions for large-scale market deployment. The next-generation computing and systems technologies and applications developed will increase European autonomy in the data economy. This is required to support future hyper-distributed applications and provide new opportunities for further disruptive digital transformation of the economy and society, new business models, economic growth, and job creation.

The HiPEAC CSA proposal directly addresses the research, innovation, and development of next generation computing and systems technologies and applications. The overall goal is to support the European value chains and value networks in computing and systems technologies across the computing continuum from cloud to edge computing to the Internet of Things (IoT).

9.4 National initiatives

9.4.1 ANR

MC-Evo2 ANR JCJC

Participants: Djamel Eddine Khelladi.

- 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.

9.4.2 DGA

LangComponent (CYBERDEFENSE)

Participants: Benoît Combemale, Olivier Barais.

- 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.

9.4.3 DGAC

OneWay

Participants: Benoît Combemale, Didier Vojtisek, Olivier Barais, Jean-Marc Jézéquel, Mathieu Acher.

- 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.

MIP 4.0

Participants: Benoît Combemale, Didier Vojtisek, Olivier Barais.

- Coordinator: Safran
- Partners: Safran, Akka, Inria.
- Dates: 2022-2023
- Abstract: The MIP 4.0 project aims at investigating integrated methods for efficient and shared propulsion systems. Inria explore new techniques for collaborative modeling over the time.

9.5 Regional initiatives

IPSCo

Participants: Benoît Combemale, Didier Vojtisek, Olivier Barais.

- Coordinator: Jamespot
- Partners: Jamespot, UR1, Logpickr.
- Dates: 2022-2023
- Abstract: The IPSCo project aims at investigating new tools and methods to bring intelligence into processes and communities.

SAD CoEvoMP

Participants: Djamel Eddine Khelladi, Benoît Combemale, Arnaud Blouin.

- Coordinator: Djamel E. Khelladi
- Partners: CNRS, UR1.
- Dates: 12/2022-12/2024
- Abstract: The CoEvoMP project aims at investigating polyglot co-evolution in parallel of the MC-Evo2 project.

10 Dissemination

Participants: All the team.

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

Member of the organizing committees

- Benoît Combemale:
  - Journal-first chair for MODELS'22
- Jean-Marc Jézéquel:
  - Chair of the Most Influencial Paper Award for Software Product Lines committee
- Olivier Barais:
  - Chair of the EduSymp@MODELS 2022 (MODELS 2022 Educators Symposium) committee

10.1.2 Scientific events: selection

Member of the conference program committees

- Arnaud Blouin:
  - ACM/SIGAPP Symposium on Applied Computing (SAC), software engineering track, 2022;
  - International Workshop on Human Factors in Modeling at MODELS'2021(HuFaMo), 2022
- Stéphanie Challita:
  - The 20th International Conference on Software and Systems Reuse (ICSR 2022)
- Olivier Barais:
  - The 20th International Conference on Software and Systems Reuse (ICSR 2022)
  - The SPLASH Onward! 2022 Conference
- Benoît Combemale:
  - Program board member for MODELS'22
  - PC member for RE'22
  - PC member for ICT4S'22
  - PC member for ECMFA'22
  - PC member for EASE'22
  - PC member for FDL'22
  - PC member for QUATIC'22
- Mathieu Acher:
  - PC member for 17th International Working Conference on Variability Modelling of Software-Intensive Systems VaMoS 2023
  - PC member for 5th International Workshop on Languages for Modelling Variability (MODEVAR)
- Olivier Zendra:
  - 17th ACM International Workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems (ICOOOLPS 2022)
- Jean-Marc Jézéquel:
  - SEAMS 2022 17th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, May 23-24, 2022, Pittsburgh, USA, co-located event with ICSE.
  - SPLC 2022 The 25th International Software Product Line Conference (Industry Track), September 12-16, 2022, Graz, Austria
- Djamel E. Khelladi:
  - MSR 22
  - FSE Artifacts 22
  - 5th International Workshop on Variability and Evolution of Software-intensive Systems, at SPLC 22
  - The 16th Workshop on Models and Evolution (ME), at MODELS 22
  - 2nd International Workshop on Model-Driven Engineering for Digital Twins (ModDiT'22), at MODELS 22

10.1.3 Journal

Member of the editorial boards

- Benoît Combemale:
  - Editor in Chief of the Springer Journal on Software and Systems Modeling (SoSyM)
  - Deputy Editor in Chief of the platinum open access journal JOT on Software and language engineering
  - Member of the Editorial Board of the Springer Software Quality Journal (SQJ)
  - Member of the Editorial Board of the Elsevier Journal of Computer Languages (COLA)
  - Member of the Editorial Board of the Elsevier Journal on Science of Computer Programming (SCP, Advisory Board of the Software Section)
- Jean-Marc Jézéquel:
  - Associate Editor in Chief of the Springer Journal on Software and Systems Modeling (SoSyM)
  - Associate Editor in Chief IEEE Computer
  - Member of the Editorial Board of the Journal of Software and Systems

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

10.1.4 Invited talks

- Benoît Combemale:
  - Keynote at the international workshop on AI-native Adaptive Enterprise (KAAE)
  - Keynote at the international workshop on Models and Evolution (ME)
- Jean-Marc Jézéquel
  - Deep variability. In Invited Lecture, University of Sevilla. Sevilla, Spain, May 2022.
  - Variability management is taming uncertainty. In Workshop on Uncertainty Management, University of Malaga. Malaga, Spain, April 2022.
  - Embracing uncertainty. In Workshop on Polyglot Development and BizDevOp at the Bellairs Research Institute of McGill University. Holetown, Barbados, April 2022.
  - How deep variability challenges performance modeling. In Invited Lecture, University of Montréal. Montréal, Canada, March 2022.
  - Variability management in software engineering. In Invited Lecture, University of Ottawa. Ottawa, Canada, March 2022. 30
  - Taming variability in software engineering: Past, present and future. In Colloquia@CS, McGill University. Montréal, Canada, February 2022
- Mathieu Acher:
  - Keynote at the "Reproducible Science and Deep Software Variability" 16th International Working Conference on Variability Modelling of Software-Intensive
  - Keynote at VariVolution workshop Machine Learning and Deep Software Variability

10.1.5 Leadership within the scientific community

- Arnaud Blouin:
  - Founding member and co-organiser of the French GDR-GPL research action on Software Engineering and Human-Computer Interaction (GL-IHM).
- Benoît Combemale:
  - Founding member and Steering Committee member of the EDT.Community Seminar Series.
  - Co-organizer of the Dagstuhl Seminar 22362 on Model-Driven Engineering of Digital Twins.
  - Co-organizer of the Bellairs workshop on Polyglot Development.
  - Founding member and co-organiser of the French GDR-GPL research group on software debugging.
  - Founding member and co-organiser of the workshop series on Model-Driven Engineering of Digital Twins (ModDiT).
  - Founding member of the workshop series on Modeling Language Engineering (MLE).
- Mathieu Acher:
  - steering committee of Systems and Software Product Line Conference (SPLC)
  - steering committee of International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS)
  - steering committee of International Workshop on Machine Learning Techniques for Software Quality Evolution (MaLTeSQuE)
- Olivier Zendra:
  - founder and a member of the Steering Committee of ICOOOLPS (International Workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems).
  - Member of the EU HiPEAC CSA project Steering Committee
  - Member of the HiPEAC Vision Editorial Board
- Jean-Marc Jézéquel:
  - Vice President of Informatics Europe.
  - Member of the Executive Committee of the GDR GPL of CNRS
- Djamel E. Khelladi:
  - Co-founding member and co-organiser of the French GDR-GPL research action on Software Engineering (GT VL).

10.1.6 Scientific expertise

- Arnaud Blouin: reviewer for the ANR and CIR agencies.
- Stéphanie Challita: member of the IEEE Conference Activities Committee.
- Olivier Zendra:
  - scientific CIR/JEI expert for the MESRI
  - scientific reviewer for the HiPEAC collaboration Grants 2022
- Olivier Barais:
  - 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)

10.1.7 Research administration

- Olivier Barais led the Associate professor committee at University of Rennes 1.
- Olivier Zendra is a Member of Inria Evaluation Committee.

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

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

Each of Jean-Marc Jézéquel, Noël Plouzeau, Olivier Barais, Benoît 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 has been the head of the Computer Science department and member of the management board at the ESIR engineering school in Rennes until 08/2021, and Benoît 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.

10.2.2 Supervision

- Cassius DE OLIVEIRA PUODZIUS successfully defended on 19/12/2022 his PhD thesis in computer science at Université Rennes 1 on “Data-Driven Malware Classification Assisted by Machine Learning Methods”. Olivier Zendra has been co-director of this thesis.
- June Sallou successfully defended on 23/02/2022 her PhD thesis in computer science at Université Rennes 1 on “On Scientific Integrity and Flexibility of Scientific Software in Environmental Sciences: Towards a Systematic Approach to Support Decision-Making”. Benoît Combemale and Johann Bourcier have been co-supervisors of this thesis.
- Pierre Jeanjean successfully defended on 29/04/2022 his PhD thesis in computer science at Université Rennes 1 on “IDE as Code : Reifying Language Protocols as First-Class Citizens”. Benoît Combemale and Olivier Barais have been co-supervisors of this thesis.
- Fabien Coulon successfully defended on 03/03/2022 his PhD thesis in computer science at Université Rennes 1 on “Towards flexible Integrated Development Environment". Benoît Combemale and Olivier Barais have been co-supervisors of this thesis.
- Dorian Leroy successfully defended on 25/03/2022 his PhD thesis in computer science at Université Rennes 1 on “Behavioral Typing for the Dynamic Analysis of Executable DSLs". Benoît Combemale has been director of this thesis.
- Piergiorgio Ladisa, CIFRE with SAP (defense in 2024). Olivier Barais is the supervisor of this thesis.
- Anne Bumiller, CIFRE with Orange (defense in 2023). Benoît Combemale, Stéphanie Chalita and Olivier Barais are co-supervisors of this thesis.
- Theo Giraudet, CIFRE with Obéo (defense in 2025). Benoît Combemale and Arnaud Blouin are co-supervisors of this thesis.

- Georges Aaron Randrianaina, (defense in 2024). Mathieu Acher, Djamel Eddine Khelladi and Olivier Zendra are co-supervisors of this thesis.
- Quentin Le Dilavrec, (defense in 2024). Djamel Eddine Khelladi and Aranaud Blouin are co-supervisors of this thesis.
- Gwendal Jouneaux, (defense in 2024). Benoît Combemale and Olivier Barais are co-supervisors of this thesis.
- Luc Lesoil, (defense in 2023). Jean-Marc Jézéquel and Marhieu Acher are co-supervisors of this thesis.
- Alif Akbar Pranata, (defense in 2023). Olivier Barais and Johann Bourcier are co-supervisors of this thesis.

10.2.3 Juries

- Olivier Barais:
  - Mohammed Chakib BELGAID (reviewer), Université de Lille
  - Timothée Riom (examiner), Université du Luxembourg
  - Humberto Alvarez (reviewer), Université des pays de l'Adour
  - Honore Mahugnon HOUEKPETODJ (reviewer), Université de Lille
  - Amina CHIKHAOUI (reviewer), Université de Brest en cotutelle avec l'Université des Sciences et de la Technologie Houari Boumediene (Alger)
- Mathieu Acher:
  - SIF committee best thesis 2022
  - agrégation informatique 2022
- Djamel E. Khelladi:
  - committee member for Prix Thèse GDR GPL best thesis 2022

10.3 Popularization

10.3.1 Articles and contents

In Journal du CNRS, an article about our research about variants "un logiciel des milliards de possibilités"

In Journal du CNRS, an article about our research about incremental build Accélérer l'étude des versions d'un logiciel grâce à un assemblage incrémental

In Journal du CNRS, an article about the prize for the HyperAST research paper on scaling temporal analysis Une équipe de l'IRISA récompensée dans une prestigieuse conférence en sciences du logiciel

10.3.2 Interventions

Olivier Barais gave an invited talk at IFRI on open-source software supply chain security.

Mathieu Acher gave an invited talk at Summer School EIT Digital (july) on Mastering Software Variability for Innovation and Science

11 Scientific production

11.1 Major publications

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] D. E. Khelladi, B. Combemale, M. Acher, O. Barais and J.-M. Jézéquel. ‘Co-Evolving Code with Evolving Metamodels’. In: ICSE 2020 - 42nd International Conference on Software Engineering. Séoul, South Korea, 6th July 2020, pp. 1–13. URL: https://hal.inria.fr/hal-03029429.

[16] 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.

[17] Q. Le Dilavrec, D. E. Khelladi, A. Blouin and J.-M. Jézéquel. ‘HyperAST: Enabling Efficient Analysis of Software Histories at Scale’. In: ASE 2022 - 37th IEEE/ACM International Conference on Automated Software Engineering. Oakland, United States: IEEE, 10th Oct. 2022, pp. 1–12. URL: https://hal.inria.fr/hal-03764541.

[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] 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* 48.11 (1st Nov. 2022), pp. 4274–4290. DOI: 10.1109/TSE.2021.3116768. URL: https://hal.inria.fr/hal-03358817.

[20] G. A. Randrianaina, X. Tërnava, D. E. Khelladi and M. Acher. ‘On the Benefits and Limits of Incremental Build of Software Configurations: An Exploratory Study’. In: ICSE 2022 - 44th International Conference on Software Engineering. Pittsburgh, Pennsylvania / Virtual, United States, 8th May 2022, pp. 1–12. URL: https://hal.science/hal-03547219.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

11.2 Publications of the year

International journals

[25] M. Acher, G. Perrouin and M. Cordy. ‘BURST: Benchmarking Uniform Random Sampling Techniques’. In: *Science of Computer Programming* (3rd Jan. 2023). URL: https://hal.inria.fr/hal-03897639.

[26] R. Eramo, F. Bordeleau, B. Combemale, M. van den Brand, M. Wimmer and A. Wortmann. ‘Conceptualizing Digital Twins’. In: *IEEE Software* 39.2 (2022), pp. 39–46. DOI: 10.1109/MS.2021.3130755. URL: https://hal.inria.fr/hal-03466396.

[27] N. Harrand, A. Benelallam, C. Soto-Valero, F. Bettega, O. Barais and B. Baudry. ‘API beauty is in the eye of the clients: 2.2 million Maven dependencies reveal the spectrum of client–API usages’. In: *Journal of Systems and Software* 184 (Feb. 2022), p. 111134. DOI: 10.1016/j.jss.2021.111134. URL: https://hal.archives-ouvertes.fr/hal-03921298.

[28] D. Leroy, J. Sallou, J. Bourcier and B. Combemale. 'On the role of computer languages in scientific computing'. In: *Computing in Science and Engineering* (2022), pp. 1–6. URL: https://hal.inria.fr/hal-03799289.

[29] G. Lyan, D. Gross-Amblard, J.-M. Jézéquel and S. Malinowski. 'Impact of Data Cleansing for Urban Bus Commercial Speed Prediction'. In: *SN Computer Science* 3.82 (2022), pp. 1–11. DOI: 10.1007/s42979-021-00966-1. URL: https://hal.inria.fr/hal-03220449.

[30] G. Lyan, J.-M. Jézéquel, D. Gross-Amblard, R. Lefeuvre and B. Combemale. 'Reasoning over Time into Models with DataTime'. In: *Software and Systems Modeling* (31st Dec. 2022), pp. 1–25. URL: https://hal.inria.fr/hal-03921928.

[31] 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* 48.11 (1st Nov. 2022), pp. 4274–4290. DOI: 10.1109/TSE.2021.3116768. URL: https://hal.inria.fr/hal-03358817.

[32] X. Tërnava, J. Mortara, P. Collet and D. Le Berre. 'Identification and visualization of variability implementations in object-oriented variability-rich systems: a symmetry-based approach'. In: *Automated Software Engineering* (24th Feb. 2022), pp. 1–52. DOI: 10.1007/s10515-022-00329-x. URL: https://hal.archives-ouvertes.fr/hal-03593967.

[33] F. Zalila, F. Korte, J. Erbel, S. Challita, J. Grabowski and P. Merle. 'MoDMaCAO: a model-driven framework for the design, validation and configuration management of cloud applications based on OCCI'. In: *Software and Systems Modeling* (25th Sept. 2022). URL: https://hal.archives-ouvertes.fr/hal-03927522.

[34] S. Zschaler, E. Bousse, J. Deantoni and B. Combemale. 'A Generic Framework for Representing and Analysing Model Concurrency'. In: *Software and Systems Modeling* (2022). URL: https://hal.inria.fr/hal-03921704.

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, pp. 1–2. URL: https://hal.inria.fr/hal-03528889.

[36] M. Acher, H. Martin, J. A. Pereira, L. Lesoil, A. Blouin, J.-M. Jézéquel, D. E. Khelladi and O. Barais. 'Feature Subset Selection for Learning Huge Configuration Spaces: The case of Linux Kernel Size'. In: SPLC 2022 - 26th ACM International Systems and Software Product Line Conference. Graz, Austria, 2022, pp. 1–12. DOI: 10.1145/3546932.3546997. URL: https://hal.inria.fr/hal-03720273.

[37] L. T. van Binsbergen, D. Frölich, M. Verano Merino, J. Lai, P. Jeanjean, T. van der Storm, B. Combemale and O. Barais. 'A Language-Parametric Approach to Exploratory Programming Environments'. In: *SLE '22: 15th ACM SIGPLAN International Conference on Software Language Engineering*. SLE 2022 - 15th ACM SIGPLAN International Conference on Software Language Engineering. Auckland, New Zealand: ACM, 1st Dec. 2022, pp. 175–188. DOI: 10.1145/3567512.3567527. URL: https://hal.inria.fr/hal-03921387.

[38] C. Brown, A. Barwell, Y. Marquer, O. Zendra, T. Richmond and C. Gu. 'Semi-automatic ladderisation: improving code security through rewriting and dependent types'. In: PEPM 2022 - ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation. Philadelphia PA, United States: ACM, 16th Jan. 2022, pp. 14–27. DOI: 10.1145/3498886.3502202. URL: https://hal.inria.fr/hal-03805561.

[39] A. Bumiller, O. Barais, N. Aillery and G. Le Lan. 'Towards a Better Understanding of Impersonation Risks'. In: SINCONF 2022 - 15th IEEE International Conference on Security of Information and Networks. Sousse, Tunisia, 2022, pp. 1–9. URL: https://hal.archives-ouvertes.fr/hal-03789500.

[40] A. Bumiller, O. Barais, S. Challita, B. Combemale, N. Aillery and G. Le Lan. 'A Context-Driven Modelling Framework for Dynamic Authentication Decisions'. In: SEAA 2022 - Euromicro Conference Series on Software Engineering and Advanced Applications. Maspalomas, Spain, 31st Aug. 2022, pp. 1–8. URL: https://hal.inria.fr/hal-03729080.

[41] L. Cleophas, T. Godfrey, D. E. Khelladi, D. Lehner, B. Combemale, M. van den Brand, M. Vierhauser, M. Wimmer and S. Zschaler. 'A community-sourced view on engineering digital twins: A Report from the EDT.Community'. In: *MODELS '22: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2nd International Workshop on Model-Driven Engineering of Digital Twins (ModDiT 2022) @ MODELS 2022. WORKSHOP SESSION: 2nd International workshop on model-driven engineering for digital twins (ModDiT 2022). Montréal, Canada: ACM, 23rd Oct. 2022, pp. 481–485. DOI: 10.1145/3550356.3561549. URL: https://hal.inria.fr/hal-03933973.

[42] J.-M. Jézéquel, J. Kienzle and M. Acher. 'From feature models to feature toggles in practice'. In: SPLC 2022 - 26th ACM International Systems and Software Product Line Conference. Graz / Hybrid, Austria: ACM, 12th Sept. 2022, pp. 234–244. DOI: 10.1145/3546932.3547009. URL: https://hal.inria.fr/hal-03788437.

[43] J. Kienzle, B. Combemale, G. Mussbacher, O. Alam, F. Bordeleau, L. Burgueño, G. Engels, J. Galasso, J.-M. Jézéquel, B. Kemme, S. Mosser, H. Sahraoui, M. Schiedermeier and E. Syriani. 'Global Decision Making Over Deep Variability in Feedback-Driven Software Development'. In: ASE 2022 - 37th IEEE/ACM International Conference on Automated Software Engineering. Rochester, MI, United States: IEEE, 10th Oct. 2022, pp. 1–6. DOI: 10.1145/3551349.3559551. URL: https://hal.inria.fr/hal-03770004.

[44] P. Ladisa, H. Plate, M. Martinez, O. Barais and S. E. Ponta. 'Risk Explorer for Software Supply Chains'. In: CCS 2022 - ACM SIGSAC Conference on Computer and Communications Security. Los Angeles, United States: ACM, 8th Nov. 2022, pp. 35–36. DOI: 10.1145/3560835.3564546. URL: https://hal.inria.fr/hal-03921373.

[45] P. Ladisa, H. Plate, M. Martinez, O. Barais and S. E. Ponta. 'Towards the Detection of Malicious Java Packages'. In: CCS 2022 - ACM SIGSAC Conference on Computer and Communications Security. Los Angeles CA USA, United States: ACM, 8th Nov. 2022, pp. 63–72. DOI: 10.1145/3560835.3564548. URL: https://hal.inria.fr/hal-03921362.

[46] Q. Le Dilavrec, D. E. Khelladi, A. Blouin and J.-M. Jézéquel. 'HyperAST: Enabling Efficient Analysis of Software Histories at Scale'. In: ASE 2022 - 37th IEEE/ACM International Conference on Automated Software Engineering. Oakland, United States: IEEE, 10th Oct. 2022, pp. 1–12. URL: https://hal.inria.fr/hal-03764541.

[47] L. Lesoil, M. Acher, A. Blouin and J.-M. Jézéquel. 'Beware of the Interactions of Variability Layers When Reasoning about Evolution of MongoDB'. In: *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*. ICPE 2022 - 13th ACM/SPEC International Conference on Performance Engineering. Beijing, China, 9th Apr. 2022, pp. 1–5. DOI: 10.1145/3491204.3527489. URL: https://hal.archives-ouvertes.fr/hal-03624309.

[48] L. Lesoil, H. Martin, M. Acher, A. Blouin and J.-M. Jézéquel. 'Transferring Performance between Distinct Configurable Systems : A Case Study'. In: VaMoS 2022 - 16th International Working Conference on Variability Modelling of Software-Intensive Systems. Florence, Italy, 23rd Feb. 2022, pp. 1–6. DOI: 10.1145/3510466.3510486. URL: https://hal.inria.fr/hal-03514984.

[49] G. A. Randrianaina, D. E. Khelladi, O. Zendra and M. Acher. 'Towards Incremental Build of Software Configurations'. In: ICSE-NIER 2022 - 44th International Conference on Software Engineering – New Ideas and Emerging Results. Pittsburgh, PA, United States, 21st May 2022, pp. 1–5. DOI: 10.1145/3510455.3512792. URL: https://hal.archives-ouvertes.fr/hal-03558479.

[50] G. A. Randrianaina, X. Tërnava, D. E. Khelladi and M. Acher. 'On the Benefits and Limits of Incremental Build of Software Configurations: An Exploratory Study'. In: ICSE 2022 - 44th International Conference on Software Engineering. Pittsburgh, Pennsylvania / Virtual, United States, 8th May 2022, pp. 1–12. URL: https://hal.archives-ouvertes.fr/hal-03547219.

[51] 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. Florence, Italy, 23rd Feb. 2022. DOI: 10.1145/3510466.3510485. URL: https://hal.archives-ouvertes.fr/hal-03527250.

[52] R. Verdecchia, L. Cruz, J. Sallou, M. Lin, J. Wickenden and E. Hotellier. 'Data-Centric Green AI: An Exploratory Empirical Study'. In: ICT4S 2022 - 8th International Conference on ICT for Sustainability. 2022 International Conference on ICT for Sustainability (ICT4S). Plovdiv, Bulgaria, June 2022, pp. 1–11. DOI: 10.1109/ICT4S55073.2022.00015. URL: https://hal.archives-ouvertes.fr/hal-03632376.

[53] S. Yalles, M. Handaoui, J.-E. Dartois, O. Barais, L. d'Orazio and J. Boukhobza. 'RISCLESS: A Reinforcement Learning Strategy to Guarantee SLA on Cloud Ephemeral and Stable Resources'. In: *2022 30th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP).* 2022 30th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). Valladolid, Spain: IEEE, 9th Mar. 2022, pp. 83–87. DOI: 10.1109/PDP55904.2022.00021. URL: https://hal.archives-ouvertes.fr/hal-03921309.

Conferences without proceedings

[54] M. Acher, L. Lesoil, G. A. Randrianaina, X. Tërnava and O. Zendra. 'A Call for Removing Variability'. In: 17th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS 2023). Odense, Denmark, 25th Jan. 2023. URL: https://hal.archives-ouvertes.fr/hal-03882594.

[55] G. A. Randrianaina. 'Incremental Build of Linux Kernel Configurations'. In: EuroDW 2022 - 16th EuroSys Doctoral Workshop. Rennes, France, 5th Apr. 2022, pp. 1–3. URL: https://hal.archives-ouvertes.fr/hal-03615777.

[56] X. Tërnava, M. Acher and B. Combemale. 'Specialization of Run-time Configuration Space at Compile-time: An Exploratory Study'. In: SAC 2023 - The 38th ACM/SIGAPP Symposium on Applied Computing. Tallinn, Estonia, 27th Mar. 2023. URL: https://hal.archives-ouvertes.fr/hal-03916459.

Scientific book chapters

[57] H. Martin, P. Temple, M. Acher, J. A. Pereira and J.-M. Jézéquel. 'Machine Learning for Feature Constraints Discovery'. In: *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines.* Springer International Publishing, 5th July 2023, pp. 175–196. DOI: 10.1007/978-3-031-11686-5_7. URL: https://hal.inria.fr/hal-03921905.

Doctoral dissertations and habilitation theses

[58] F. Coulon. 'Vers un environnement de développement intégré flexible'. Université Rennes 1, 3rd Mar. 2022. URL: https://theses.hal.science/tel-03854875.

[59] P. Jeanjean. 'IDE as Code : reifying language protocols as first-class citizens'. Université Rennes 1, 29th Apr. 2022. URL: https://theses.hal.science/tel-03881947.

[60] C. Puodzius. 'Data-Driven Malware Classification Assisted by Machine Learning Methods'. Inria Rennes, 19th Dec. 2022. URL: https://hal.inria.fr/tel-03935152.

[61] J. Sallou. 'On reliability and flexibility of scientific software in environmental science : towards a systematic approach to support decision-making'. Université Rennes 1, 23rd Feb. 2022. URL: https://theses.hal.science/tel-03854849.

Reports & preprints

[62] Y. Marquer, O. Zendra and A. Heuser. *The Indiscernibility Methodology: quantifying information leakage from side-channels with no prior knowledge.* 30th Sept. 2022. URL: https://hal.inria.fr/hal-03793085.

[63] S. Yalles, M. Handaoui, J.-E. Dartois, O. Barais, L. d'Orazio and J. Boukhobza. *RISCLESS: A Reinforcement Learning Strategy to Exploit Unused Cloud Resources.* ENSTA Bretagne - École nationale supérieure de techniques avancées Bretagne, 27th Apr. 2022, pp. 1–9. URL: https://hal.archives-ouvertes.fr/hal-03652738.

Other scientific publications

[64] A. Blouin and J.-M. Jézéquel. *Journal First: Interacto: A Modern User Interaction Processing Model.* Pittsburgh / Virtual, United States, 8th May 2022. URL: https://hal.inria.fr/hal-03613422.

11.3 Other

Scientific popularization

[65] X. Tërnava, M. Acher, L. Lesoil, A. Blouin and J.-M. Jézéquel. 'Scratching the Surface of ./configure: Learning the Effects of Compile-Time Options on Binary Size and Gadgets'. In: ICSR 2022 - 20th International Conference on Software and Systems Reuse. Montpellier, France, 15th June 2022, pp. 1–18. URL: https://hal.archives-ouvertes.fr/hal-03627246.

11.4 Cited publications

[66] 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.

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

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] 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.

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

[76] 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.

[77] 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.

[78] 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.

[79] 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).

[80] 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.

[81] 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.

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

[83] 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.

[84] 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.

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

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

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

[95] P. Ladisa, H. Plate, M. Martinez and O. Barais. *Taxonomy of Attacks on Open-Source Software Supply Chains.* 2022. DOI: 10.48550/ARXIV.2204.04008. URL: https://arxiv.org/abs/2204.04008.

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

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

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

[99] 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.

[100] 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.

[101] 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.

[102] 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.

[103] 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.

[104] 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.

[105] 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.

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

[107] 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.

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

[109] 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.

[110] 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.

[111] 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.

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

[113] 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.

[114] 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.

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

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

[117] 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.

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

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

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

[121] 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.

[122] 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.

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