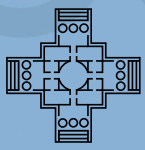
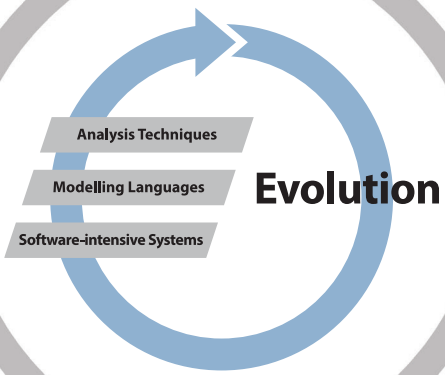
The Karlsruhe Series on
Software Design
and Quality
38



Architecture-based Evolution of Dependable Software-intensive Systems

Robert Heinrich



KIT Scientific Publishing

Robert Heinrich

Architecture-based Evolution of Dependable Software-intensive Systems

The Karlsruhe Series on Software Design and Quality
Volume 38

Dependability of Software-intensive Systems group
Faculty of Computer Science
Karlsruhe Institute of Technology

and

Software Engineering Division
Research Center for Information Technology (FZI), Karlsruhe

Editor: Prof. Dr. Ralf Reussner

Architecture-based Evolution of Dependable Software-intensive Systems

by
Robert Heinrich

KIT Scientific Publishing

Habilitation, Karlsruher Institut für Technologie
KASTEL – Institut für Informationssicherheit und Verlässlichkeit, 2022
Tag des Habilitationskolloquiums: 14. Dezember 2022

Impressum

Scientific Publishing

Karlsruher Institut für Technologie (KIT)
KIT Scientific Publishing
Straße am Forum 2
D-76131 Karlsruhe

KIT Scientific Publishing is a registered trademark
of Karlsruhe Institute of Technology.
Reprint using the book cover is not allowed.

www.ksp.kit.edu

Print on Demand 2023 – Gedruckt auf FSC-zertifiziertem Papier

ISSN 1867-0067
ISBN 978-3-7315-1294-3
DOI: 10.5445/KSP/1000157920

Abstract

Software permeates many areas of our daily life. However, this software often shows quality flaws from which critical issues may arise, as also reported manifold in the press. Recent innovations lead to complex systems that combine contributions from several heterogeneous disciplines on which software has an essential influence. Thus, these systems are denoted as software-intensive systems. Highly relevant quality properties of software-intensive systems comprise performance, maintainability, and confidentiality. Quality properties like these are commonly subsumed under the broader term dependability.

Enabling the development and, in particular, the evolution of today's and tomorrow's increasingly heterogeneous and complex, dependable software-intensive systems requires to address the following challenges. First, highly relevant quality properties strongly depend on architectural design decisions. There is little support for reasoning about quality properties, especially maintainability and confidentiality, in early development. As a result, issues can be identified only late and cause high effort. Second, not only software-intensive systems evolve but also the modelling languages and analysis techniques to reason about the systems have to evolve to satisfy emerging or changing requirements on modelling and analysis due to novel system properties. However, there is little flexibility in recent approaches to model-driven engineering. Third, software-intensive systems typically drift away from their development models due to adaptive and evolutionary changes. Observing the system in operation is a feasible approach to keep track of the system. However, there is a gap of abstraction between the data elicited by observation in operation and the architectural models used in development. Evolutionary changes of source code not reflected in architectural models is another reason for systems drifting away. Both result in loss of architectural knowledge and thus makes the initial development models increasingly less useful.

In this cumulative habilitation thesis, we propose and evaluate concepts to address these challenges. First, we describe concepts for modelling and

analysing especially maintainability and confidentiality based on architectural models of software-intensive systems early in development. Second, we present concepts for decomposition and composition of modelling languages and analysis techniques for software-intensive systems to enable more flexibility in their evolution. Third, we show concepts for bridging the divergent levels of abstraction between data of the operation phase, architectural models and source code of the development phase.

Zusammenfassung

Software durchdringt viele Bereiche unseres täglichen Lebens. Allerdings weist diese Software oft Qualitätsmängel auf, aus denen sich kritische Situationen ergeben können, über die auch vielfach in der Presse berichtet wird. Neuste Innovationen führen zu komplexen Systemen, die Beiträge aus mehreren heterogenen Disziplinen in sich vereinen, auf die Software einen wesentlichen Einfluss hat. Daher werden diese Systeme als software-intensive Systeme bezeichnet. Hochrelevante Qualitätseigenschaften software-intensiver Systeme umfassen Effizienz, Wartbarkeit und Vertraulichkeit. Qualitätseigenschaften wie diese werden gemeinhin unter dem umfassenderen Begriff Verlässlichkeit subsumiert.

Um die Entwicklung und insbesondere die Evolution dieser zunehmend heterogenen und komplexen, zuverlässigen software-intensiven Systeme heute und in Zukunft zu ermöglichen, müssen die folgenden Herausforderungen angegangen werden. Erstens, hochrelevante Qualitätseigenschaften hängen stark von Entwurfsentscheidungen bezüglich der Systemarchitektur ab. Es gibt wenig Unterstützung zur Untersuchung von Qualitätseigenschaften, insbesondere Wartbarkeit und Vertraulichkeit, in der frühen Entwicklung. Dadurch können Probleme erst spät erkannt werden und verursachen einen hohen Aufwand. Zweitens, nicht nur software-intensive Systeme evolvieren, sondern auch die Modellierungssprachen und Analysetechniken zur Untersuchung dieser Systeme müssen evolvieren, um neu entstehende oder sich ändernde Anforderungen an die Modellierung und Analyse zu erfüllen, die sich aus neuen Systemeigenschaften ergeben. Aktuelle Ansätze des Model-Driven Engineering weisen jedoch wenig Flexibilität auf. Drittens, aufgrund adaptiver und evolutionärer Änderungen weichen software-intensive Systeme typischerweise zunehmend von ihren Entwicklungsmodellen ab. Das Beobachten des Systems im Betrieb ist ein praktikabler Ansatz, um den Überblick über das System zu behalten. Es besteht jedoch eine Abstraktionslücke zwischen den durch Beobachtung im Betrieb erhobenen Daten und den in der Entwicklung verwendeten Architekturmodellen. Evolutionäre Änderungen

des Quellcodes, die nicht in Architekturmodellen widergespiegelt werden, sind ein weiterer Grund für das Abweichen von Systemen. Beides führt zu einem Verlust an Architekturwissen und macht damit die ursprünglichen Entwicklungsmodelle immer weniger brauchbarer.

In dieser kumulativen Habilitationsschrift schlagen wir Konzepte zur Bewältigung dieser Herausforderungen vor und evaluieren diese. Wir beschreiben erstens Konzepte zur Modellierung und Analyse insbesondere von Wartbarkeit und Vertraulichkeit in der frühen Entwicklung basierend auf Architekturmodellen software-intensiver Systeme. Zweitens stellen wir Konzepte zur Dekomposition und Komposition von Modellierungssprachen und Analysetechniken für software-intensive Systeme vor, um mehr Flexibilität bei deren Evolution zu ermöglichen. Wir zeigen drittens Konzepte zur Überbrückung der divergierenden Abstraktionsebenen zwischen Daten der Betriebsphase, Architekturmodellen und Quellcode der Entwicklungsphase.

Acknowledgements

First of all, I would like to thank Professor Ralf Reussner for great advice to foster my research and the creation of a pleasant and productive working atmosphere. My appreciation also goes to the reviewers of this habilitation thesis for their support and valuable time they put into reviewing this work.

I would like to thank all the employees of the DSiS research group for exciting cooperation and the nice team spirit. Special appreciation goes to the doctoral researchers I supervised and I am currently supervising, Doktor Kiana Busch, Doktor Misha Stittmatter, Doktor Stephan Seifermann, Emre Taspolatoglu, Maximilian Walter, Sandro Koch, Frederik Reiche, and Sebastian Hahner, as well as to the students I supervised, especially Alessandro Giusa, Tobias Pöppke, Nicolas Boltz, and David Monschein, for intensive and fruitful collaboration, helpful discussions, and valuable contributions to my research.

I would also like to thank all my collaboration partners who supported my research in various ways in the last years, especially Professor Tomáš Bureš, Professor Francisco Durán, Professor Petr Hnětynka, Professor Bernhard Rumpe, Doctor Carolyn Talcott, Professor Birgit Vogel-Heuser, and Doktor Steffen Zschaler, as well as the German Research Foundation (DFG), the German Federal Ministry of Education and Research (BMBF), the Ministry of Science, Research and the Arts Baden-Württemberg (MWK BW), the Karlsruhe Institute of Technology, and my industrial collaboration partners.

Last, but not the least, I would like to thank my family for their support while preparing this habilitation thesis. Special thanks to my wife and children for their encouragement and patience during this time-intensive endeavour.

Contents

List of Figures

1 Introduction

This chapter starts with a motivation of the topic of this habilitation thesis in Section 1.1. Then, Section 1.2 introduces the research goals and questions to be addressed. Section 1.3 gives an overview of contributions. The outline of the thesis is described in Section 1.4.

1.1 Motivation

Software is an essential part in various facets of our daily life. Mobility, production, energy supply, economics, and infrastructure, to name only a few examples, strongly depend on software. This software is not always of high quality. Critical issues that arose from poor software quality are even reported manifold publicly in the press. For example, Denver International Airport opened, delayed, and over budget, due to a dysfunctional automated baggage-handling system [75]; a new online banking system at TSB Bank led to access and confidentiality issues for its customers, and eventually forced the CEO to step down [182]; and a supply-chain attack against SolarWinds inserted a Trojan Horse into installation packages enabling attackers to access customers' systems running the affected products [126] [1].

Recent innovations, like the Internet of Things, production automation, smart mobility, and cyber-physical systems, brought forth systems that combine contributions from several heterogeneous disciplines, such as software, electrics/electronics, and mechanics, and are involved in complex organisational processes. In these systems, software contributes an essential influence factor on parts of the system from other disciplines. Therefore, these systems are denoted as software-intensive systems [102].

[1] Parts of this motivation are taken from the Introduction chapter of my book Composing Model-Based Analysis Tools [114].

Besides functional correctness, highly relevant quality properties for today's and tomorrow's software-intensive systems include, for instance, performance, as directly perceived by users, for example in form of response time when invoking system services, maintainability, as significant decision factor for system evolution, and confidentiality, as important legal constraint for system design [114].

Quality properties like the aforementioned are commonly grouped under the broader term *dependability* [20, 83]. Dependability of a computer system is generally understood as the "justifiable confidence that it will perform specified actions or deliver specified results in a trustworthy and timely manner" [195]. Dependability has various facets and there are many different quality properties that make a system dependable. An overview is given in [83]. In order to keep this thesis concise, we focus on three very different but equally important quality properties of dependable software-intensive systems — performance, maintainability, and confidentiality. *Performance* refers to timing behaviour and resource efficiency of a system. Performance (efficiency) is defined in [128] "relative to the amount of resources used under stated conditions". *Maintainability* is commonly understood as the capability of a system to be modified. Maintainability is defined as "degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers" in [128]. *Confidentiality* ensures that "information is not made available or disclosed to unauthorised individuals, entities, or processes" [129].

As described by Eusgeld and Freiling [83], the inclusion of performance in a dependability definition reaches back to a standard of the ISO/Technical Committee 176 [131] and, according to Avizienis et al. [20], comes from a definition from the telephony area. Maintainability has already been considered as a property of dependability by Avizienis et al. [20] when discussing threats to dependability that may be evoked by maintaining the system. Confidentiality has been mentioned by Avizienis et al. [20] when discussing the relationship between dependability and security. Building upon this work, confidentiality is listed as a property of dependability in [83].

1.1.1 Architecture-based Modelling and Analysis of Dependability

The quality properties performance, maintainability, and confidentiality strongly depend on design decisions regarding the system architecture. For the domain of information systems [189] it has been shown that the influence of software architecture design on quality properties is so strong, that knowledge about the architecture is sufficient to make good quality predictions [206, 248]. A *software architecture* specifies the "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" [130]. Consequently, a software architecture represents on the one hand a structural plan of the system and on the other hand a set of documented design decisions [206]. In literature, several situations have been described in which engineers benefit from analysing quality properties based on the software architecture [206, 248]. For example, when making design decisions regarding system structuring or defining component boundaries, analyses enable to predict the implications of these design decision on quality properties. Another example is refactoring or evolution (see Chapter 2) where it is easier and more efficient to investigate the implications of design decisions on the quality properties of the systems by modelling and analysing the architectural design compared to implementing the change in source code and investigating implications in retrospect.

Given the definition of software architecture and the benefits it brings for quality prediction, there is nothing which restricts these to pure software systems. Our assumption is that the notion of architecture-based modelling and analysis of quality properties can be applied to software-intensive systems in general. This is because both, software architecture as well as system architecture, represent (i) a structural plan (or a collection of structural plans) in terms of elements and relationships, and (ii) a set of design decisions which makes both a reasonable basis for quality analysis.

In order to ensure high quality of software-intensive systems, research and practice is interested in approaches to analyse the system design for quality properties. This needs to be done already in early development to identify and fix emerging issues efficiently. Studies on software engineering economics show the earlier issues can be fixed, the more cost-efficient the fix is [37]. Owing to the strong dependencies of quality properties on architectural

design decisions, the system architecture is one of the earliest artefacts based on which quality analysis can be conducted reasonably.

While there is research on performance engineering on architectural level for decades that brought forth promising approaches (e.g., [231, 280, 206]), there is little support for modelling and analysing the quality properties maintainability and confidentiality on architectural level. Furthermore, current approaches (discussed in Section 3.1) are limited to software and neglect dependencies to other parts of a software-intensive system and its context.

1.1.2 Evolution of Modelling Languages and Analysis Techniques

Although modelling and analysis are powerful means to enable well-founded design decisions in development, adaptation, and evolution (see Chapter 2) of software-intensive systems, they lead to new challenges, because modelling languages and analysis techniques also have to evolve in order to satisfy new or changing requirements that result from novel system properties. For example, the appearance of self-adaptive systems has prompted changes as this kind of systems cannot be investigated properly using strict models and static analysis techniques. There is a huge variety of different languages for modelling software-intensive systems and for analysing their quality properties (to be used depending on the size and complexity of the systems and the available details [114]). These languages are specific to individual quality properties, types of systems, tools, or development paradigms. Developing a language that encompasses all possible combinations of these aspects is just infeasible in practical settings. Initiatives to unify and extend existing modelling languages and analysis techniques in the past have resulted in languages and analysis techniques that were hard to maintain and overly complex. However, when comparing different languages for modelling and analysing different quality properties of software-intensive systems, substantial overlaps can be identified, for example, for modelling structure and behaviour of the systems [109].

Recent software-intensive systems combine contributions from several disciplines, such as software, electrics/electronics, mechanics, and organisational processes. Consequently, also the analyses for each of these individual disciplines need to be combined to adequately investigate the overall behaviour

and quality of the system. The purpose-specific composition of modelling languages and analysis techniques is a challenging but unavoidable issue for today's and tomorrow's increasingly heterogeneous and complex systems [114].

Existing modelling languages and analysis techniques, however, often show monolithic structures. They are seldom designed so that parts can be extended and reused for specific purposes in different contexts. Indeed, an approach like Palladio [206], for example, already provides a modelling language and analysis techniques for reasoning about performance, reliability, maintainability, and other quality properties based on software architecture design [117]. However, the Palladio approach, like similar model-driven approaches, relies on a monolithic modelling language and monolithic analysis techniques. This makes modification, extension, and partial reuse of the modelling language and analysis techniques challenging when providing support for new quality properties and disciplines [114]. In consequence, the internal structure of Palladio's modelling language and tools implementing the analysis techniques eroded over time due to, for example, feature overload, feature scattering, uncontrolled growth of dependencies, instance incompatibility, and incompatible extensions [241, 113]. Even more importantly, there is no support for leveraging commonalities between analysis techniques provided.

1.1.3 System Development and Operation

The quality of software-intensive systems must be ensured not only during the initial development (also called design time, development time, or Dev-time in literature) but also must be maintained during system operation (also called runtime, operation time, or Ops-time in literature). Throughout their lifetime these systems must react to a variety of different events, such as outage of resources, peak loads, emerging requirements, and changes of infrastructures or third party services [107]. Addressing these events requires strong interaction of evolution activities on development level and adaptation activities on operation level. Examples of adaptive and evolutionary changes include the replication or migration of a database component [106], replacement of one algorithm implementation by another [202], or more profound modifications of the system design like adding a new feature or changing the architecture style, for example, moving from monolithic system to microservice architecture [107].

Architectural models that are used in development of a software-intensive system usually differ from those created during operation of the system with regard to purpose, content, and, in particular, the level of abstraction [110]. Consequences of these differences are limited reusability of development models in operation, loss of knowledge about the architecture, and little phase-spanning consideration of the architecture [107]. Once software-intensive systems are in operation, they typically drift away from their initial development models due to adaptive and evolutionary changes [187]. Runtime models [28] can be kept in sync with the system in operation. However, typical runtime models are close to the level of abstraction of the implementation in source code [258, 28]. Although useful for self-adaptation, for example, such a low level of abstraction makes it difficult for humans to understand the system and keep overview of the system as it evolves. Moreover, runtime models may grow in detail or become unnecessarily complex due to various modifications throughout the system's lifetime. This severely limits comprehensibility of this kind of runtime models for humans during system evolution [107].

Furthermore, there is a risk that source code of the implemented system drifts away from its architectural models in case changes are not properly reflected due to the increasing dynamics of adaptive and evolutionary changes and development cycles that get increasingly shorter due to the growing popularity of practices such as DevOps [48].

1.2 Challenges, Objectives, and Research Questions

Enabling the development and, in particular, the evolution of today's and tomorrow's increasingly heterogeneous and complex, dependable software-intensive systems requires to address the challenges described in this section.

Challenge 1, Little support for early reasoning about quality properties: Highly relevant quality properties strongly depend on architectural design decisions. There is little support for reasoning about quality properties, especially maintainability and confidentiality, in early development. As a result, issues can be identified only late or even occur once the system is

in operation. This typically leads to dramatic increase in effort required for fixing these issues [37]. As there is little support for reasoning about quality properties in early development, there is also little support for comparing design alternatives based on these quality properties in early development. Again, this may result in unfavourable design decisions that, in turn, create high effort in case they need to be revised later. Furthermore, little support for reasoning about quality properties and for comparing design alternatives in early development may result in the need to implement prototypes to estimate the quality properties before developing the actual system. This is costly in any case and infeasible in practical settings for some quality properties, especially for maintainability and confidentiality.

Challenge 2, Little flexibility of modelling languages and analysis techniques: Not only the software-intensive system itself evolves but also the modelling languages and analysis techniques to reason about the system have to evolve to satisfy emerging or changing requirements on modelling and analysis. However, there is little flexibility in recent approaches to model-driven engineering for purpose-specific creation, extension, and reuse of modelling languages and analysis techniques. This is in contrast to object-oriented software design (as coined for example by Grady Booch [42]) where decomposition and composition concepts are established. There exist several commonalities between object-oriented software design and engineering of modelling languages (especially metamodel design) and analysis techniques, for example in the notion of classes, their attributes and dependencies, inheritance hierarchies, and packaging (a detailed discussion is given in [241]). However, the transfer of concepts from object-oriented software design to support decomposition and composition of modelling languages and analysis techniques is yet to show.

Challenge 3, Divergent levels of abstraction between development and operation: Software-intensive systems typically drift away from their initial development models due to adaptive and evolutionary changes. Observing the system in operation is a feasible approach to gather information about behaviour and status of the system and even about its context. Based on the concept of a control loop (such as the MAPE loop [282]) the system in operation is observed to gather this information and to plan and execute adaptions to handle upcoming issues. However, there is a gap of abstraction between the data elicited by observation in operation and the architectural models used in development. Evolutionary changes of source code without changing the architectural model is another reason for software-intensive

systems drifting away from their initial architectural models. Both result in loss of architectural knowledge and thus makes the initial development models increasingly less useful or even entirely useless for reasoning about quality properties of the system for upcoming changes.

In order to address aforementioned challenges we derive the following objectives and research questions of the thesis:

Objective 1, Modelling and analysing maintainability and confidentiality on architectural level: The first objective of this work is to allow for reasoning about dependability of software-intensive systems early in development by investigating concepts for modelling and analysing especially maintainability and confidentiality based on the system architecture. This objective addresses Challenge 1 (little support for early reasoning about quality properties) and results in the following research questions:

Research Question 1.1: How can the maintainability of software-intensive systems be modelled and analysed based on the system architecture?

Research Question 1.2: How can the confidentiality of software-intensive systems be modelled and analysed based on the system architecture?

Objective 2, Decomposition and composition of modelling languages and analysis techniques: Another objective of this work is to provide more flexibility in model-driven engineering of software-intensive systems by investigating concepts for decomposition and composition specifically for model-based analyses of software-intensive systems. This objective addresses Challenge 2 (little flexibility of modelling languages and analysis techniques). In order to achieve this objective we raise the following research question:

Research Question 2: How can decomposition and composition concepts as known from object-oriented software design be transferred to modelling languages and analysis techniques for reasoning about the dependability of software-intensive systems?

Objective 3, Bridging the divergent levels of abstraction between development and operation: The last objective of this work is to preserve architectural knowledge in adaptation and evolution of software-intensive systems by investigating concepts for bridging the divergent levels of abstraction between data of the operation phase, architectural models and source

code of the development phase. This objective addresses Challenge 3 (divergent levels of abstraction between development and operation) and results in the following research question:

Research Question 3: How to bridge the divergent levels of abstraction between data of the operation phase, architectural models and source code of the development phase?

1.3 Overview of Contributions

This section gives a brief overview of contributions described in this thesis. A detailed introduction and discussion of the contributions is given in Chapter 4.

In order to answer Research Question 1.1, we investigate the maintainability of a software-intensive system by analysing the impact of changes while propagating through the system. The maintainability of a software-intensive system correlates with the impact of certain changes in the system. A change in the system may cause further changes. This is denoted as *change propagation*. Consequently, the maintainability of a system is determined not only by characteristics of the system but also by the changes the system faces [50]. We exemplary focus on two different kinds of software-intensive systems: (i) automated production systems as cross-disciplinary systems and special class of mechatronic systems [219, 41] that besides software typically comprise electrical/electronical and mechanical parts, and (ii) information systems [189] as software systems that are closely involved in business processes [281] and an organisational environment. We chose these two kinds of software-intensive systems because, due to their differences, they represent extreme poles and therefore open up a space in which other systems can be classified. First, we propose an approach to architecture-based change impact analysis in automated production systems. We specify modelling languages to model the automated production system on architectural level and maintenance-relevant annotations. We investigate algorithms and rules for change propagation analysis. This contribution is described in [112] and Chapter 5 of this thesis. Moreover, we propose an approach to architecture-based change impact analysis in business processes and involved information systems building upon the modelling concepts in [115]. We specify modelling languages and investigate algorithms and rules that allow to consider the

mutual dependencies between business processes and information systems in change propagation analysis. This contribution is described in [211] and Chapter 6 of this thesis.

In order to answer Research Question 1.2, we investigate the architecture-based modelling and analysis of confidentiality. Confidentiality is closely related to data and data processing which is handled by the software parts of a software-intensive system. Consequently, we focus on the software parts of a software-intensive system to answer this research question. We first introduce data flows in an architecture description language [130] to describe data and data processing. This enables the investigation of confidentiality issues in the architectural model by proposing an analysis technique using logic programming. This contribution is described in [227] and Chapter 7 of this thesis. Moreover, we propose an extended data flow diagram syntax that supports modelling both, information flow and access control, in the same language as well as data flow diagram semantics that support various types of confidentiality analyses. This contribution is described in [228] and Chapter 8 of this thesis. Further, we propose an architecture-based approach for analysing the dependencies between system vulnerabilities and access control policies to identify attack paths and confidentiality issues. Note, we focus on cyber attacks in this thesis and do not consider physical attacks. This contribution is described in [269] and Chapter 9 of this thesis.

In order to answer Research Question 2, we investigate the applicability of decomposition and composition concepts known from object-oriented software design and the idea of a reference architecture known from software engineering to metamodels for quality modelling and analysis of software-intensive systems to systematically create, extend, and reuse metamodel parts. This contribution is described in [109] and Chapter 10 of this thesis. Furthermore, we investigate the composition of analysis techniques with a special focus on performance. We propose concepts for modelling and analysing business processes and human actor behaviour and discuss composition operators for performance simulators for business processes and information systems. This contribution is described in [115] and Chapter 11 of this thesis. We generalise these composition operators for performance simulators in Chapter 4 and give examples of how these are implemented in existing simulators in [113].

In order to answer Research Question 3, we first propose foundations to keep the architectural model consistent with the system in operation by considering operation-level adaptation and development-level evolution as two mutual

interwoven processes. We develop the notion of an architectural runtime model that is usable for automated adaptation and is simultaneously comprehensible for humans during evolution. This contribution is described in [106] and Chapter 12 of this thesis. Then, we detail and extend the modelling concepts introduced in [106] with regard to correspondence maintenance, transformation pipeline specification, and workload modelling to align architectural models used in development and operation. This contribution is described in [107] and Chapter 13 of this thesis. Furthermore, we propose an approach to keep various representations of the system — architectural model, source code, and monitoring data — consistent throughout evolution and adaption building upon [107]. The approach allows for self-validation and reduces monitoring overhead while observing the system in operation. This contribution is described in [184] and Chapter 14 of this thesis.

1.4 Outline

The remainder of this thesis is organised as follows. In Chapter 2, we introduce relevant terms and definitions. Chapter 3 discusses the state of the art with respect to the contributions proposed in this thesis. In Chapter 4, we first give an overview of the publications that are part of this cumulative thesis grouped by the objectives of this thesis. Then, the contributions of this thesis are introduced and placed in context before the single contributions are described in detail in the following chapters.

The next chapters represent the publications that together form this cumulative thesis.

- The publication about architecture-based change impact analysis in cross-disciplinary automated production systems is included as Chapter 5 in this thesis.
- Chapter 6 represents the publication about architecture-based change impact analysis in information systems and business processes.
- The publication that introduces data flows in an architecture description language for investigation of confidentiality issues is included as Chapter 7.

- Chapter 8 represents the publication that describes our approach to detect violations of access control and information flow policies in data flow diagrams.
- The publication about architectural attack propagation analysis for identifying confidentiality issues is included as Chapter 9.
- Chapter 10 represents our approach to decomposition and composition concepts and the reference architecture for metamodels to tailor quality modelling and analysis.
- The publication about composing business process simulation and information system simulation for performance prediction is represented in Chapter 11.
- Chapter 12 represents the publication about first attempts to architectural runtime models for bridging the divergent levels of abstraction in modelling between development and operation.
- The publication about extensions and elaborations of our approach to architectural runtime models is represented in Chapter 13.
- Chapter 14 represents the publication about enabling consistency between software artefacts for software adaption and evolution.

Finally, Chapter 15 gives a summary of the contributions proposed in this thesis and an outlook of future research topics and research directions.

2 Terms and Definitions

This chapter introduces terms and definitions relevant in the thesis. We first present basic terms and their definitions in Section 2.1. Then, we introduce terms and definitions related to model-based analyses in Section 2.2.

2.1 Basic Terms

Important basic terms that need to be clarified before we go further into the details of the thesis are discussed in this section.

A central term in this thesis is software-intensive system. A *software-intensive system* denotes a system on which software contributes essential influence [102]. Automated production systems, information systems, or systems from automotive are examples of software-intensive systems.

The term *quality property* refers to the ISO/IEC 25010 quality models [128]. Examples of quality properties include performance, reliability, maintainability, and confidentiality.

The term *domain* refers to one or several *discipline(s)* (i.e. branch of knowledge), which comprise (parts of) systems providing different functionality, but addressing similar business and technical needs, and are subject to common requirements and terminology (cf. [50]). For example, the domain of automated production systems typically comprises the disciplines software, electrics/electronics, and mechanics.

Refactoring is the process of improving the internal structure of a software to make it easier to understand and cheaper to modify without changing its observable behaviour [90].

The term evolution in the context of software first appears with Lehman [164]. *Evolution* of a software-intensive system is commonly understood as the

continual development after the system has been initially released to satisfy new or changed requirements.

In our research, we distinguish evolution and *adaptation*. This is inspired by Oreizy et al. [192] and described as follows: Evolution activities are conducted by human developers in a non-automated or partly automated way to implement perfective, adaptive, or corrective changes to the system [168]. Adaptation activities are performed fully automatically by predefined procedures where possible without human intervention [107].

2.2 Modelling and Analysis Terms

The concepts introduced in this thesis for the decomposition and composition of model-based analyses are shown in Figure 2.1 and explained in the following[1].

A *model-based analysis* is a type of analysis that uses models for reasoning about the system and for communicating the results [114].

A *modelling language* is created and applied to specify models to efficiently design and reason about systems [121]. Modellers can use standardised languages, such as UML [214] or SysML [92], or they can design their own domain-specific modelling languages (DSMLs) [91, 59]. An *architecture description language* is a specific kind of modelling language used to describe a system architecture [130]. A *modelling language feature* is an abstraction of a thing to be modelled [109].

An *analysis technique* is applied for reasoning about structure, behaviour, and/or quality of systems based on a model. Various different analysis techniques are possible, for example, based on queuing networks or Markov chains. While we on the one hand belief the hypothesis that analysis techniques can (to a large extent) be decomposed into individual, reusable algorithms described in form of their analysis features, we on the other hand in detail investigate in the interactions between analysis techniques in our research, because these interaction points are the glue to compose analysis techniques to higher system understanding.

[1] I would like to express my appreciation to Professor Bernhard Rumpe for discussion and feedback on the definitions presented in this section.

An *analysis feature* is an abstraction of a property to be analysed. For sake of compositional reuse, it is of high interest to use analysis features, operating on language features. Analysis features are needed to decompose the increasing complexity of analysis techniques.

A *simulator* is a software tool that implements one or more techniques of simulative analysis for approximating the behaviour and/or quality properties of a system under study [113]. A *simulation* is the execution of a simulative technique using a simulator [113]. Simulation is therefore an example of an automated analysis (cf. Chapter 2 of [114]).

A *feature model* [64] is a formalism to capture the variability and interdependencies of features of a specific subject [109]. Based on a feature model, subsets of the given features are selected to specify which features are of current interest, for example, to be composed. In [109], we use feature models to specify interdependencies between language features and select those of current interest for language composition. In this thesis, we apply our notion of feature models to two dimensions, (a) modelling languages and (b) analysis techniques, and use them to specify the interdependencies between analysis features, language features, and from analysis features to language features as well as to select those features (both analysis and language features) of current interest. As shown in Figure 2.1, parent-child relationships form a tree allowing, for example, type mandatory, optional, and alternative feature groups [64]. Language features are implemented by modelling language components. A *modelling language component* describes language constituents, for example through metamodels or grammars, has explicit interfaces and composition operators [53, 109] for other modelling language components, and has an individual, composable semantics. Analysis features are implemented by analysis components. An *analysis component* contains analysis algorithms realised in source code. These analysis components are executable on the needed language features and have explicit interfaces and composition operators for other analysis components.

Modelling language composition is a combination of sub-languages into one complete language, where the individual sub-models adhere to their sub-languages and the complete model derives its syntax and semantics from the composed sub-languages [215, 121]. Language features can be selected from the feature model visualised in Figure 2.1 to configure and extend a modelling language. By selecting language features from the feature model their associated language components are composed.

We define *modelling language decomposition* as a split of one complete language into individual sub-languages. In accordance, a model that adheres to the complete language can be projected into individual sub-models adhering to their sub-languages.

The decomposition of analysis techniques first builds on the decomposition of modelling languages and especially on the models (the analysis input) as defined before. Second, the analysis techniques can themselves be decomposed (and then also composed) in various ways as defined in the following. We define *analysis technique composition* as a combination of sub-analysis techniques into one complete analysis technique, where (1) the individual sub-models adhere to their sub-languages, (2) the individual sub-results adhere to their sub-analysis techniques, and (3) an appropriate "orchestration" of sub-analysis techniques defines the order and use of sub-results into a complete result.

We distinguish three general forms of composition in our previous work [246]: *Model composition* (white-box composition) is the analysis input model composition realised by language integration. *Result composition* (black-box composition) is the composition of the analysis results by orchestrating encapsulated analyses. *Analysis composition* (grey-box composition) is the composition of the analysis techniques by orchestrating the steps of two or more analysis algorithms.

Analysis features can be selected from the feature model visualised in Figure 2.1 to configure and extend a model-based analysis. By selecting analysis features from the feature model their associated analysis components are composed.

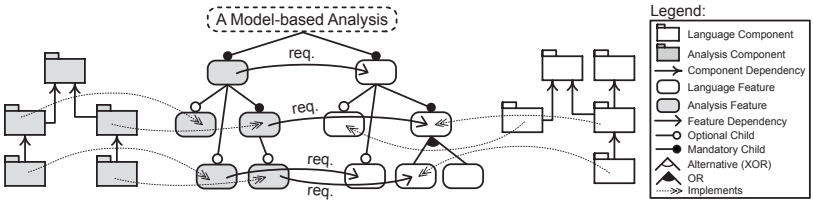


Figure 2.1: Relationships between concepts of modelling languages and analysis techniques: analysis features that are implemented by analysis components require (req.) modelling language features that are implemented by modelling language components (some elements omitted for clarity).

We define *analysis technique decomposition* as a split of one complete analysis technique into individual sub-algorithms, where (1) a model that adheres to the language of the complete analysis technique can be projected into individual sub-models adhering to the languages of the sub-analysis techniques (cf. modelling language decomposition), and (2) the result of the complete analysis technique arises out of the results of the individual sub-algorithms by appropriate orchestration of the sub-analysis techniques.

3 State of the Art

This chapter gives an overview of the state of the art related to the contributions proposed in this thesis. In Section 3.1, we discuss work on quality modelling and analysis on architectural level with a special focus on maintainability and confidentiality. The decomposition and composition of modelling languages and analysis techniques is discussed in Section 3.2. Work on bridging the divergent levels of abstraction in modelling between development and operation is discussed in Section 3.3. The chapter concludes with Section 3.4.

3.1 Architecture-based Modelling and Analysis of Dependability

Many quality properties of a software-intensive system are influenced by the system's architectural design. For the quality properties and disciplines we consider in this thesis this influence is strong enough that knowledge about the architectural design can be used for adequately predicting these quality properties for the system once it is in operation, even if only the architectural design and relevant context specifications are used as the basis for prediction.

Approaches to architecture-based quality modelling and analysis either build upon domain-specific modelling languages or quality-specific extensions to general purpose modelling languages, such as UML [214] and SysML [92]. Examples of domain-specific modelling languages are the Palladio Component Model (PCM) [206] for modelling component-based software systems, AutomationML [18] for modelling automated production systems, MECHATRONICUML [79] for modelling software embedded in mechatronic systems, and BPMN [190] for modelling business process designs. Given the PCM as an example, there exist several language extensions for which different

analyses for various quality properties like performance [26, 177, 24], reliability [44], scalability and elasticity [167], maintainability [210], security [247], and energy consumption [240] are available. Examples of quality-specific extensions to general purpose modelling languages are UML MARTE [191] for performance modelling and UMLSec [137] for security modelling.

Analysis techniques for predicting various quality properties build upon the modelling languages. These analysis techniques enforce individual models for each desired form of analysis. Examples of analysis techniques are based on queuing networks, Petri nets, Markov chains, data flows, and fault trees. For aforementioned quality-specific extensions of the PCM examples of analysis techniques can be found in [44, 167, 240, 210].

In the following, we focus on the state of the art with respect to modelling and analysing the quality properties maintainability and confidentiality for software-intensive systems.

3.1.1 Architecture-based Modelling and Analysis of Maintainability

Often a software-intensive system is in operation for several decades while facing various modifications over time. These modifications include, for example, replacement of parts due to physical abrasion, platform changes due to environmental conditions, and software evolution due to emerging requirements [112]. The software-intensive system must be designed to be repeatedly changed in a cost-efficient way. Consequently, maintainability refers to the effort required for implementing changes to the system [210]. Typically, a system is understood to be the more maintainable the less effort it takes to implement a change. The effort required for implementing a change to the system depends on the *change impact*, i.e. the artefacts affected by the change while propagating through the system. Each affected artefact causes effort by activities performed to change the artefact.

A software-intensive system may involve, besides software, artefacts of other disciplines, such as electrics/electronics or mechanics, and is typically involved in business processes to add value to an organisation. All the different parts of a software-intensive system and the processes the system is involved in mutually affect each other. Consequently, the discussion of the state of

the art comprises approaches from three categories: (i) change propagation modelling and analysis in software systems (Section 3.1.1.1), (ii) change propagation modelling and analysis in automated production systems as an example of cross-disciplinary software-intensive systems (Section 3.1.1.2), and (iii) change propagation modelling and analysis in business processes and involved information systems (Section 3.1.1.3). While we in the following give an overview of the state of the art further details are described in [50] and [239].

3.1.1.1 Change Propagation Modelling and Analysis in Software Systems

Various *graph-based approaches* to change propagation analysis in software systems exist. A comprehensive literature overview and classification has been provided by Lehnert [166, 165]. One of the most established graph-based approaches to change propagation analysis is program slicing originally introduced by Weiser [275]. A program slice denotes "the parts of a program that (potentially) affect the values computed at some point of interest" [252]. Program slicing is the method of reducing a program to the program slice [275, 252]. Efficient realisations of program slicing exist based on a Program Dependence Graph (PDG) [193, 98]. In [161] and [138], the concept of program slicing is generalised for slicing methods in UML design models. Another approach by Reps [205] identifies attributes affected by a change based on dependency graphs. Further, approaches exist with a special focus on object-oriented programs, such as the approach by Ryder and Tip [218] based on call graphs.

The literature review by Lehnert [166] revealed that 65 percent of the investigated approaches to identify change impacts are based on source code neglecting other representations of the software system. In contrast, we focus on architecture-based change propagation analysis in this thesis in order to identify change impacts already early in development. Moreover, all these approaches obviously are limited to software and do not consider parts of a software-intensive system from other disciplines.

Architecture-based approaches related to analysing change impacts in software systems comprise (i) architecture-based project planning, (ii) architecture-based software evolution, and (iii) scenario-based software architecture analysis.

Approaches to *architecture-based project planning* use the software architecture as an artefact in project planning. For example, the approach Architecture-Centered Software Project Planning (ACSPP) [196] estimates cost and schedule in a project using architectural knowledge. Carbon [55] proposes an approach to align software architecture and project planning. However, to the best of our knowledge no approach in this category supports change propagation analysis.

Examples of approaches to *architecture-based software evolution* are those by Garlan [94] and Naab [188]. The approach by Garlan [94] addresses the evolution of a given software architecture and the reasoning about the quality of evolution paths. The approach by Naab [188] analyses the flexibility of a software architecture for modification. However, these approaches do not support change propagation analysis.

Approaches to *scenario-based software architecture analysis* investigate the impact of a change based on change scenarios to a given architectural model. The Software Architecture Analysis Method (SAAM) [141] analyses the maintainability (modifiability in [141]) of the software architecture by investigating components and their connections as well as data flows. The Architecture Tradeoff Analysis Method (ATAM) [142] as a successor of SAAM allows making trade-offs between various quality properties like maintainability and performance. The approach Architecture-Level Prediction of Software Maintenance (ALPSM) [30] estimates the impact of a change based on the size of software components using weighted change scenarios. Similarly, Architecture-Level Modifiability Analysis (ALMA) [29] estimates the impact of change scenarios on a software architectural model. Change impact analysis in these approaches is limited to artefacts of the system structure and associated activities. Besides these structural artefacts, however, there are various technical and organisational artefacts, such as test cases and staff specifications, that may be affected by a change and thus cause activities, such as updating test cases and assigning tasks to responsible staff [210]. Consequently, they need to be considered in change impact analysis.

The Karlsruhe Architectural Maintainability Prediction (KAMP) approach [210, 239] goes beyond aforementioned approaches by using comprehensive architectural models that allows to include technical and organisational artefacts, besides structural artefacts, in estimation. KAMP like all approaches discussed before, however, is limited to software systems and does not con-

sider, for example, electrical/electronical or mechanical artefacts that are often part of software-intensive systems.

There are also approaches to change impact analysis in software systems working on UML design models, such as class diagrams, sequence diagrams, or statechart diagrams. For example, Briand et al. [43] propose an approach based on UML design models using OCL-based change propagation rules and distance measures to identify and prioritise parts of the system affected by a change. Dam and Winikoff [65] create repair plans for each identified violation of an OCL constraints in a UML design model. The cost of the repair plans are calculated and the cheapest plan is identified. However, these approaches are restricted to UML design models and do not support change propagation analysis on the architectural level. Furthermore, there are various approaches based on UML design models, for example [143, 235, 279, 197], or domain-specific design models, for example [169], that are limited to comparing and analysing model differences but do not analyse change propagation.

3.1.1.2 Change Propagation Modelling and Analysis in Automated Production Systems

Automated production systems are one kind of software-intensive systems we exemplary focus on in this thesis. A wide-spread approach for decades was estimating change impacts in automated production systems based on counting the number of input or output signals [261] without considering any architectural insights. Only in recent years modelling languages are used to manage complexity and estimate change impacts in these systems. Approaches related to change propagation modelling and analysis in automated production systems exist based on both, general purpose modelling languages and domain-specific modelling languages.

Examples of *approaches based on general purpose models* (UML and SysML models) are discussed in the following. For analysing change propagation in the software modules of an automated production system aforementioned approaches based on UML design models (e.g., [43, 65]) are widely used. As these approaches have been developed for software systems, they obviously cannot consider change propagation from the software parts to other parts of a software-intensive systems and vice versa. When replacing one software module of the system by another widely used approaches to analyse and

compare differences between the two modules are the aforementioned ones based on UML design models (e.g., [143, 235, 279, 197]). Again, as these approaches have been developed for software systems they are limited to software modules and cannot investigate impacts on other parts of the system. Moreover, there are various modelling approaches for automated production systems based on SysML, such as [251, 22, 84]. None of these approaches support the analysis of change propagation to the best of our knowledge.

In [169], differences between domain-specific models and UML models are discussed as follows. In contrast to UML, (i) domain-specific modelling languages allow for creating models tailored to specific application areas. However, (ii) domain-specific models are considered as instance-based models which means that they are often large models having repetitive and nested hierarchical structures and contain many objects of the same type. This conforms to our experience with existing models of automated production systems. In contrast, UML models are primarily class-based models according to [169].

Examples of *approaches based on domain-specific models* related to change propagation modelling and analysis in automated production systems are the following. Biffl et al. [34] provide an approach to support linking and versioning different engineering artefacts of an automated production system but do not support change propagation analysis. Ladiges et al. [158] observe the input and output signals of the control system of a plant and generate behavioural models which are then used to detect behavioural changes. Prähofer et al. [201] propose a feature-oriented modelling framework that supports analysing change impacts. However, neither Ladiges et al. [158] nor Prähofer et al. [201] consider the system structure (i.e. architecture) for change propagation analysis. Furthermore, none of these approaches consider technical or organisational artefacts that, besides structural artefacts of the system, can be affected by a change. In a previous work [51], we proposed an approach to model and analyse change propagation in control programs deployed on Programmable Logic Controllers (PLC). The approach in [51] is restricted to PLC software because it is an extension of the broader approach proposed in this thesis.

3.1.1.3 Change Propagation Modelling and Analysis in Business Processes and Involved Information Systems

Busch [50] identified two categories of approaches related to change propagation modelling and analysis in business processes — approaches to dynamic change propagation in business processes and approaches to change propagation in collaborative processes. Moreover, there is work on change propagation modelling and analysis between business processes and information systems discussed in [50].

Approaches to *dynamic change propagation in business processes* migrate instances of a process schema (i.e., a process metamodel [194] in our terminology) after a schema change which is denoted as "dynamic change" [207]. Kradolfer and Geppert [153] analyse and migrate process instances (i.e., process models in our terminology) after schema changes but neglect the propagation of changes in the process instances. Similar, the approaches in [203, 204] consider the change propagation from a process schema to its instances. Sadiq et al. [222] identify different types of process changes and a set of complete and minimal operations — add task, remove task, modify task properties, and modify task order. A modification methodology to support dynamic changes is proposed in [221]. The approach by Yoo et al. [284] allows defining schema modification rules but does not support the change propagation in or between instances. All these approaches are limited to change propagation analysis in processes. They neglect co-evolution of business processes and information systems (i.e., the impact of changes in the business process to information systems and vice versa). Moreover, they neglect change propagation to technical and organisational artefacts that, besides structural artefacts, can be affected by a change.

Changes in a single process may affect other related processes called collaborative processes [85, 86]. In the following, we discuss approaches to *change propagation in collaborative processes.* Approaches to describe and analyse change propagation in process choreographies are proposed, for example, in [85, 86, 209, 208]. In [157], an approach to analyse the change propagation in a process ecosystem (i.e., collection of interrelated processes) is described. Weidmann et al. [273] propose an approach to change propagation and synchronisation of business processes across different levels of abstraction. Weidlich et al. [272] propose an approach to change propagation to overlapping processes based on activities that correspond between processes.

All these approaches, however, neglect the co-evolution of business processes and information systems. They also neglect change propagation to technical and organisational artefacts that, besides structural artefacts, can be affected by a change.

Business processes and involved information systems mutually affect each other in various ways. This is what approaches to *change propagation between business processes and information systems* address. There are several approaches concerned with the co-design [170] of business processes and information systems. Sani et al. [223] propose guidelines for co-designing business processes and information systems. Vanderfeesten et al. [256] identify similarities between business processes and information systems. Warboys et al. [271] propose modelling the mutual dependencies between business processes and information systems to support co-evolution. Also Aerts et al. [5] describe mutual dependencies between business processes and information systems. Gasson [96] considers knowledge of different stakeholders in co-design of business processes and information systems. However, none of these approaches provide an analysis of change propagation. The graph matching approach in [133] identifies change patterns in the software architectural model resulting from changes in the business process model, but neglects the impact of changes in each of the single models. Sunkle et al. [245] propose an enterprise architecture ontology and change propagation rules based on heuristic. A similar approach based on heuristics describing change propagation rules is proposed in [38]. An approach to investigate misalignment between business processes and information systems as a result of changes is described in [36]. However, only a coarse-grained strategy based on metrics is presented in [36] without analysing change propagation. Avia et al. [19] propose actions as guidelines to handle misalignment between business processes and information systems, however, do not support change propagation analysis. These approaches either do not support change propagation analysis between business processes and information systems at all or neglect considering technical and organisational artefacts that, besides structural artefacts, can be affected in change propagation analysis.

3.1.1.4 Summary of Architecture-based Modelling and Analysing of Maintainability

The related approaches discussed before already provide attempts to change propagation modelling and analysis for certain disciplines. However, we identified limitations in the body of related approaches. These limitations are listed in the following. Further reading is given in [50].

- *Limitation 3.1.1-1, Cross-disciplinary change propagation:* The majority of existing approaches to change propagation modelling and analysis are restricted to a single discipline. We have seen examples limited to software or business processes. These approaches do not consider the impact of changes to artefact of one discipline to those of another discipline. However, this is required to adequately analyse change impacts and support co-evolution of different disciplines. Approaches restricted to source code cannot be generalised to systems other than software systems as these typically comprise other kinds of artefacts, such as electrical/electronical or mechanical artefacts. Consequently, a higher level of abstraction is required to describe systems for analysing cross-disciplinary change propagation. This is why we focus on the architectural level in our research. An architecture-based approach allows considering various kinds of artefacts and thus can be generalised to different disciplines [108].
- *Limitation 3.1.1-2, Use of domain-specific language:* Most of the existing approaches discussed are based on models. These either adhere to a general purpose modelling language or a domain-specific modelling language. Several approaches apply the general purpose modelling language UML. However, UML shows limitations like “single definition for syntax and static semantics” and “class-based” nature according to [169]. Existing UML-based approaches are focused on design models like class diagrams, sequence diagrams, or statechart diagrams but cannot represent change propagation on architectural level. Furthermore, as UML has been developed to represent software systems it cannot be applied to describe artefacts of other disciplines [250]. For these reasons, domain-specific languages for modelling and analysing change propagation are required.
- *Limitation 3.1.1-3, Support for technical and organisational artefacts:* Another limitation of most of the existing approaches is that they

neglect change propagation to technical and organisational artefacts that, besides structural artefacts, can be affected by a change. In consequence, this negatively affects estimation results due to neglected effort. Thus, besides structural artefacts considering various technical and organisational artefacts is required for proper change impact analysis.

3.1.2 Architecture-based Modelling and Analysis of Confidentiality

Achieving confidentiality in software-intensive systems is challenging [13, 267]. Nevertheless, it is important to consider confidentiality in system design in order to avoid high penalties and loss of reputation. Common confidentiality mechanisms described in literature are access control [220] and information flow control [104].

Approaches to modelling and analysing confidentiality can be distinguished into two categories — approaches based on *control flows* (described in Section 3.1.2.1) and those based on *data flows* (described in Section 3.1.2.2). Moreover, work on modelling and analysis of attack propagation (described in Section 3.1.2.3) for identifying confidentiality issues is related.

3.1.2.1 Modelling and Analysis of Control Flows

A control flow specifies the actions to be executed and the order in which these actions are executed by the system. We distinguish control flow modelling and analysis approaches into approaches that work on specific abstractions of the system and those working on source code.

Approaches working on abstractions of the system, for example, apply UML design models or architectural models to represent the system. These approaches can be applied already early in development when source code is not yet available. Consequently, confidentiality issues identified by these approaches can be addressed early and thus usually in a more efficient way compared to issues identified only in the source code [37, 180, 122, 173]. Examples of approaches to modelling and analysing confidentiality based on abstractions are proposed by Gerking et al. [97], Katkalov et al. [139], Jürjens [136], Hoisl et al. [120], Almorsy et al. [14], and Abdellatif et al. [2].

The approaches in [97, 139, 136] apply detailed system specifications to analyse information flow properties. These approaches lack traceability of data that is processed by a user as they cannot represent data processing by the behaviour of actors [228] but restrict behaviours to individual calls to the system. UMLSec [136] is able to analyse access control besides information flow. However, UMLSec is limited to control access to actions and does not consider access to data. Hoisl et al. [120] apply taint analysis using behaviour specifications for processes and actors. Examples of approaches that do not support behaviour specifications for processes and actors are [14] and [2]. These approaches do not analyse data propagation and are limited to pattern matching.

Approaches working on source code for analysing confidentiality can be distinguished into three categories — taint analyses, such as FlowDroid [17], full-fledged information flow analyses, such as JOANA [232] and IFcB [217], and verification approaches, such as KeY [6]. These approaches merely support one particular confidentiality mechanism and are barely extensible.

A data flow path represents a sequence of nodes a data item can take to reach a particular node [228]. All these approaches to control flow modelling and analysis are able to discover multiple data flow paths but restrict themselves to data flows via calls. Thus, they cannot represent complex data flow patterns. A comparison and detailed discussion of all these approaches can be found in [228] and Chapter 8 of this thesis.

3.1.2.2 Modelling and Analysis of Data Flows

We distinguish data flow modelling and analysis approaches into threat modelling approaches and approaches to data propagation analysis.

Threat modelling comprises various approaches to identify and mitigate threats [229]. Examples of approaches to threat modelling are proposed by Abi-Antoun et al. [3], Deng et al. [71], Yampolskiy et al. [283], Berger et al. [31], Sion et al. [230], and Frydman et al. [93]. All these approaches are limited to structural analysis using pattern matching. They do not derive confidentiality-relevant properties of data exchanged based on the data processing. Consequently, to be able to investigate information flow either all exchanged data need to be labelled manually, or results remain on the granularity level of simple taint analysis. Moreover, these approaches do not

allow to reason about multiple data classification levels [228] nor consider multiple data flow paths.

Approaches to *data propagation analysis* require a limited set of initial labels that are propagated through the system. Thus, in contrast to aforementioned threat modelling approaches, only few labels have to be assigned manually. Examples of approaches to data propagation analysis are proposed by Alghathbar et al. [11], Tuma et al. [253], and van den Berghe et al. [32]. Also our own approach in [227] and Chapter 7 of this thesis falls in this category. FlowUML [11] as well as the similar approach authUML [12] in a previous publication by the same authors derive data flows from UML sequence diagrams. Then, represent these data flows in a logic program and detect violations of information flow requirements and access control requirements. However, the publications [11] and [12] only give a brief description of the approaches and we could neither find a publication reporting about an evaluation nor an implementation of FlowUML or authUML. Tuma et al. [253], similar to our approach in [227] and Chapter 7 of this thesis, specify the system behaviour as a sequence of label propagation functions and initial labels on data. [253] is restricted to information flow control. van den Berghe et al. [32] specifies data flows between predefined processing operators to investigate security properties including a simple form of information flow control.

None of these approaches to data flow modelling and analysis report on systematically considering all possible data flow paths. A comparison and detailed discussion of all these approaches can be found in [228] and Chapter 8 of this thesis.

3.1.2.3 Modelling and Analysis of Attack Propagation

Another topic of this thesis with respect to confidentiality is modelling and analysis of attack propagation for identifying confidentiality issues. For modelling and analysing the propagation of attacks in a software-intensive system, besides aforementioned work on modelling and analysing control flows and data flows, also work on access control policy analysis and attack path analysis is related.

Examples of approaches to *access control policy analysis* are proposed by Fisler et al. [87], Alberti et al. [9], and Turkmen et al. [255]. The software suite Margrave [87] analyses role-based access control (RBAC) policies to

understand the effect of policy change. Alberti et al. [9] propose an automated analysis technique for administrative attribute-based RBAC policies. The approach in [255] analyse access control policies against security properties. These security properties express requirements on policies and on relations between policies. However, none of these approaches supports the analysis of policies for attack propagation on architectural level.

A comprehensive overview of existing approaches to attack modelling is given in [151]. Examples of approaches to *attack path analysis* are described in the following. Modelling attack trees is an established approach to analyse attack paths in the system [151]. The modelling language and probabilistic inference approach proposed by Sommestad et al. [233] investigates vulnerabilities of enterprise architectures by analysing the probability that attack paths can be accomplished. Other approaches, such as those by Polatidis et al. [198, 199] and Deloglos et al. [70], reuse existing vulnerability classifications to analyse attack paths. However, these approaches do not support analysing access control policies. In contrast, the approaches by Aksu et al. [7] and Yuan et al. [285] generate attack paths by applying simple access control models.

3.1.2.4 Summary of Architecture-based Modelling and Analysing of Confidentiality

The related approaches discussed before already provide modelling languages and analysis techniques for automated confidentiality analyses of software-intensive systems. However, we identified limitations in the body of related approaches. These limitations are listed in the following and discussed in further detail in [228] and [269].

- *Limitation 3.1.2-1, Exploration of multiple data flow paths:* In realistic systems, multiple data flow paths providing the same type of data to the same node commonly occur. Existing approaches investigate only particular paths data can take in a system design or are restricted to data flows via calls. However, considering all possible data flow paths is necessary to investigate confidentiality issues systematically.
- *Limitation 3.1.2-2, Coverage of multiple confidentiality mechanisms:* Aforementioned approaches that are focused on single confidentiality mechanisms (e.g., information flow or access control) show accurate analysis results for specific purposes. However, they lack flexibility as

engineers have to decide for a specific confidentiality mechanisms before starting modelling. Replacing the confidentiality mechanism by another implies remodelling large parts of the system. This may result in consistency problems. Support for various confidentiality mechanisms in modelling and analysis is necessary.

- *Limitation 3.1.2-3, Support for user-defined confidentiality analyses:* When engineers are forced to use predefined confidentiality mechanisms, modelling even simple requirements may become complex [228]. Means for specifying custom analyses and corresponding modelling concepts are needed. Therefore, a formalism supporting user-defined analyses for various confidentiality mechanisms as well as an appropriate modelling language are required.
- *Limitation 3.1.2-4, Support for architecture-based attack propagation:* Attackers may exploit vulnerabilities and access control policies to identify attack paths to propagate through the system. Existing approaches lack the analysis of attack propagation on architectural level by considering access control policies and existing vulnerabilities. However, this is required to identify confidentiality issues in early development.

3.2 Decomposition and Composition of Modelling Languages and Analysis Techniques

The discussion of the state of the art regarding the decomposition and composition of modelling languages and analysis techniques is organised in the following two categories. First, we discuss approaches to decomposition and composition of modelling languages (Section 3.2.1). Then, we discuss those to decomposition and composition of analysis techniques (Section 3.2.2).

3.2.1 Decomposition and Composition of Modelling Languages

The body of research in software language engineering has yielded relevant contributions to *reusing and composing language fragments* to create modelling languages. The majority of this work focuses on syntax-defining language

fragments (i.e., grammars or metamodels). Some more advanced approaches additionally consider the composition of analyses. For example, GEMOC Studio [58] supports composing modelling languages based on generic language components in form of metamodels with built-in interpretation and analyses. In GEMOC Studio, built-in analyses rely on the composition mechanisms of the Java programming language. The Neverlang [57] language workbench enables reusing and composing language components comprising grammars and analyses. MontiCore [216] provides means to support modular definition of languages and reuse analyses as part of reusing a language component [52]. Melange [69] is an approach to a modular and reusable development of modelling language syntaxes and interpreters by combining and subtyping existing language artefacts. Spoofax [140] supports the composition of context-free grammars that describe the syntax of modelling languages. Similar methods for syntax reuse are supported with the rule extension of LISA [178] or the grammar extension of Xtext [33]. LanGems [276] is a role-based language compositions system for modular language development. Also Leduc et al. [163] propose a modular approach for the definition and composition of modelling languages. MPS [265] is a language workbench in which analyses are tied to the abstract syntax of the language. There is no composition of analyses in MPS. However, none of these approaches provide any guidance for the decomposition and composition of a modelling language with respect to the specifics of given quality properties or disciplines.

Further, there is work focused on the *decomposition of modelling languages* in language engineering. For example, Strüber et al. [244, 243] propose clustering algorithms to decompose large modelling languages and models. Degueule et al. [68] propose the concept of language interfaces to abstract the various constituents of a given language. In contrast to these approaches, the decomposition in our research is based on language features. However, none of these approaches provide any guidance for the decomposition of a modelling language with respect to the specifics of given quality properties or disciplines.

Language engineering brought forth various *tools supporting the decomposition and composition of modelling languages*. EMF Splitter [95] decomposes monolithic metamodels based on the structural modularity concepts project, package, and unit. In contrast, the decomposition in our research is based on language features. GTSMorpher [287] is a tool for the composition of domain-specific modelling languages based on graph transformation systems building upon [76]. EMF Refactor [81] refactors design smells in modelling languages

based on model metrics. Puzzle [176] is a tool for extracting reusable language modules. None of these tools provide guidance for the decomposition and composition of a modelling language with respect to specifics of given quality properties or disciplines.

Other approaches to language composition provide concepts for structuring reuse without supporting specific composition mechanisms. These include, for example, the CORE approach [8] which enables concern-oriented and model-based software reuse. Concern-oriented language development (COLD) [59] is a purely conceptual framework that extends CORE to language engineering.

Furthermore, various approaches to *modelling language variability*, for example [156, 175, 89], apply product line techniques for developing languages. They focus on closed variability of either abstract syntaxes or abstract syntaxes combined with interpreters.

Another approach to the formalisation of composition in modelling and analysing is *multi-paradigm modelling* (MPM) [186] which proposes paradigms by explaining them as the composition of languages and workflows. Multi-paradigm tools, such as AToM^3 [162] and OsMoSys [257], couple heterogeneous formalisms for simulation. Foundations for multi-paradigm modelling for cyber-physical systems have recently been proposed [56]. While multi-paradigm modelling for cyber-physical systems aims to encompass similar challenges than our work, it is still on a conceptual level [15] and results towards its formalisation have yet to show [16].

The body of work in language engineering focuses on reusing and composing language fragments to create modelling languages or the decomposition of modelling languages. There are language engineering tool to support the decomposition and composition of modelling languages. Conceptional attempts to language composition provide foundations for structuring reuse without supporting specific composition mechanisms. However, we identified the following limitation.

Limitation 3.2-1, Guidance for the decomposition and composition of modelling languages: None of the work discussed before provides any guidance for the decomposition and composition of modelling languages with respect to specifics of given quality properties or disciplines.

Further discussion on the decomposition and composition of modelling languages is given in [241].

3.2.2 Decomposition and Composition of Analysis Techniques

While aforementioned approaches address the white-box form of composition, in the following, we discuss work on integration and orchestration of analysis techniques which is black-box or grey-box composition depending on how the analyses interact.

Interaction by result exchange is black-box composition. For example, Dwyer et al. [78] proposes to combine analysis tools using common representation and storage of analysis results, and means to compose these results. Cruanes et al. [61] designed the Evidential Tool Bus (ETB) to integrate diverse tools into coherent workflows. In our previous work [116], we propose a reference architecture for the integration of analysis tools into modelling environments and give examples of orchestration strategies. However, all these approaches merely focuses on storing and sharing analysis results between distributed analysis tools. They do not provide guidance for the decomposition and composition of analysis techniques with respect to the specifics of given quality properties or disciplines.

Work on *analysis coupling* [246] is grey-box composition if the single steps of the analyses are orchestrated. There is a large body of work on simulation coupling and co-simulation. Approaches like DEVS [286] and CODES [249] support the composition of discrete event-based simulations. DIS (Distributed Interactive Simulation) [124] is a suite of protocols for interconnecting simulators [134]. HLA (High Level Architecture) [125] as a successor of DIS enables the coupling of simulations. An overview of co-simulation approaches is given in [99]. All these approaches enable interaction by exchanging events during simulation. However, they do not provide guidance for the decomposition and composition of analysis techniques with respect to the specifics of given quality properties or disciplines.

The decomposition of model-based analysis techniques is not well discussed in literature to the best of our knowledge. This is because model-based analyses are often decomposed by decomposing the analysis input models and modelling languages instead of decomposing the analysis techniques. Approaches like MontiCore [216] discussed before in Section 3.2.1 are examples. These approaches reuse analyses as part of reusing a language fragment (e.g., [52]) but do not provide support for the decomposition of analysis techniques.

Existing approaches to interaction of analysis tools as well as to analysis coupling, for instance simulation coupling and co-simulation, show the following limitation.

Limitation 3.2-2, Guidance for the decomposition and composition of analysis techniques: None of the work discussed before provides any guidance for the decomposition and composition of analysis techniques with respect to specifics of given quality properties or disciplines.

3.3 Bridging the Divergent Levels of Abstraction between Development and Operation

The state of the art in bridging the divergent levels of abstraction in modelling between development and operation can be structured in eight categories: (i) work on reusing development models in operation, (ii) work on model extraction, (iii) work on user behaviour modelling and user group detection, (iv) work on consistency management in development, (v) hybrid approaches that span development and operation, (vi) instrumentation approaches to observe the running system, (vii) approaches to estimate the resource demands based on monitoring data, and (viii) work to characterise parametric dependencies.

Work on *reusing development models during operation*, such as Morin et al. [185], Ivanovic et al. [132], Schmieders and Metzger [225], and Canfora et al. [54], employs development models as foundation for reflecting software systems during operation in form of runtime models [28]. However, these approaches do not reflect component-based system architectures. Further, these approaches update the model with respect to single parameters and do not change the model structure.

Work on *model extraction* creates and updates model content based on data gathered from monitoring the system in operation. Approaches, such as those by Schmerl et al. [224], Song et al. [234], van der Aalst et al. [1], von Massow et al. [171], Brosig et al. [45], Langhammer et al. [159], PMW by Brunnert et al. [47], SLAstic by van Hoorn [123], and PMX by Walter et al. [268], establish the semantic relation between executed applications and runtime models based on monitoring data. Starting with an empty model, these approaches create model content during operation from scratch by, for

example, observing and interpreting operation traces. Therefore, they neglect information that cannot be elicited from monitoring data, such as design perspectives on component structures and component boundaries. These approaches show continuously high monitoring effort required to extract or update models. Further, there is no validation of the quality of the extracted models in these approaches.

Work on *user behaviour modelling and user group detection*, such as Langhammer et al. [159], Menascé et al. [174], Ruffo et al. [213], Vögele et al. [263, 264], Walter et al. [268], and Jung and Adolf [135], extracts user behaviour models and applies clustering algorithms to identify user groups from observed user interaction during operation. As these approaches extract user groups only based on monitoring data they neglect user groups already specified in development. Furthermore, there is work on modelling usage intensity without detecting user groups. LIMBO [146] implements an approach for modelling variations in usage intensity for seasonal patterns, trends, bursts, and noise. Herbst et al. [118] present an approach for predicting usage intensities based on decision trees and direct feedback cycles.

Work on *consistency management in development*, such as JITTAC [49], mbeddr [266], reverse engineering approaches (for example, SoMoX [27], Extract [159], ROMANTIC-RCA [80], Archimetrix [74]), and approaches concerned with architecture erosion (for example, [67]), avoids or handles inconsistencies to keep artefacts consistent in development. However, approaches in this category are limited to maintaining consistency in development but neglect inconsistencies in operation.

Work on *hybrid approaches*, such as Konersmann et al. [149, 150], EjbMox by Langhammer [160], and Spinner et al. [238], uses both, source code and monitoring data, to extract architectural models. However, these approaches apply very fine-grained monitoring which causes a high monitoring overhead. Further, these approaches show small scope of consistency preservation (e.g., limited recognition of evolution and adaption scenarios).

Instrumentation approaches, such as AIM [277] and AjaxScope [144], observe the running system to identify emerging quality issues. However, the instrumentation is rather coarse-grained and there is no way to specify which parts of the system are of specific focus and thus should be instrumented more fine-granular.

There is work on *estimating resource demands* based on either coarse-grained monitoring data, such as LibReDE [236, 237], or fine-grained monitoring data, such as [46, 278]. Approaches using fine-grained monitoring data show the advantage of high estimation accuracy for the cost of high monitoring overhead. An approach to continuously updating resource demand is proposed by Grohmann et al. [101]. However, the result of demand estimation in [101] is a constant value while parametric dependencies are not considered.

There are approaches focused on the *characterisation of parametric dependencies* in performance models, such as those by Courtois and Woodside [60], Ackermann et al. [4], and Grohmann et al. [100]. These approaches lack covering work of aforementioned categories. Krogmann et al. [155, 154] extended SoMoX with parametric dependencies based on the dynamic analysis approach Beagle. However, also SoMoX and Beagle are limited to the extraction of models.

In the body of related approaches discussed before we identified limitations. These limitations are listed in the following.

- *Limitation 3.3-1, Bridging the levels of abstraction:* As shown in the discussion, there are various approaches addressing single aspects relevant for bridging the divergent levels of abstraction in modelling between development and operation. However, there is not yet a comprehensive approach that covers all aforementioned aspects.
- *Limitation 3.3-2, Support for model validation:* Approaches discussed before do not support the validation of the quality of the extracted models. This can result in inaccuracies in the extracted models and in consequence inaccuracies in the prediction results.
- *Limitation 3.3-3, Reduction of monitoring overhead*: Approaches to coarse-grained monitoring typically show little estimation accuracy. Approaches to fine-grained monitoring show the advantage of high estimation accuracy for the cost of high monitoring overhead. The discussed approaches do not allow to adjust the granularity of monitoring and thus cannot reduce monitoring overhead.

3.4 Concluding Remark

In this chapter, we discussed the state of the art in three different areas relevant for the architecture-based evolution of dependable software-intensive systems — architecture-based modelling and analysis of maintainability and confidentiality, decomposition and composition of modelling languages and analysis techniques, and bridging the divergent levels of abstraction in modelling between development and operation. In each of these areas we identified limitations. In order to address these limitations we introduce contributions in the following chapter. In the description of these contributions we refer to the tackled limitations.

4 Approaches to Architecture-based Evolution of Dependable Software-intensive Systems

This chapter starts with an overview of the published contributions that are part of this cumulative thesis in Section 4.1. Then, the individual contributions, their relations, and how they address limitations in the state of the art are described in Section 4.2.

4.1 Publication Overview

The following publications are part of this cumulative thesis. They are clustered according to the objectives of the thesis. A detailed discussion of the contributions is given in the following section.

- Objective 1, Modelling and Analysing Maintainability and Confidentiality on Architectural Level:
  - Robert Heinrich, Sandro Koch, Kiana Rostami, Suhyun Cha, Ralf Reussner, Birgit Vogel-Heuser. Architecture-based Change Impact Analysis in Cross-disciplinary Automated Production Systems. *Journal of Systems & Software*, 146:167–185, Elsevier, 2018.
  - Kiana Rostami, Robert Heinrich, Axel Busch, Ralf Reussner. Architecture-based Change Impact Analysis in Information Systems and Business Processes. *IEEE International Conference on Software Architecture*, pages 179–188, IEEE, 2017.

  - Stephan Seifermann, Robert Heinrich, Ralf Reussner. Data-driven Software Architecture for Analyzing Confidentiality. *IEEE International Conference on Software Architecture*, pages 1–10, IEEE, 2019.
  - Stephan Seifermann, Robert Heinrich, Dominik Werle, Ralf Reussner. Detecting Violations of Access Control and Information Flow Policies in Data Flow Diagrams. *Journal of Systems & Software*, 184, Elsevier, 2022.
  - Maximilian Walter, Robert Heinrich, Ralf Reussner. Architectural Attack Propagation Analysis for Identifying Confidentiality Issues. *IEEE International Conference on Software Architecture*, pages 1–12, IEEE, 2022.
- Objective 2, Decomposition and Composition of Modelling Languages and Analysis Techniques:
  - Robert Heinrich, Misha Strittmatter, Ralf Reussner. A Layered Reference Architecture for Metamodels to Tailor Quality Modeling and Analysis. *IEEE Transactions on Software Engineering*, 47(4):775–800, IEEE, 2019.
  - Robert Heinrich, Philipp Merkle, Jörg Henss, Barbara Paech. Integrating Business Process Simulation and Information System Simulation for Performance Prediction. *Software & Systems Modeling*, 16:257–277, Springer, 2017.
- Objective 3, Bridging the Divergent Levels of Abstraction between Development and Operation:
  - Robert Heinrich. Architectural Run-time Models for Performance and Privacy Analysis in Dynamic Cloud Applications. *ACM SIGMETRICS Performance Evaluation Review*, 43(4):13–22, ACM, 2016.
  - Robert Heinrich. Architectural Runtime Models for Integrating Runtime Observations and Component-based Models. *Journal of Systems & Software*, 169, Elsevier, 2020.
  - David Monschein, Manar Mazkatli, Robert Heinrich, Anne Koziolek. Enabling Consistency between Software Artefacts for

Software Adaption and Evolution. *IEEE International Conference on Software Architecture*, pages 1–12, IEEE, 2021.

4.2 Discussion of Contributions

This section introduces my contributions proposed in this thesis, places them in context, refers to the limitations of Chapter 3 tackled by the contributions, and points out the origin of the contributions.

4.2.1 Architecture-based Modelling and Analysis of Maintainability and Confidentiality

In order to reason about maintainability and confidentiality of software-intensive systems early in development, we investigate concepts for modelling and analysing these quality properties in the architectural design of software-intensive systems.

4.2.1.1 Architecture-based Modelling and Analysis of Maintainability

As discussed in Chapter 1, maintainability is seen as an important property of dependability and often is investigated by analysing change propagation in the system. Prediction of maintainability is beneficial in early development as well as in evolution of a software-intensive system to reason about design alternatives and thus identify an appropriate system design. Predicting the maintainability of a software-intensive system in terms of change propagation is a challenging task especially in case the system comprises artefacts from multiple disciplines and is involved in complex organisational processes.

Exemplary for such a cross-disciplinary software-intensive system we consider an automated production system in this thesis. An automated production system is a special class of mechatronic systems [219, 41] that produces products [262]. It is common for an automated production system to be in operation for several decades while continuously facing changes over time [262]. Examples of these changes include replacement of mechanical or electrical/electronical parts due to physical abrasion, platform changes due to

environmental conditions, and software changes due to emerging requirements [112]. While maintaining merely software systems is already a difficult task [164], maintaining an automated production system is even more challenging. This is because an automated production system comprises various artefacts, typically from the disciplines mechanics, electrics/electronics, and control software, that mutually affect each other. These mutual dependencies between heterogeneous artefacts are the reason why even small changes can cause extensive side effects and evoke additional modifications in the respective disciplines leading to a large amount of artefacts being affected by the change [260]. As a result, predicting all artefacts affected by a change in an automated production system is time consuming and often nearly as complicated as implementing the change.

There are few attempts to change propagation analysis in automated production systems (discussed in Section 3.1.1.2). This is in contrast to several approaches existing to change propagation analysis in software systems. Existing approaches to software systems apply the software architectural model as a foundation for analysing the propagation of changes in the system. When being able to represent the various artefacts from multiple disciplines of an automated production system in an architectural model this can serve as a foundation for change propagation analysis to predict maintainability.

The publication in [112] proposes an approach to architecture-based change propagation analysis in automated production systems. Contributions described in [112] and in Chapter 5 of this thesis are as follows:

- The specification of modelling languages for: (a) modelling the detailed structure of an automated production system as a foundation for architecture-based change propagation analysis. In contrast to related work described in Section 3.1.1, this modelling language comprises the structure of electrical/electronical, mechanical, and software parts of the system in order to address Limitation 3.1.1-1 (cross-disciplinary change propagation) and Limitation 3.1.1-2 (use of domain-specific language); (b) modelling elements that represent maintenance-relevant artefacts, such as test cases, documentation but also stock and staff specifications, as annotations to structural model elements in order to address Limitation 3.1.1-3 (support for technical and organisational artefacts). These annotations are not supported by most of the related approaches discussed in Section 3.1.1; (c) representing the initial changes (denoted as seed modifications) and

steps in the propagation of changes based on the models of the system structure in order to address Limitation 3.1.1-1 (cross-disciplinary change propagation) and Limitation 3.1.1-2 (use of domain-specific language).

- The specification of algorithms and rules for change propagation analysis based on the aforementioned modelling languages in order to address Limitation 3.1.1-1 (cross-disciplinary change propagation). In contrast to related work described in Section 3.1.1, these algorithms and rules not only consider maintenance tasks for modifying structural model elements but also maintenance tasks for modifying model elements that represent technical and organisational artefacts other than those of the system structure. In consequence, the approach described in [112] and Chapter 5 results in more comprehensive task lists compared to related approaches. These comprehensive task lists are a good basis for maintenance effort estimation by domain experts.

These modelling languages as well as algorithms and rules for change propagation analysis in automated production systems I developed together with Kiana Busch while supervising her doctoral thesis [50].

Another kind of software-intensive systems we consider in this thesis is information systems involved in business processes and organisational environments. According to the established definition of the Workflow Management Coalition, a business process is a "set of one or more linked activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships" [281]. In modern organisations, software services are an integral part of their business processes. The activities of a business process are often implemented by services of software systems [115] in this context from the domain of information systems [189]. In the following, we focus on the interaction between information systems and business processes.

There are mutual dependencies between information systems and the business processes these systems are involved in [211]. On the one hand, an information system may need to be adapted to reflect changes caused in a corresponding business process. For example, supporting new activities in a business process can cause changes in the involved information system. On the other hand, a business process may need to be modified due to changes

in the involved information system. For example, replacing software components by components of alternative vendors may cause some services and in consequence activities of the business process to be changed. Again, the mutual dependencies between heterogeneous artefacts make change propagation analysis difficult as even small changes can cause extensive side effects leading to a large amount of affected artefacts.

Business process modelling is commonly used to better understand, structure, and analyse the business processes of an organisation [21]. Existing approaches to change propagation analysis in software systems (discussed in Chapter 3) are based on the architecture of the system without taking the mutual dependencies to the business process design into account.

The publication in [211] proposes an approach to architecture-based change propagation analysis in information systems and business processes building upon [210, 239] and [115] that allows to consider the mutual dependencies between information systems and business processes. Thus, the approach presented in [211] extends a previous approach [210, 239] to architecture-based change propagation analysis in software systems by more fine-grained change propagation analysis for information systems and the change propagation analysis for business processes as well as between information systems and business processes. Contributions described in [211] and in Chapter 6 of this thesis are as follows:

- The specification of modelling languages for: (a) modelling the detailed structure of information systems and business processes in order to address Limitation 3.1.1-1 (cross-disciplinary change propagation) and Limitation 3.1.1-2 (use of domain-specific language). In contrast to related work described in Section 3.1.1, this modelling language allows for explicitly representing the mutual dependencies between information systems and business processes. These dependencies are change propagators which then can be considered in change propagation analysis; (b) modelling maintenance-relevant annotations to structural model elements of the information system architecture and the business process design in order to address Limitation 3.1.1-3 (support for technical and organisational artefacts). Examples of these annotations to the information system architecture are test cases and deployment descriptors [210]. Example of these annotation to the business process design are training courses and organisational units [211, 50]; (c) representing seed modifications and

steps in the propagation of changes based on the models of the system architecture and the business process design in order to address Limitation 3.1.1-1 (cross-disciplinary change propagation) and Limitation 3.1.1-2 (use of domain-specific language).

- The specification of algorithms and rules for change propagation analysis in information systems and business processes based on the aforementioned modelling languages in order to address Limitation 3.1.1-1 (cross-disciplinary change propagation). In contrast to related work described in Section 3.1.1, these algorithms and rules not only consider maintenance tasks for modifying structural model elements but also maintenance tasks for modifying model elements that represent technical and organisational artefacts other than those of the structure of information systems and business processes. Consequently, the approach described in [211] and Chapter 6 results in more comprehensive task lists compared to related approaches. Again, these comprehensive task lists serves as a basis for maintenance effort estimation by domain experts.

These modelling languages based on the language I proposed in [115] as well as the algorithms and rules for change propagation analysis for business processes and between information systems and business processes I developed together with Kiana Busch while supervising her doctoral thesis [50]. Note, the modelling languages, algorithms, and rules for information systems are extensions of a previous approach to architecture-based change propagation analysis in software systems originally described in [239].

4.2.1.2 Architecture-based Modelling and Analysis of Confidentiality

Another important quality property of current and future software-intensive systems is confidentiality. As discussed in Chapter 1, confidentiality is seen as a property of dependability. Confidentiality plays a special role compared to other quality properties subsumed by the term dependability, such as performance or maintainability. While issues of other quality properties, for example, may degrade user satisfaction or lead to high maintenance costs, confidentiality issues can have legal consequences due to strong data protection regulations, such as the General Data Protection Regulation (GDPR) [82] of the European Union. For example, British Airways is facing a penalty of £20m [72] and Marriott International is facing a penalty of £18.4m [73] because of

confidentiality breaches. Loss of reputation after information disclosure is another threat for organisations we often find in press. A prominent example is Facebook. Users lost trust [274] after the Cambridge Analytica scandal. This loss of trust even affected Facebook's market value [127].

Confidentiality of software-intensive systems typically refers to data and data processing which is handled by the software parts of a software-intensive system. Software engineers need to consider confidentiality in every phase of development and operation to ensure compliance right from the beginning on. This is because confidentiality can barely be incorporated as a polishing step once the system has been designed. In consequence, prediction of confidentiality needs to be applied in early development as well as in evolution of a software-intensive system to reason about design alternatives and thus identify an appropriate system design.

Considering confidentiality early is crucial to avoid high effort for fixing confidentiality issues. Studies on software engineering economics in general, for example [37], show the earlier issues are identified the more cost-efficient they can be fixed. The same holds true for security issues [180, 122, 173].

Identifying and fixing confidentiality issues already in the architectural design phase seems reasonable as in this phase sufficient knowledge about the system is available and significant decision decisions are made, such as about the system structure and distribution, about which types of data to be processed, and by which data processing operators, yet the design can be modified in a cost-efficient manner compared to later phases, such as the implementation phase, where related approaches discussed in Section 3.1.2 are located. Ensuring proper architectural design does not free software engineers from considering confidentiality in the following phases but forms a solid foundation for subsequent phases by identifying and fixing fundamental issues that can barely be addressed later even when spending considerable effort.

Model-based confidentiality analyses have demonstrated in a case study [136] to be appropriate for identifying confidentiality issues. As manually inspecting the system architecture is complex and labour-intensive, automated confidentiality analysis based on appropriate models can speed up the detection of confidentiality issues [254]. Confidentiality analyses operating on data flow diagrams are promising as confidentiality issues tend to follow the data flow [229].

The publication in [227] introduces data flows in an architecture description language to describe data and data processing and to enable the investigation of confidentiality issues in the architectural design phase. Contributions described in [227] and in Chapter 7 of this thesis are as follows.

- The specification of a modelling language for the representation of data and predefined data processing operations as first class entities in architectural models. A sequence of these data processing operations that exchange data describes the system behaviour to be analysed for confidentiality issues.
- The specification of a confidentiality analysis technique using the confidentiality mechanism access control to detect confidentiality issues by comparing access rights assigned to data with roles assigned to processing operations on this data. The analysis technique considers that data processing can change data properties, for example, by declassifying data using an aggregation operation that removes confidential details.

This modelling language and analysis technique I developed together with Stephan Seifermann while supervising his doctoral thesis [226].

The approach we proposed in [227], just like related approaches discussed in Section 3.1.2, faces limitations that need to be addressed to adequately detect confidentiality issues in the architectural design phase. As discussed in Section 3.1.2, these limitations refer to the systematic exploration of multiple data flow paths, the coverage of multiple confidentiality mechanisms, and the support for user-defined confidentiality analyses.

In order to address these limitations, the publication in [228] proposes an extended data flow diagram syntax that supports modelling both, information flow and access control, in the same language. Moreover, the publication in [228] proposes analysis semantics that support various types of confidentiality analyses. Contributions described in [228] and in Chapter 8 of this thesis are as follows.

- The specification of an extended data flow diagram syntax by a metamodel that provides syntactical extensions for representing confidentiality mechanisms. The publication introduces the concept of alternative data flows via pins to represent multiple data sources and destinations to address Limitation 3.1.2-1 (exploration of multiple data flow paths). The extended data flow diagram syntax allows to

distinguish between system parts that depend on particular confidentiality mechanisms and those that do not. All system parts related to specific confidentiality mechanisms are encapsulated in user-defined model extensions to address Limitation 3.1.2-3 (support for user-defined confidentiality analyses). Such a model extension consists of confidentiality properties and behaviour descriptions that specify how the system changes these properties during its execution. The metamodel can represent information flow and access control by the extensions to address Limitation 3.1.2-2 (coverage of multiple confidentiality mechanisms).

- The specification of data flow diagram semantics for confidentiality analyses based on label propagation that support various types of confidentiality analyses. Confidentiality properties are mapped to labels. Behaviour descriptions are mapped to label propagation functions. An analysis then compares labels resulting from the label propagation with expected labels coming from requirements. The semantics consider all data flow paths to address Limitation 3.1.2-1 (exploration of multiple data flow paths). This approach covers information flow analysis and access control analysis to address Limitation 3.1.2-2 (coverage of multiple confidentiality mechanisms) as well as user-defined analyses to address Limitation 3.1.2-3 (support for user-defined confidentiality analyses).

This extended data flow diagram syntax and semantics I developed together with Stephan Seifermann while supervising his doctoral thesis [226].

Besides analysing data flows, confidentiality issues can be detected by considering system vulnerabilities and potential attack paths. Single vulnerabilities, such as sensitive information stored in cleartext [63, 62], viewed in isolation are seldom critical since they often require certain privileges to exploit or are hidden within internal networks without access from the outside. However, attackers may build attack paths based on the combination of several vulnerabilities to breach the system. To be more specific, attackers may exploit single vulnerabilities to gain more and more credentials and further propagate through the system until they are able to access critical data or resources. By doing so, attackers may exploit existing access control policies to further infiltrate the system. Consequently, investigating the dependencies between access control policies and vulnerabilities is important already in early development to identify confidentiality issues. Analyses to reveal these

confidentiality issues require information about system structure and behaviour, data, and deployment. Therefore, the software architecture seem to be a reasonable foundation to identifying attack paths based on vulnerabilities and access control policies.

Existing approaches to identifying attack paths based on vulnerabilities and access control policies, for example Bloodhound [35] or approaches from the Darpa Cyber Grand Challenge (CGC) [66], require deployed systems. Thus, they cannot be applied in early development. Related approaches discussed in Section 3.1.2 lack attack propagation on architectural level by considering detailed access control policies and existing vulnerabilities.

The publication in [269] proposes an architecture-based approach for analysing the dependency between vulnerabilities and access control policies to identify attack paths and confidentiality issues in order to address Limitation 3.1.2-4 (support for architecture-based attack propagation). Contributions described in [269] and in Chapter 9 of this thesis are as follows:

- The specification of a modelling language that allows modelling vulnerabilities and access control policies. In contrast to existing approaches discussed in Section 3.1.2, the approach in [269] uses a fine-grained access control model together with a vulnerability model based on commonly used attack classifications.
- The specification of an attack propagation analysis technique to investigate how a malicious user propagates through the system by using stolen credentials and exploiting existing vulnerabilities. The attack propagation analysis technique builds upon the previously described approaches to change propagation analysis and a generic methodology [108]. The analysis technique considers architectural information, such as system structure, behaviour, and deployment, as well as considers fine-grained access control policies together with vulnerabilities.

This modelling language and analysis technique for attack propagation analysis I developed together with Maximilian Walter while supervising his doctoral thesis.

4.2.2 Decomposition and Composition of Modelling Languages and Analysis Techniques

So far we have discussed several modelling languages and model-based analysis techniques for reasoning about quality properties of software-intensive systems. Model-based analyses support the initial development as well as the evolution of software-intensive systems by identifying quality issues and allowing to compare design alternatives based on their expected quality properties. However, not only the systems face evolutionary changes but also the modelling languages and analysis techniques have to evolve in order to satisfy new or changing requirements.

Today, specific modelling languages and analysis techniques are used for model-based analysis of software-intensive systems. They enforce individual models for each desired form of analysis. Approaches to unification, such as extending existing well-proven modelling languages and analysis techniques, showed these become unmanageable due to growth in complexity and number of features, for example new quality properties or disciplines to be analysed, being added over time.

For example, the Palladio approach [206] was initially designed for performance analysis based on software architectures. Over time, Palladio has been extended for modelling and analysing reliability, scalability and elasticity, costs, maintainability, energy consumption, and many other quality properties [117]. These modifications led to serious degradation of the internal structure of the modelling language [242, 241] and the tools implementing the analysis techniques [113]. Feature overload, feature scattering, and unconstrained creation of dependencies, for example, harm the evolvability and reusability of the modelling language and analysis techniques. This is because Palladio, like similar approaches, relies on a monolithic modelling language and monolithic analysis techniques, making modifications, extensions, and partial reuse challenging.

Guidance for the flexible creation, modification, extension, and partial reuse of modelling languages and analysis techniques with respect to specifics of given quality properties or disciplines is currently not given as discussion in Section 3.2. Feature-based decomposition would give the flexibility to compose those (partial) modelling languages and analysis techniques needed in a purpose-specific way. This would reduce situations that substantial parts of the modelling languages and analysis techniques for reasoning about

structure, behaviour, and quality properties of a software-intensive system show similar features, but are realised completely independent over and over again.

In our research, we focus on metamodels for specifying the abstract syntax of a modelling language. This is because of the close relationship between metamodel design and object-oriented software design [109, 241] as described in Section 1.2 which makes it reasonable to applying concepts from object-oriented software design to metamodel design.

A plethora of different metamodels for modelling software-intensive systems and their quality properties is available in literature. Existing metamodels are specific to different quality properties (e.g., performance versus reliability), tools (e.g., the Palladio bench [206] versus QPN-Tool [23]), or analysis tasks (e.g., mean time analysis versus prediction of a statistical distribution) [109]. When comparing metamodels developed for modelling and analysing different quality properties, substantial parts of these models for the specification of structure and behaviour of the system show similar language features. However, existing metamodels are seldom designed in a way that they are extensible and reusable in different contexts. A systematic way of creating, extending, and reusing (partial) metamodels for quality modelling and analysis while preserving their internal structure is required to reduce complexity.

The publication in [109] investigates the applicability of decomposition and composition concepts known from object-oriented software design and the idea of a reference architecture known from software engineering to metamodels for quality modelling and analysis in order to systematically create, extend, and reuse (partial) metamodels. This approach addresses Limitation 3.2-1 (guidance for the decomposition and composition of modelling languages) and allows to tailor metamodels to specific modelling purposes. Requirements on the reference architecture are gathered from the historically-grown metamodel of the Palladio approach. Decomposition and composition concepts are specified as a foundation of the reference architecture. Detailed application guidelines are described. The reference architecture supports instance compatibility and non-intrusive, independent extension of metamodels. Contributions described in [109] and in Chapter 10 of this thesis are as follows.

- The specification of fundamental concepts for decomposing metamodels that enable the description of language features, language components (metamodel modules in [109]), their relations, and their

grouping. In contrast to related approaches described in Section 3.2, the decomposition concepts enable clear distinction between language features and their implementation in language components.

- The investigation of composition concepts in form of extension mechanisms for metamodels to compose (partial) metamodels in a purpose-specific way.
- The definition of the first reference architecture for metamodels for quality modelling and analysis based on the decomposition and composition concepts for metamodels. In contrast to related approaches discussed Section 3.2, the reference architecture provides guidance and a systematic way of creating, extending, and reusing metamodels for different quality properties and disciplines. Furthermore, design rationale behind the reference architecture are discussed and detailed guidelines on the application of the reference architecture are given for two scenarios: (1) designing a metamodel from scratch and (2) refactoring an existing metamodel.

These decomposition and composition concepts and the reference architecture for metamodels I developed together with Misha Strittmatter while supervising his doctoral thesis [241].

Furthermore, there exists a plethora of different analysis techniques relying on different forms of the same or similar modelling languages. The Palladio approach [206], for example, comprises at least eight different analysis techniques for reasoning about software architectural design. They all rely on different variants of Palladio's metamodel. Again, those analysis techniques are specific to certain quality properties, tools or analysis tasks.

When comparing different analysis techniques for reasoning about quality properties, substantial parts of the analysis techniques show similar analysis features. For example, there are three different performance simulators for Palladio's metamodel — SimuCom [25], SimuLizar [24], and EventSim [177] — that show similar or identical analysis features. However, existing analysis techniques are seldom designed in a way that parts can be reused in different contexts. Like for modelling languages, a systematic way of creating, extending, and reusing (partial) analysis techniques is required to reduce complexity of existing analysis techniques.

Decomposition and purpose-specific composition is key for flexible use of dedicated analysis techniques, extension, and reuse of (partial) analysis tech-

niques for different variants of modelling languages. Our hypothesis is that the decomposition of a model-based analysis technique can follow the decomposition of the modelling language it is based on. The decomposition of analysis techniques can be achieved by transferring the decomposition concepts and reference architecture for metamodels proposed in [109] to analysis techniques. Analysis techniques may be decomposed into analysis components along the features they provide. The individual analysis components may be composed to satisfy a specific analysis purpose. A conceptual overview and introduction of decomposition concepts for analysis techniques (analysis feature, analysis component, and their dependencies) as well as the relation between concepts of modelling languages and those of analysis techniques has already been given in Figure 2.1. In [148], we proposed first attempts to feature-based investigation of partial simulations for reuse. The decomposition of analysis techniques, however, is topic of future work as described in Chapter 15 of this thesis. In the following, we focus on the composition of analysis techniques.

The publication in [115] investigates the composition of analysis techniques with a special focus on performance simulation of information systems and business processes. In contrast to existing approaches discussed in Section 3.2, the approach in [115] considers the specifics of given disciplines — information systems and business processes — and a quality property — performance. Thus, it represents a concrete example and guidance for the composition of analysis techniques for these specific disciplines and quality property. Consequently, the approach in [115] addresses the composition part of Limitation 3.2-2 (guidance for the decomposition and composition of analysis techniques) specific for the disciplines information systems and business processes and the quality property performance. Contributions described in [115] and in Chapter 11 of this thesis are as follows.

- The provision of an overview of composition concepts in form of concrete composition operators for performance simulators with a special focus on analysing the mutual impact between information systems and business processes. These composition operators are: composition by result exchange between isolated simulators, composition by co-simulation, composition by transformation into a joint formalism, and composition by extension of one simulator by another. Further description of these composition operators is given below.

- The specification of an extension of Palladio's software architecture description language by modelling concepts for representing business processes and their organisational environment in order to enable an integrated simulation. Business processes are represented in terms of sequences of actor steps and system steps, and workload specifications. The organisational environment of a business process is represented in terms of resources involved in the business process. Resources of the organisational environment encompass human actors and their roles as well as their equipment.
- The specification of an integrated performance simulator for information systems and business processes by extending Palladio's simulator EventSim [177] in order to contribute to: (a) the alignment of business process design and information system architecture while considering the mutual impact in between in simulation; (b) a more accurate performance prediction compared to existing isolated simulators in cases of high workload burstiness. Workload burstiness has "paramount importance for queuing prediction" [179] because it reflects whether load is dispersed equally or in bursts [105]. Consequently, workload burstiness is an important criteria for composition of performance simulators; (c) a more accurate performance prediction compared to existing simulators due to the representation of human actor behaviour in simulation in form of a scheduling policy specific for human actors based on dedicated modelling constructs.

The extension of Palladio's architecture description language by concepts to represent business processes and their organisational environment as well as the integrated performance simulation including scheduling policies for human actors, and the discussion of composition operators are my contributions. [115] represents a connecting link between the research of my doctoral thesis [105] and my following research. Progress of [115] in comparison to my doctoral thesis [105] includes a comprehensive discussion of the simulation of human actor behaviour while reflecting it against open issues reported in literature, an elaborated discussion of composition operators for performance simulators, and a comprehensive scalability study of our integrated simulator comprising three different experiments.

In general, three forms of composition exist in the context of model-based analysis — model composition (white-box composition), result composition

(black-box composition), and analysis composition (grey-box composition) — as we have identified in [246]. In [113][1], we give examples of how these forms of composition are implemented in existing simulators by discussing specific composition operators for simulators in the context of Palladio building upon [115].

Composition by result exchange between isolated simulators [115] conforms to the form result composition (black-box composition) [246]. It is the most simple way of simulator composition. This composition operator can only be applied if one simulator requires the results of another simulator, but there is no interaction between the simulators required during simulation. Both simulators are executed in isolation, and information is exchanged ex-post by inserting the results of one simulator as input into another simulator. Composition by result exchange shows limitations. For example, when exchanging results between isolated queue-based simulators, workload burstiness cannot be considered adequately leading to inaccurate simulation results [115]. The forms discussed in the following are able to adequately represent workload burstiness by applying more elaborated composition operators.

Composition by co-simulation [115] conforms to the form analysis composition (grey-box composition) [246]. This composition operator enables information exchange during simulation. Simulators are interlinked in order to exchange information during simulation. Co-simulation commonly requires additional efforts, for example, a coordinator for time management, model synchronisation, and connectivity in order to enable coherent simulation.

Composition by transformation into a joint formalism [115] conforms to the form model composition (white-box composition) [246]. This composition operator uses model transformations for creating a homogeneous simulation model. A characteristic of this way of simulator composition is that a single formalism model is used as input to the simulation. Commonly, general-purpose simulation formalisms like Petri nets or queuing networks are used as the target formalism. This way of simulator composition can only be applied if there is a joint formalism to integrate the models of all the simulators, or if such an integrated formalism can be constructed.

[1] The book chapter [113] has not been included in this habilitation thesis because it is part of an anthology not suitable for an habilitation thesis as there was no anonymous peer-review process because of my editorship for this book. However, the following description of composition operators is taken from [113].

Composition by extension [115] is another way to implement the form model composition (white-box composition) [246]. This way of simulator composition is about extending the modelling language and simulation routines of one simulator by the modelling language and simulation routines of another simulator to form an integrated and unified simulator. Composition by extension is applicable if all the simulators build upon the same (or compatible) modelling paradigm and simulation formalism.

4.2.3 Bridging the Divergent Levels of Abstraction between Development and Operation

The previously described contributions were focused on the development and evolution of software-intensive systems as well as the corresponding modelling languages and analysis techniques. The operation of software-intensive systems is addressed by the following contributions.

Modern software-intensive systems operate based on distributed, decentralised, and cloud-based computing resources while facing various events as described in Section 1.1. Handling these events requires strong interaction of evolution activities on development level and adaptation activities on operation level. The software architecture is a central artefact to keep track of software-intensive systems for both, developers while evolving the system and operators while adapting the system. Existing architectural models used in the development phase, however, differ from those used in the operation phase in terms of purpose, content, and, in particular, abstraction [110]. These differences result in limited reuse of development models during operation, lost architectural knowledge, and limited phase-spanning consideration of the software architecture.

The publication in [106] proposes foundations for bridging the divergent levels of abstraction between models used in development and those used in operation in order to address Limitation 3.3-1 (bridging the levels of abstraction) by considering operation-level adaptation and development-level evolution as two mutual interwoven processes. Central to this perception is the notion of an architectural runtime model. Contributions described in [106] and in Chapter 12 of this thesis are as follows.

- The introduction of the notion of an architectural runtime model which reflects updates of component structures, deployments, and

application usage caused by changes in the software application and its environment during operation. In contrast to related work discussed in Section 3.3, the architectural runtime model targets to be usable for automated adaptation and is simultaneously comprehensible for humans during evolution.

- The introduction of first attempts to modelling concepts to align architectural models used in development and those used in operation. As an umbrella a megamodel integrates development models, code generation, monitoring, and runtime model update.

The content presented in this single-author publication is my contribution.

The publication in [107] details and extends the modelling concepts introduced in [106] to align architectural models used in development and those used in operation and thus addresses Limitation 3.3-1 (bridging the levels of abstraction). Contributions described in [107] and Chapter 13 of this thesis are as follows.

- The specification of a correspondence model to bridge the divergent levels of abstraction between elements of the component-based architectural model and implementation artefacts. The publication refines the concepts introduced in [106] by providing a detailed description of the Runtime Architecture Correspondence Model (RAC) to specify the correspondence between the architectural level and the implementation level. Consequently, the RAC bridges the divergent levels of abstraction between component-based architectural models created in development and the artefacts implementing services to be observed in operation. While generating source code from the architectural model, correspondence information is recorded in the RAC and is subsequently used for updating the architectural model based on observations of the executed system.
- The development of a transformation pipeline that uses the information stored in the correspondence model to update architectural models based on changes observed in operation. The publication extends the transformation pipeline introduced in [106] by transformations to update the architectural runtime model for changes in component deployment and allocation of execution containers.

- The specification of an approach to model complex workload, such as several user groups of different behaviour and nested user behaviour, based on observations in operation for architecture-based quality prediction. In contrast to related approaches discussed in Section 3.3, this approach exploits knowledge from the existing architectural model to identify different user groups, their user behaviour, and their usage intensity. It is fully integrated in the modelling environment by exploiting the information specified in the RAC to identify user interaction and drive model updates using the transformation pipeline.

The content presented in this single-author publication is my contribution.

Aforementioned events that occur in modern software-intensive systems may not only lead to the system drifting away from its development models due to adaptations during operation but also due to evolutionary changes to the source code that are not reflected in the architectural models. Consequently, for keeping track of modern software-intensive systems it is required to keep all the representations of the system (i.e. architectural model, source code, monitoring data) consistent.

The quality of a model-based analysis heavily relies on the accuracy of the models used. A general disadvantage of using models is that their accuracy is uncertain and thus also the prediction results are uncertain. Consequently, approaches to identify inaccuracies in models and strategies to handle them are required.

Furthermore, while monitoring is essential to observe the running system in [107], a key disadvantage of monitoring is that it introduces overhead which can negatively affect the performance of the observed system. Consequently, approaches to minimise monitoring overhead while at the same time allow for sufficient insights into the running system are required.

The publication in [184] presents an approach to keep various representations of the system consistent throughout evolution and adaption building upon [172] and [107]. Moreover, the approach allows for self-validation and reduces monitoring overhead while observing the running system. Contributions described in [184] and in Chapter 14 of this thesis are as follows.

- The specification of an in-depth automated consistency preservation strategy between the system design (architectural model and source code) and adaptive as well as evolutionary changes based on the

consistency preservation approach Vitruvius [147]. Thus, the publication further contributes to address Limitation 3.3-1 (bridging the levels of abstraction). By covering both, development and operation, an up-to-date architectural model is available at any point in time and can be used for quality analysis (performance prediction in [184]). In contrast to related approaches discussed in Section 3.3, the system composition in form of components and their interfaces is represented in the architectural model. The system composition is analysed in the development phase based on the source code and in the operation phase based on monitoring data. By introducing a new graph-based data structure, the system composition can be updated in an automated way. Consequently, the modelling effort can be reduced compared to existing non-automated approaches and architectural design decisions can be evaluated.

- The development of a self-validation concept as an extension of the existing CIPM approach [172] to address Limitation 3.3-2 (support for model validation). During operation, simulations are performed based on the architectural model and the simulation results are compared to measurements to reason about accuracy of the model-based performance prediction. Thus, the approach in [184] can reveal inaccuracies in the model and react to them. This results in more dynamic maintenance of consistency relationships informed by previous shortcomings.
- The extension of the transformation pipeline proposed in [107] that uses monitoring data as input to update the architectural model. Aforementioned validation is also used to adjust the granularity of the monitoring (i.e. the level of detail of the monitoring) to reduce the monitoring overhead in order to address Limitation 3.3-3 (reduction of monitoring overhead). Thus, only parts of the system can be observed that are poorly represented in the model. For example, in case of large deviations between simulation results and observations, the monitoring granularity can be increased to create a more accurate model. The monitoring granularity can be reduced if the model is accurate enough.

To this publication I contributed extended concepts for updating architectural models by monitoring data collected in operation as part of the consistency preservation strategy, the extension of the transformation pipeline, and the

monitoring adjustment according to the validation results building upon [107]. Furthermore, I was involved in the conception of the derivation of the architectural model based on source code and the self-validation concepts as an extension of [172]. All this I developed together with David Monschein while supervising his master thesis [183].

5 Architecture-based Change Impact Analysis in Cross-disciplinary Automated Production Systems

The paper is available at the following reference.

Robert Heinrich, Sandro Koch, Kiana Rostami, Suhyun Cha, Ralf Reussner, Birgit Vogel-Heuser. Architecture-based Change Impact Analysis in Cross-disciplinary Automated Production Systems. *Journal of Systems & Software*, 146:167–185, Elsevier, 2018. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2018.08.058.

6 Architecture-based Change Impact Analysis in Information Systems and Business Processes

The paper is available at the following reference.

Kiana Rostami, Robert Heinrich, Axel Busch, Ralf Reussner. Architecture-based Change Impact Analysis in Information Systems and Business Processes. *IEEE International Conference on Software Architecture*, pages 179–188, IEEE, 2017. DOI: https://doi.org/10.1109/ICSA.2017.17.

7 Data-driven Software Architecture for Analyzing Confidentiality

The paper is available at the following reference.

Stephan Seifermann, Robert Heinrich, Ralf Reussner. Data-driven Software Architecture for Analyzing Confidentiality. *IEEE International Conference on Software Architecture*, pages 1–10, IEEE, 2019. DOI: `https://doi.org/10.1109/ICSA.2019.00009`.

8 Detecting Violations of Access Control and Information Flow Policies in Data Flow Diagrams

The paper is available at the following reference.

Stephan Seifermann, Robert Heinrich, Dominik Werle, Ralf Reussner. Detecting Violations of Access Control and Information Flow Policies in Data Flow Diagrams. *Journal of Systems & Software*, 184, Elsevier, 2022. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2021.111138.

9 Architectural Attack Propagation Analysis for Identifying Confidentiality Issues

The paper is available at the following reference.

Maximilian Walter, Robert Heinrich, Ralf Reussner. Architectural Attack Propagation Analysis for Identifying Confidentiality Issues. *IEEE International Conference on Software Architecture*, pages 1–12, IEEE, 2022. DOI: `https://doi.org/10.1109/ICSA53651.2022.00009`.

10 A Layered Reference Architecture for Metamodels to Tailor Quality Modeling and Analysis

The paper is available at the following reference.

Robert Heinrich, Misha Strittmatter, Ralf Reussner. A Layered Reference Architecture for Metamodels to Tailor Quality Modeling and Analysis. *IEEE Transactions on Software Engineering*, 47(4):775–800, IEEE, 2019. ISSN: 0098-5589. DOI: https://doi.org/10.1109/TSE.2019.2903797

11 Integrating Business Process Simulation and Information System Simulation for Performance Prediction

The paper is available at the following reference.

Robert Heinrich, Philipp Merkle, Jörg Henss, Barbara Paech. Integrating Business Process Simulation and Information System Simulation for Performance Prediction. *Software & Systems Modeling*, 16:257–277, Springer, 2017. ISSN: 1619-1366. DOI: `https://doi.org/10.1007/s10270-015-0457-1`.

12 Architectural Run-time Models for Performance and Privacy Analysis in Dynamic Cloud Applications

The paper is available at the following reference.

Robert Heinrich. Architectural Run-time Models for Performance and Privacy Analysis in Dynamic Cloud Applications. *ACM SIGMETRICS Performance Evaluation Review*, 43(4):13–22, ACM, 2016. ISSN: 0163-5999. DOI: `https://doi.org/10.1145/2897356.2897359`.

13 Architectural Runtime Models for Integrating Runtime Observations and Component-based Models

The paper is available at the following reference.

Robert Heinrich. Architectural Runtime Models for Integrating Runtime Observations and Component-based Models. *Journal of Systems & Software*, 169, Elsevier, 2020. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2020.110722.

14 Enabling Consistency between Software Artefacts for Software Adaption and Evolution

The paper is available at the following reference.

David Monschein, Manar Mazkatli, Robert Heinrich, Anne Koziolek. Enabling Consistency between Software Artefacts for Software Adaption and Evolution. *IEEE International Conference on Software Architecture*, pages 1–12, IEEE, 2021. DOI: `https://doi.org/10.1109/ICSA51549.2021.00009`.

15 Conclusion

This chapter concludes the thesis with a summary and gives an outlook of future areas of research to extend and improve the approaches described before.

15.1 Summary

This cumulative habilitation thesis presented approaches to architecture-based evolution of dependable software-intensive systems. We focused on three important quality properties of dependable software-intensive systems — performance, maintainability, and confidentiality. These quality properties strongly depend on architectural design decisions. However, we have shown that there is little support for reasoning about quality properties, especially maintainability and confidentiality, early in development. This little support for architecture-based quality analysis may result in quality issues identified only late or even occur once the system is in operation, unfavourable design decisions that are costly to be revised later, or the need for prototypes. We proposed approaches to modelling and analysing maintainability of software-intensive systems based on architectural models in Chapter 5 and Chapter 6 in order to answer Research Question 1.1. Moreover, we proposed approaches to modelling and analysing confidentiality of software-intensive systems based on architectural models in Chapter 7, Chapter 8, and Chapter 9 in order to answer Research Question 1.2.

Besides the software-intensive system itself also the modelling languages and analysis techniques to reason about the system have to evolve to satisfy emerging or changing requirements due to novel system properties. However, we have shown that there is little flexibility for purpose-specific creation, extension, and reuse of modelling languages and analysis techniques with respect to the specifics of given quality properties or disciplines in approaches

to model-driven engineering. In order to answer Research Question 2, we investigated concepts for decomposition and composition known from object-oriented software design and transferred them to modelling languages for quality modelling and analysis of software-intensive systems in Chapter 10. Moreover, we investigated the composition of analysis techniques with a special focus on performance simulation of information systems and business processes in Chapter 11.

Once a software-intensive system is in operation it typically drifts away from its initial development models due to adaptive and evolutionary changes. Observing the system in operation is a feasible approach to keep track of the system. However, there is a gap of abstraction between the data elicited by observation in operation and the architectural models used in development. This gap results in loss of architectural knowledge and thus makes the initial development models increasingly less useful or even entirely useless for reasoning about quality properties of the system for upcoming changes. Source code changes without changing the architectural model is another reason for systems drifting away from their architectural models. We have shown that existing approaches are limited to single aspects relevant for bridging the divergent levels of abstraction in modelling between development and operation. In order to answer Research Question 3, we proposed approaches to bridge the divergent levels of abstraction between data of the operation phase, architectural models (in Chapter 12 and Chapter 13) and source code (in Chapter 14) of the development phase.

15.2 Outlook

An outlook of possible future research activities with respect to architecture-based evolution of dependable software-intensive systems is given in this section. The outlook is structured by the objectives of this thesis.

15.2.1 Architecture-based Quality Modelling and Analysis

In this thesis, we presented approaches to architecture-based modelling and analysis of software-intensive systems for the quality properties maintainability, confidentiality, and performance. An obvious topic of future work is to consider additional quality properties of the ISO/IEC 25010 quality models

[128] in our research. Reliability is a candidate for a quality property to be considered next as there is already a large body of work on modelling and analysing reliability in different disciplines available that, however, needs to be investigated for applicability on the level of the system architecture. Moreover, developing approaches to analyse trade-offs between different quality properties of software-intensive systems, such as performance versus confidentiality or maintainability versus security, and to identify optimal design decisions with respect to these trade-offs seems to be an interesting topic of future work. There is already a large body of work for software systems (e.g., [141, 10]) that needs to be investigated for applicability on the level of the system architecture. For both, the concepts proposed in this thesis serve as foundations for future research.

An overview of future work regarding architecture-based modelling and analysis of maintainability is given by Busch [50] as summarised in the following.

We have presented approaches to architecture-based change propagation analysis in automated production systems as well as in business processes and information systems. In [108], we introduced and further detailed in [50] a methodology for domain-spanning change impact analysis that generalises the approaches presented in this thesis and allows to be instantiate in various disciplines. The instantiation of this methodology in new disciplines is expected to reveal further insights on the applicability of the methodology. Instantiation to modelling and analysing systems from the automotive domain seems to be an interesting future application areas that promises new insights due to the various heterogeneous disciplines involved. Instantiating the methodology in different ways in a given discipline would provide further insights on the applicability of the methodology. The instances can be developed based on influencing factors discussed in [50] to systematically analyse the effects of the influencing factors and to develop guidelines for the creation of possible future instances of the methodology. Moreover, the effects of different influencing factors on analysis results can be investigated further.

The approaches to change propagation analysis presented in this thesis are based on the approach in [210, 239]. The approach in [210, 239] was evaluated in an empirical study that indicated an automated approach helps less-experienced users to estimate change effort more completely and precisely. The approaches presented in this thesis were evaluated using community

case studies [119, 259] from different domains as we focused on the demonstration that architecture-based change propagation analysis can be used to reason about various kinds of software-intensive systems. In the future, empirical studies can be conducted for the approaches proposed in this thesis to investigate how these approaches can help domain experts in analysing change propagation in different domains. These empirical studies can be set-up similar to the previous study in [210, 239].

Another interesting question to be answered in the future is how the automated approaches to change propagation analysis proposed in this thesis scale, for instance, for different numbers of model elements, different types of change propagation rules, or different types of dependencies between model elements. Moreover, the effort required to develop the modelling languages and analysis algorithms for instantiating the methodology at different abstraction levels as well as the effort required to create models and analyse the change propagation, both manually and using an automated approach, can be investigated to better understand when applying a model-based and automated approach to change propagation analysis pays off.

The approach presented in Chapter 6 uses fairly generic modelling languages to represent business processes and information systems. Consequently, fairly generic change propagation rules have been specified based on these languages. Using modelling languages tailored to more specific kinds of processes or systems would allow for defining more specific change propagation rules and thus promises to result in more precise analysis results. However, more specific modelling languages and rules come with the disadvantage that they may not be applicable to an arbitrary process or system and development of these languages and rules may be more costly. This needs to be further investigated — also for other disciplines — in the future.

The approaches to change propagation analysis proposed in this thesis result in a set of potentially affected model elements which is used by domain experts to estimate the effort of implementing a given change request. In the future, we may further investigate the needs of domain experts, for example, in terms of the granularity of the set of model elements for effort estimation.

In the presented approaches, domain experts have to indicate seed modifications in the model based on a change request. An interesting question to be investigated in the future is whether these seed modifications can be identified automatically based on a given representation of a change request. Ways to automated identification of seed modifications obviously depend

on the representation of the change request. For textual representation of a change request the application of approaches to natural language processing seem to be promising.

The approaches to change propagation analysis presented in this thesis apply models of the system architecture. These models may be extracted automatically by approaches like those proposed in this thesis to address Challenge 3 (cf. Chapter 12, Chapter 13, and Chapter 14) or related approaches discussed in Section 3.3 and be extended automatically with technical and organisational artefacts other than those of the system structure in the future. A first attempt to the automated extension with technical and organisational artefacts has been proposed in [212] but needs to refined in the future.

An overview of future work regarding architecture-based modelling and analysis of confidentiality is given by Seifermann [226] as summarised in the following.

The approaches presented in this thesis are focused on confidentiality. However, considering only one security objective is not enough to secure a system. A topic of future work is to investigate how the modelling languages and analysis techniques proposed in this thesis can be applied and extended to reason about other security objectives. As a next step investigating the security objective integrity seem to be promising because the mechanisms access control and information flow control discussed in this thesis can also be applied to protect the integrity of information.

For the approaches presented in this thesis we assumed the models adequately represent the system under study. However, software-intensive systems will further evolve to process data in highly dynamic contexts and will show dynamically changing system structure and behaviour in the future. The high level of heterogeneity, complexity, and dynamicity of future software-intensive systems make these systems different from traditional systems as the sheer number of possible situations that may occur during operation will result in high level of uncertainty, for example about system structure, behaviour, and contexts. Ensuring confidentiality in such a system requires significant paradigm shift with respect to existing approaches. The high level of dynamicity deprecates existing techniques to static modelling and analysis. The high level of uncertainty results in even more significant challenges as it collides with the traditional interpretation and modelling of confidentiality where, for example, access/deny decisions are sharp and fully determined. In future systems fraught with uncertainty the rigid interpretation of access

control causes many problems. Instead, confidentiality must be understood in a “fluid” sense, and not be determined by rigid rules, but rather as continuous space where risk and loss associated with confidentiality mechanisms and together are tied to dynamic situations. The combination of dynamicity and uncertainty creates new challenges for modelling and analysing software-intensive systems which need to be addressed in the future. In the ongoing research project FluidTrust [88] funded by the German Research Foundation, we take a first stance in this new direction of research by providing modelling languages and analysis techniques for reasoning about the system in development as well as for enforcing confidentiality in operation in highly dynamic and uncertain systems. First attempts to consider uncertainty in the context [39, 40] and in the structure and behaviour [270, 103] of a system have been developed based on the approaches presented in this thesis. These first attempts need to be improved and extended in the future.

Parts of our approaches to modelling and analysing confidentiality depend on a particular system and others can be reused for reasoning about other systems as well, as discussed in [228]. Security engineers often use catalogues to provide reusable artefacts to the community. In the future, we may investigate how parts of our approaches can be provided in form of catalogues and understand how engineers can apply such catalogues to provide and/or use partial analysis definitions. This is related to future work on the decomposition and composition of modelling languages and analysis techniques discussed in the following section.

The approach to attack propagation analysis may be extended in the future by investigating the automated creation of vulnerability models by combining existing vulnerability analysis techniques and reverse engineering approaches for technology-induced architectures, such as [145].

Another topic of future work is to investigate the scalability of our approaches to modelling and analysing confidentiality. Especially the approach to attack propagation analysis shows potential for further research as for every newly gained credential and newly compromised element all connected elements have to be checked because they may provide new attack possibilities. To address this, more efficient data-structures may be used in the future. Also our approaches to confidentiality analysis based on data flows proposed in this thesis can be examined for scalability in the future, for instance, depending on the number of model elements or the kind of label propagation. Moreover,

like discussed before, the effort required to create models and analyse confidentiality issues, both manually and using the automated approaches, may be investigated also for our approaches to modelling and analysing confidentiality to better understand when applying a model-based and automated approach pays off.

15.2.2 Decomposition and Composition of Modelling Languages and Analysis Techniques

In the Dagstuhl Seminar 19481 [77], we brought together members of the software engineering and formal methods communities and representatives from industry with the goal of establishing the foundations for a common understanding of the decomposition and composition of model-based analysis tools. We identified several challenges with respect to the decomposition and composition of model-based analyses and proposed first solutions to address these challenges in a book [114] created by the community as a result of the seminar. We collected topics of future work related to the decomposition and composition of model-based analyses and arranged them in form of a research roadmap. In the following, we discuss an excerpt of this research roadmap while the complete roadmap is described in [114].

Further research to understand the compositionality of specific quality properties and their analyses is required. While for some properties research has already advanced well, many other important properties remain for which compositionality is still not well understood. For example, research advances on compositionality of performance properties have been achieved in the last decades but compositionality of security properties is still not well understood [114].

Further investigation on the composition of analysis techniques is required in the future. We have introduced different forms of composition in the context of model-based analysis (black-, white-, and grey-box) in [246]. Getting a detailed understanding under which conditions each of them should be used and the consequences of use still remains a topic for future research. In this thesis, we gave examples with a special focus on performance simulation of information systems and business processes and described how these forms of composition can be implemented in existing simulators by discussing specific composition operators in Chapter 4. However, there is still a lot of research

required to come to a general understanding. Investigating these composition operators for other types of analysis techniques and other quality properties is one topic of future work. Moreover, further insights on the applicability of these composition operators can be gathered in case studies for analysis techniques in other disciplines we will conduct in the future.

An important challenge that needs to be addressed in the future is the decomposition of analysis techniques along the analysis features they provide. Based on first experience with analysis decomposition our hypothesis is that the decomposition of a model-based analysis technique can follow the decomposition of the modelling language it is based on like we proposed in this thesis. However, further research is necessary and case studies for different types of analysis techniques need to be conducted in the future to develop guidelines for the decomposition of analysis techniques similar to those we proposed for the decomposition of modelling languages in this thesis.

We have introduced fundamental concepts for the integration and orchestration of analysis tools in our previous work [116]. A comprehensive study of orchestration strategies and the contexts in which these are most appropriate, however, is a topic of future research. A related challenge is to develop a language providing appropriate primitives for the specification and operationalisation of new orchestration strategies building on the foundational principles introduced in our previous work [116]. Implementing orchestration strategies efficiently requires substantial future research. For example, a safe and general approach to modular language development across concrete and abstract syntax as well as semantics is missing. This requires future research in language engineering and tool support.

Getting a better understanding of the compositionality of semantics is an important topic of future work. The decomposition and composition concepts proposed in this thesis are a good foundation to investigate the compositionality of semantics when decomposing and composing model-based analyses along different quality properties and disciplines. The concepts proposed in this thesis are starting point based on which fundamental questions on semantically correctness, validity, semantics and property preservation, and proper executability when decomposing and composing model-based analyses can be answered.

An overview of additional future work on the decomposition and composition of modelling languages is given by Strittmatter [241] as summarised in the following.

Besides metamodels, grammars are commonly used to specify the syntax of a modelling language. A topic of future work is to apply the reference architecture described for metamodels in this thesis to grammar-based languages. Starting points are described, for example, in [53]. Moreover, the layers of the reference architecture proposed in the thesis are specific to languages for quality modelling and analysis. Similar reference architectures showing (some) other layers may be applied for modelling languages of other application areas. For these languages the specific layers of the reference architecture may change while the decomposition and composition concepts described in this thesis will remain as they are applicable to metamodels in general.

Another topic of future work refers to the decomposition of metamodels. Language features may be identified by analysing a large amount of models and, for example, clustering metaclasses that are instantiated by groups of models into language features. First attempts to clustering metamodels have been shown, for example in [243], and need to be further examined.

In the future, our decomposition and composition approach as well as the reference architecture may be better align with related initiatives like MPM [186] or ideas we had for COLD [59]. Moreover, applying language interfaces [68] can allow for information hiding and decoupling of tools from modelling languages.

Moreover, our decomposition and composition concepts as well as the reference architecture may be applied to refactor and consolidate various existing metamodels and thus gather new insights on the applicability of our approach in the future. For example, metamodels that support related concepts, such as the Palladio Component Model [206] and the Descartes Modeling Language [152], can be consolidated to share a common basis. We may also consider modelling languages that cover new quality properties and/or disciplines and new types of analyses. For example, we are currently working on basing aforementioned approaches to confidentiality modelling and analysis on a modular version of the Palladio Component Model. Furthermore, applying our approach to new domains, such as automotive, promises interesting future insights.

Another topic of future work is further investigation on the extension mechanisms for composing modelling languages. The extension mechanisms proposed in this thesis are based on EMOF [181]. In the future, we may survey and examine other forms of extension mechanism for modelling languages,

both metamodel-based and grammar-based modelling languages, by using and adapting the comparison criterion for extension mechanisms presented in [241]. The compatibility of different extension mechanisms may also be investigated in the future.

15.2.3 Bridging the Divergent Levels of Abstraction between Development and Operation

Our approaches to bridging the divergent levels of abstraction in modelling between development and operation can be extended and revised in the future as described in the following.

We will further broaden our approaches to better support the planning and execution phases of the MAPE control loop model. We have proposed first attempts in [111, 110, 200]. As next steps, we (i) will further investigate design space exploration and optimisation techniques to identify optimal candidate architectural models, (ii) will further investigate the execution of adaptation plans to allow for a maximum degree of automation where adaptation is possible without human intervention, and (iii) will examine approaches to efficient operator-in-the-loop adaptation in case of trade-off decisions, lack of information, or criticality of decisions. Live visualisation approaches have demonstrated support for human operators in cases where human intervention is required. In the future, we may extend our visualisation approach sketched in [110] by additional views to further improve support of human operators.

Another topic of future work is expanding the approaches proposed in this thesis by considering, besides monitoring data of the software application, additional kinds of data gathered in operation, such as sensor data of an automated production plant, and represent this data in architectural runtime models. These architectural runtime models then need to adhere to corresponding modelling languages. The modelling languages for automated production systems we proposed in Chapter 5, for example, can serve as a basis which then need to be extended for specific purposes. Furthermore, correspondence relations must be specified accordingly.

Our incremental model calibration may be extended in the future. We may apply genetic algorithms to analyse the entire behaviour and additional quality properties of services instead of focusing on performance parameters.

Another topic of future work is to extend the scalability experiments of the proposed approaches. Extensions to the scalability experiments may comprise the consideration of influence factors like variation in the complexity of user behaviour or optimisations using genetic algorithm.

Currently, the implementation of our consistency preservation rules are focused on the Java programming language and other specific technologies. However, the Vitruvius approach [147] our consistency preservation approach is based on allows for defining domain-specific metamodels and consistency preservation rules. In the future, we may adjust the metamodels and consistency preservation rules to make our approach independent of specific technologies.

Bibliography

[1] W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. "Time prediction based on process mining". In: *Information Systems* 36.2 (2011), pp. 450–475. ISSN: 03064379. DOI: 10.1016/j.is.2010.09.001. URL: http://linkinghub.elsevier.com/retrieve/pii/S0306437910000864.

[2] Takoua Abdellatif et al. "Automating information flow control in component-based distributed systems". In: *14th International ACM Sigsoft Symposium on Component-based Software Engineering*. ACM, 2011, pp. 73–82. ISBN: 978-1-4503-0723-9. DOI: 10.1145/2000229.2000241.

[3] Marwan Abi-Antoun, Daniel Wang, and Peter Torr. "Checking threat modeling data flow diagrams for implementation conformance and security". In: *22nd IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2007, pp. 393–396. ISBN: 978-1-59593-882-4. DOI: 10.1145/1321631.1321692.

[4] Vanessa Ackermann et al. "Black-box Learning of Parametric Dependencies for Performance Models". In: *13th Workshop on Models@run.time*. CEUR Workshop Proceedings Vol-2245. 2018, pp. 78–86.

[5] Ad Aerts et al. "Architectures in context: On the evolution of business, application software, and ICT platform architectures". In: *Information & Management* 41 (2004), pp. 781–794. DOI: 10.1016/j.im.2003.06.002.

[6] Wolfgang Ahrendt et al., eds. *Deductive Software Verification – The KeY Book*. Springer, 2016. ISBN: 978-3-319-49811-9. DOI: 10.1007/978-3-319-49812-6.

[7] M. Ugur Aksu et al. "Automated Generation of Attack Graphs Using NVD". In: *Eighth ACM Conference on Data and Application Security and Privacy*. ACM, 2018, pp. 135–142. ISBN: 9781450356329. DOI: 10.1145/3176258.3176339. URL: https://doi.org/10.1145/3176258.3176339.

[8] Omar Alam, Jörg Kienzle, and Gunter Mussbacher. "Concern-oriented software design". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2013, pp. 604–621. DOI: 10.1007/978-3-642-41533-3_37.

[9] Francesco Alberti, Alessandro Armando, and Silvio Ranise. "Efficient Symbolic Automated Analysis of Administrative Attribute-Based RBAC-Policies". In: *6th ACM Symposium on Information, Computer and Communications Security*. ACM, 2011, pp. 165–175. ISBN: 9781450305648. DOI: 10.1145/1966913.1966935.

[10] Aldeida Aleti et al. "Software Architecture Optimization Methods: A Systematic Literature Review". In: *IEEE Transactions on Software Engineering* 39.5 (2013), pp. 658–683. DOI: 10.1109/TSE.2012.64.

[11] Khaled Alghathbar, Csilla Farkas, and Duminda Wijesekera. "Securing UML Information Flow using FlowUML". In: *Journal of Research and Practice in Information Technology* 38.1 (2006), pp. 111–120. URL: https://50years.acs.org.au/content/dam/acs/50-years/journals/jrpit/JRPIT38.1.111.pdf.

[12] Khaled Alghathbar and Duminda Wijesekera. "authUML: A Three-Phased Framework to Analyze Access Control Specifications in Use Cases". In: *2003 ACM Workshop on Formal Methods in Security Engineering*. ACM, 2003, pp. 77–86. ISBN: 1581137818. DOI: 10.1145/1035429.1035438.

[13] Rasim Alguliyev, Yadigar Imamverdiyev, and Lyudmila Sukhostat. "Cyber-physical systems and their security issues". In: *Computers in Industry* 100 (2018), pp. 212–223. DOI: 10.1016/j.compind.2018.04.017.

[14] Mohamed Almorsy, John Grundy, and Amani S. Ibrahim. "Automated software architecture security risk analysis using formalized signatures". In: *35th International Conference on Software Engineering*. IEEE, 2013, pp. 662–671. ISBN: 978-1-4673-3076-3. DOI: 10.1109/ICSE.2013.6606612.

[15] Moussa Amrani et al. "Multi-paradigm modelling for cyber-physical systems: a descriptive framework". In: *Software & Systems Modeling* 20 (2021), pp. 611–639. DOI: 10.1007/s10270-021-00876-z. URL: https://doi.org/10.1007/s10270-021-00876-z.

[16] Moussa Amrani et al. “Towards a Formal Specification of Multi-Paradigm Modelling”. In: *22nd International Conference on Model Driven Engineering Languages and Systems Companion*. IEEE. 2019, pp. 419–424.

[17] Steven Arzt et al. “FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps”. In: *35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2014, pp. 259–269. DOI: 10.1145/2594291.2594299.

[18] AutomationML. URL: https://www.automationml.org/ (visited on 03/21/2022).

[19] Oscar Avila and Kelly Garcés. “Change Management Contributions for Business-IT Alignment”. In: *Lecture Notes in Business Information Processing*. Vol. 183. Springer, 2014. DOI: 10.1007/978-3-319-11460-6_14.

[20] A. Avizienis et al. “Basic concepts and taxonomy of dependable and secure computing”. In: *IEEE Transactions on Dependable and Secure Computing* 1.1 (2004), pp. 11–33. DOI: 10.1109/TDSC.2004.2.

[21] Wasana Bandara, Guy Gable, and Michael Rosemann. “Factors and measures of business process modelling: Model building through a multiple case study”. In: *European Journal of Information Systems* 14 (2005), pp. 347–360. DOI: 10.1057/palgrave.ejis.3000546.

[22] Luca Bassi et al. “A SysML-Based Methodology for Manufacturing Machinery Modeling and Design”. In: *IEEE/ASME Transactions on Mechatronics* 16.6 (2011), pp. 1049–1062. DOI: 10.1109/TMECH.2010.2073480.

[23] Falko Bause, Peter Buchholz, and Peter Kemper. “QPN-Tool for the Specification and Analysis of Hierarchically Combined Queueing Petri Nets”. In: *Quantitative Evaluation of Computing and Communication Systems*. Vol. 977. 1995, pp. 224–238.

[24] Matthias Becker, Markus Luckey, and Steffen Becker. “Performance Analysis of Self-adaptive Systems for Requirements Validation at Design-time”. In: *9th International ACM Sigsoft Conference on Quality of Software Architectures*. ACM, 2013, pp. 43–52. DOI: 10.1145/2465478.2465489.

[25] Steffen Becker. "Coupled model transformations for QoS enabled component-based software design". PhD thesis. 2008. 297 pp. ISBN: 978-3-86644-271-9. DOI: 10.5445/KSP/1000009095.

[26] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "The Palladio component model for model-driven performance prediction". In: *Journal of Systems and Software* 82.1 (2009). Special Issue: Software Performance - Modeling and Analysis, pp. 3–22. DOI: https://doi.org/10.1016/j.jss.2008.03.066.

[27] Steffen Becker et al. "Reverse Engineering Component Models for Quality Predictions". In: *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE, 2010, pp. 194–197. DOI: 10.1109/CSMR.2010.34.

[28] Nelly Bencomo et al. *Models@run.time*. Springer, 2014. ISBN: 978-3-319-08914-0. DOI: 10.1007/978-3-319-08915-7.

[29] PerOlof Bengtsson et al. "Architecture-level modifiability analysis (ALMA)." In: *Journal of Systems and Software* 69.1-2 (2004), pp. 129–147. DOI: 10.1016/S0164-1212(03)00080-3.

[30] PerOlof Bengtsson and Jan Bosch. "Architecture level prediction of software maintenance". In: *Third European Conference on Software Maintenance and Reengineering*. IEEE, 1999, pp. 139–147. DOI: 10.1109/CSMR.1999.756691.

[31] Bernhard J. Berger, Karsten Sohr, and Rainer Koschke. "Automatically Extracting Threats from Extended Data Flow Diagrams". In: *8th International Symposium on Engineering Secure Software and Systems*. Ed. by Juan Caballero, Eric Bodden, and Elias Athanasopoulos. Vol. 9639, Lecture Notes in Computer Science. Springer, 2016, pp. 56–71. ISBN: 978-3-319-30805-0. DOI: 10.1007/978-3-319-30806-7_4.

[32] Alexander van den Berghe et al. "A Lingua Franca for Security by Design". In: *2018 IEEE Cybersecurity Development*. IEEE, 2018, pp. 69–76. DOI: 10.1109/SecDev.2018.00017.

[33] Lorenzo Bettini. *Implementing Domain Specific Languages with Xtext and Xtend - Second Edition*. Packt Publishing, 2016. ISBN: 1786464969.

[34] Stefan Biffl et al. "Linking and versioning support for AutomationML: A model-driven engineering perspective". In: *2015 IEEE 13th International Conference on Industrial Informatics*. IEEE, 2015, pp. 499–506. DOI: 10.1109/INDIN.2015.7281784.

[35] BloodHound Enterprise. URL: https://bloodhoundenterprise.io/ (visited on 03/20/2022).

[36] Thierry Bodhuin et al. "Impact Analysis for Supporting the Co-Evolution of Business Processes and Supporting Software Systems". In: *CAiSE'04 Workshops in connection with The 16th Conference on Advanced Information Systems Engineering, Riga, Latvia, 7-11 June, 2004, Knowledge and Model Driven Information Systems Engineering for Networked Organisations, Proceedings, Vol. 2*. Ed. by Janis Grundspenkis and Marite Kirikova. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, 2004, pp. 146–150.

[37] Barry W. Boehm, Robert K Mcclean, and D. E. Urfrig. "Some experience with automated aids to the design of large-scale reliable software". In: *IEEE Transactions on Software Engineering* SE-1.1 (1975), pp. 125–133. ISSN: 1939-3520. DOI: 10.1109/TSE.1975.6312826.

[38] F.S. de Boer et al. "Change impact analysis of enterprise architectures". In: *IEEE International Conference on Information Reuse and Integration*. 2005, pp. 177–181. ISBN: 0-7803-9093-8. DOI: 10.1109/IRI-05.2005.1506470.

[39] Nicolas Boltz, Maximilian Walter, and Robert Heinrich. "Context-Based Confidentiality Analysis for Industrial IoT". In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2020, pp. 589–596. DOI: 10.1109/SEAA51224.2020.00096.

[40] Nicolas Boltz et al. "Handling Environmental Uncertainty in Design Time Access Control Analysis". In: *2022 48th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2022.

[41] Marcello Bonfé and Cesare Fantuzzi. "Design and verification of industrial logic controllers with UML and statecharts". In: *2003 IEEE Conference on Control Applications*. Vol. 2. 2003, pp. 1029–1034. ISBN: 0-7803-7729-X. DOI: 10.1109/CCA.2003.1223152.

[42] Grady Booch. "Object-oriented development". In: *IEEE Transactions on Software Engineering* SE-12.2 (1986), pp. 211–221. DOI: 10.1109/TSE.1986.6312937.

[43] L.C. Briand, Y. Labiche, and L. O'Sullivan. "Impact analysis and change management of UML models". In: *International Conference on Software Maintenance*. IEEE, 2003, pp. 256–265. DOI: 10.1109/ICSM.2003.1235428.

[44] Franz Brosch et al. “Architecture-Based Reliability Prediction with the Palladio Component Model”. In: *IEEE Transactions on Software Engineering* 38.6 (2012), pp. 1319–1339. DOI: 10.1109/TSE.2011.94.

[45] Fabian Brosig, Nikolaus Huber, and Samuel Kounev. “Automated extraction of architecture-level performance models of distributed component-based systems.” In: *2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2011, pp. 183–192. ISBN: 978-1-4577-1638-6. DOI: 10.1109/ASE.2011.6100052.

[46] Fabian Brosig, Samuel Kounev, and Klaus Krogmann. “Automated Extraction of Palladio Component Models from Running Enterprise Java Applications”. In: *1st International Workshop on Run-time mOdels for Self-managing Systems and Applications*. ACM, 2009, 10:1–10:10. ISBN: 978-963-9799-70-7.

[47] Andreas Brunnert, Christian Vögele, and Helmut Krcmar. “Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications”. In: *Computer Performance Engineering - 10th European Workshop*. Ed. by Maria Simonetta Balsamo, William J. Knottenbelt, and Andrea Marin. Vol. 8168. Lecture Notes in Computer Science. Springer, 2013, pp. 74–88. DOI: 10.1007/978-3-642-40725-3_7.

[48] Andreas Brunnert et al. *Performance-oriented DevOps: A Research Agenda*. Tech. rep. SPEC-RG-2015-01. SPEC Research Group — DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), 2015.

[49] Jim Buckley et al. “JITTAC: A Just-in-Time tool for architectural consistency”. In: *2013 35th International Conference on Software Engineering*. IEEE, 2013, pp. 1291–1294. DOI: 10.1109/ICSE.2013.6606700.

[50] Kiana Busch. “An Architecture-based Approach for Change Impact Analysis of Software-intensive Systems”. PhD thesis. Karlsruhe Institute of Technology (KIT), 2019. 275 pp. DOI: 10.5445/IR/1000097837.

[51] Kiana Busch et al. “A Model-Based Approach to Calculate Maintainability Task Lists of PLC Programs for Factory Automation”. In: *44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 2949–2954. DOI: 10.1109/IECON.2018.8591302.

[52] Arvid Butting et al. “Compositional Modelling Languages with Analytics and Construction Infrastructures Based on Object-Oriented Techniques—The MontiCore Approach”. In: *Composing Model-Based Analysis Tools*. Ed. by Robert Heinrich et al. Springer, 2021, pp. 217–234. ISBN: 978-3-030-81915-6. DOI: 10.1007/978-3-030-81915-6_11.

[53] Arvid Butting et al. “Systematic Composition of Independent Language Features”. In: *Journal of Systems and Software* 152 (2019), pp. 50–69. DOI: 10.1016/j.jss.2019.02.026.

[54] Gerardo Canfora et al. “A framework for QoS-aware binding and re-binding of composite web services”. In: *Journal of Systems and Software* 81.10 (2008), pp. 1754–1769. DOI: https://doi.org/10.1016/j.jss.2007.12.792.

[55] Ralf Carbon. “Architecture-centric software producibility analysis”. PhD thesis. Fraunhofer IESE, Kaiserslautern; University of Kaiserslautern, 2012. 202 pp. ISBN: 978-3-8396-0372-7.

[56] Paulo Carreira, Vasco Amaral, and Hans Vangheluwe. “Multi-Paradigm Modelling for Cyber-Physical Systems: Foundations”. In: *Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems*. Springer, 2020, pp. 1–14. ISBN: 978-3-030-43945-3. DOI: 10.1007/978-3-030-43946-0_1.

[57] Walter Cazzola and Edoardo Vacchi. “Neverlang 2–Componentised Language Development for the JVM”. In: *International Conference on Software Composition*. Springer. 2013, pp. 17–32. DOI: 10.1007/978-3-642-39614-4_2.

[58] B. Combemale, O. Barais, and A. Wortmann. “Language Engineering with the GEMOC Studio”. In: *2017 IEEE International Conference on Software Architecture Workshops*. IEEE, 2017, pp. 189–191. DOI: 10.1109/ICSAW.2017.61.

[59] Benoit Combemale et al. “Concern-oriented language development (COLD): Fostering reuse in language engineering”. In: *Computer Languages, Systems & Structures* 54 (2018), pp. 139–155.

[60] Marc Courtois and Murray Woodside. “Using Regression Splines for Software Performance Analysis”. In: *2nd International Workshop on Software and Performance*. ACM, 2000, pp. 105–114. DOI: 10.1145/350391.350416.

[61] Simon Cruanes et al. “Tool Integration with the Evidential Tool Bus”. In: *Lecture Notes in Computer Science*. Springer, 2013, pp. 275–294. DOI: 10.1007/978-3-642-35873-9_18.

[62] *CVE-2021-28374*. URL: https://nvd.nist.gov/vuln/detail/CVE-2021-28374 (visited on 03/29/2022).

[63] *CWE-312*. URL: https://cwe.mitre.org/data/definitions/312.html (visited on 03/29/2022).

[64] Krysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming*. Addison-Wesley, 2000. ISBN: 0201309777.

[65] Hoa Khanh Dam and Michael Winikoff. “Supporting change propagation in UML models”. In: *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–10. DOI: 10.1109/ICSM.2010.5609712.

[66] Darpa Cyber Grand Challenge. URL: https://www.darpa.mil/program/cyber-grand-challenge (visited on 03/20/2023).

[67] Lakshitha de Silva and Dharini Balasubramaniam. “Controlling software architecture erosion: A survey”. In: *Journal of Systems and Software* 85.1 (2012), pp. 132–151. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2011.07.036.

[68] Thomas Degueule, Benoit Combemale, and Jean-Marc Jézéquel. “On Language Interfaces”. In: *PAUSE: Present And Ulterior Software Engineering*. Ed. by Bertrand Meyer and Manuel Mazzara. Springer, 2017, pp. 65–75. DOI: 10.1007/978-3-319-67425-4_5.

[69] Thomas Degueule et al. “Melange: A meta-language for modular and reusable development of DSLs”. In: *ACM SIGPLAN International Conference on Software Language Engineering*. ACM. 2015, pp. 25–36. DOI: 10.1145/2814251.2814252.

[70] Christopher J. Deloglos, Carl R. Elks, and Ashraf Tantawy. “An Attacker Modeling Framework for the Assessment of Cyber-Physical Systems Security”. In: *Computer Safety, Reliability, and Security*. Vol. LNCS 12234. Springer, 2020, pp. 150–163. DOI: 10.1007/978-3-030-54549-9.

[71] Mina Deng et al. “A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements”. In: *Requirements Engineering* 16.1 (2011), pp. 3–32. DOI: 10.1007/s00766-010-0115-7.

[72] Elizabeth Denham. *Penatly Notice*. Penatly Notice COM0783542. United Kingdom: Information Commissioner's Office, Oct. 2020. URL: https://web.archive.org/web/20210620130131/https://edpb.europa.eu/sites/default/files/article-60-final-decisions/uk_2010-10_data_breach_security_of_processing_decisionpublic_final.pdf (visited on 08/02/2021).

[73] Elizabeth Denham. *Penatly Notice*. Penatly Notice COM0804337. United Kingdom: Information Commissioner's Office, Oct. 2020. URL: https://web.archive.org/web/20210802034347/https://edpb.europa.eu/sites/default/files/article-60-final-decisions/uk_2020-10_personal_data_breach_decisionpublic_final.pdf (visited on 08/02/2021).

[74] Markus von Detten. "Archimetrix: A Tool for Deficiency-Aware Software Architecture Reconstruction". In: *2012 19th Working Conference on Reverse Engineering*. IEEE, 2012, pp. 503–504. DOI: 10.1109/WCRE.2012.61.

[75] John Donaldson. *A Case Narrative of the Project Problems with the Denver Airport Baggage Handling System (DABHS)*. Tech. rep. Software Forensics Centre Technical Report TR 2002-01, 2002.

[76] Francisco Durán et al. "Amalgamation of domain specific languages with behaviour". In: *Journal of Logical and Algebraic Methods in Programming* 86.1 (2017), pp. 208–235. ISSN: 2352-2208. DOI: 10.1016/j.jlamp.2015.09.005.

[77] Francisco Durán et al. "Composing Model-Based Analysis Tools (Dagstuhl Seminar 19481)". In: *Dagstuhl Reports* 9.11 (2020). Ed. by Francisco Durán et al. ISSN: 2192-5283. DOI: 10.4230/DagRep.9.11.97. URL: https://drops.dagstuhl.de/opus/volltexte/2020/11985.

[78] Matthew B. Dwyer and Sebastian Elbaum. "Unifying verification and validation techniques". In: *FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 93–98. DOI: 10.1145/1882362.1882382.

[79] Stefan Dziwok et al. *The MechatronicUML Design Method: Process and Language for Platform-Independent Modeling*. Tech. rep. tr-ri-16-352. Version 1.0. Software Engineering Department, Fraunhofer IEM / Software Engineering Group, Heinz Nixdorf Institute, 2016.

[80] Aladin El Hamdouni, Abdelhak Seriai, and Marianne Huchard. "Component-based Architecture Recovery from Object Oriented Systems via Relational Concept Analysis". In: *7th International Conference on Concept Lattices and Their Applications*. Vol. 672 CEUR Workshop Proceedings. 2010, pp. 259–270.

[81] EMF Refactor. URL: https://www.eclipse.org/emf-refactor (visited on 03/27/2022).

[82] European Union. "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)". In: *Official Journal of the European Union* 59 (2016), pp. 1–88. URL: https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[83] Irene Eusgeld and Felix C. Freiling. "Introduction to Dependability Metrics". In: *Dependability Metrics: Advanced Lectures*. Ed. by Irene Eusgeld, Felix C. Freiling, and Ralf Reussner. Springer, 2008, pp. 1–4. ISBN: 978-3-540-68947-8. DOI: 10.1007/978-3-540-68947-8_1.

[84] Alexander Fay et al. "Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns". In: *Journal of Systems and Software* 101 (2015), pp. 221–235. DOI: 10.1016/j.jss.2014.12.028.

[85] Walid Fdhila, Stefanie Rinderle-Ma, and Manfred Reichert. "Change propagation in collaborative processes scenarios". In: *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing*. IEEE, 2012, pp. 452–461. DOI: 10.4108/icst.collaboratecom.2012.250408.

[86] Walid Fdhila et al. "Dealing with change in process choreographies: Design and implementation of propagation algorithms". In: *Information Systems* 49 (2015), pp. 1–24. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2014.10.004.

[87] Kathi Fisler et al. "Verification and Change-Impact Analysis of Access-Control Policies". In: *27th International Conference on Software Engineering*. ACM, 2005, pp. 196–205. ISBN: 1581139632. DOI: 10.1145/1062455.1062502.

[88] FluidTrust Project Website. URL: https://fluidtrust.ipd.kit.edu/home/ (visited on 03/01/2022).

[89] Jaime Font et al. “Automating the Variability Formalization of a Model Family by Means of Common Variability Language”. In: *19th International Conference on Software Product Line*. ACM, 2015, pp. 411–418. ISBN: 9781450336130. DOI: 10.1145/2791060.2793678.

[90] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2018. ISBN: 9780134757704.

[91] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010. ISBN: 0321712943.

[92] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014. ISBN: 0128002026.

[93] Maxime Frydman et al. “Automating Risk Analysis of Software Design Models”. In: *The Scientific World Journal* 2014 (2014), pp. 1–13. DOI: 10.1155/2014/805856.

[94] David Garlan et al. “Evolution styles: Foundations and tool support for software architecture evolution”. In: *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*. IEEE, 2009, pp. 131–140. DOI: 10.1109/WICSA.2009.5290799.

[95] Antonio Garmendia et al. “EMF splitter: A structured approach to EMF modularity”. In: *3rd Workshop on Extreme Modeling*. Vol. 1239 CEUR Workshop Proceedings. 2014, pp. 22–31.

[96] Susan Gasson. “A Framework for the Co-design of Business and IT Systems”. In: *41st Annual Hawaii International Conference on System Sciences*. IEEE, 2008, pp. 348–348. DOI: 10.1109/HICSS.2008.20.

[97] Christopher Gerking, David Schubert, and Eric Bodden. “Model Checking the Information Flow Security of Real-Time Systems”. In: *International Symposium on Engineering Secure Software and Systems*. Springer, 2018, pp. 27–43. ISBN: 978-3-319-94495-1. DOI: 10.1007/978-3-319-94496-8_3.

[98] Michael Goldapp, Ulrich Grottker, and Gregor Snelting. “Validierung softwaregesteuerter Meßsysteme durch Program Slicing und Constraint Solving”. In: *Statusseminar des BMBF Softwaretechnologie* (1996), pp. 405–425.

[99] Cláudio Gomes et al. “Co-Simulation: A Survey”. In: *ACM Computing Surveys* 51.3 (2018), pp. 1–33. ISSN: 0360-0300. DOI: 10.1145/3179993.

[100] Johannes Grohmann et al. “Detecting Parametric Dependencies for Performance Models Using Feature Selection Techniques”. In: *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE, 2019, pp. 309–322. DOI: 10.1109/MASCOTS.2019.00042.

[101] Johannes Grohmann et al. “SARDE: A Framework for Continuous and Self-Adaptive Resource Demand Estimation”. In: *ACM Transactions on Autonomous and Adaptive Systems* 15.2 (2021), pp. 1–31. ISSN: 1556-4665. DOI: 10.1145/3463369.

[102] IEEE Architecture Working Group. *IEEE Std 1471-2000, Recommended practice for architectural description of software-intensive systems*. Tech. rep. IEEE, 2000, pp. i–23.

[103] Sebastian Hahner. “Architectural Access Control Policy Refinement and Verification under Uncertainty”. In: *Companion Proceedings of the 15th European Conference on Software Architecture*. Vol. 2978. CEUR Workshop Proceedings. 2021.

[104] Daniel Hedin et al. “A Principled Approach to Tracking Information Flow in the Presence of Libraries”. In: *6th International Conference on Principles of Security and Trust*. Springer, 2017, pp. 49–70. ISBN: 9783662544549. DOI: 10.1007/978-3-662-54455-6_3.

[105] Robert Heinrich. “Aligning Business Process Quality and Information System Quality”. PhD thesis. Heidelberg University, Software Engineering Heidelberg, 2013. DOI: 10.11588/heidok.00016033. URL: http://www.ub.uni-heidelberg.de/archiv/16033.

[106] Robert Heinrich. “Architectural Run-time Models for Performance and Privacy Analysis in Dynamic Cloud Applications”. In: *ACM SIGMETRICS Performance Evaluation Review* 43.4 (2016), pp. 13–22. ISSN: 0163-5999. DOI: 10.1145/2897356.2897359.

[107] Robert Heinrich. “Architectural runtime models for integrating runtime observations and component-based models”. In: *Journal of Systems and Software* 169 (2020). ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2020.110722.

[108] Robert Heinrich, Kiana Busch, and Sandro Koch. "A Methodology for Domain-spanning Change Impact Analysis". In: *2018 44th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE, 2018, pp. 326–330. DOI: 10.1109/SEAA.2018.00060.

[109] Robert Heinrich, Misha Strittmatter, and Ralf Heinrich Reussner. "A Layered Reference Architecture for Metamodels to Tailor Quality Modeling and Analysis". In: *IEEE Transactions on Software Engineering* 47.4 (2019), pp. 775–800. ISSN: 0098-5589. DOI: 10.1109/TSE.2019.2903797.

[110] Robert Heinrich et al. "An Architectural Model-Based Approach to Quality-Aware DevOps in Cloud Applications". In: *Software Architecture for Big Data and the Cloud*. Ed. by Ivan Mistrik et al. Morgan Kaufmann, 2017, pp. 69–89. ISBN: 978-0-12-805467-3. DOI: 10.1016/B978-0-12-805467-3.00005-3.

[111] Robert Heinrich et al. "Architectural run-time models for operator-in-the-loop adaptation of cloud applications". In: *IEEE 9th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments*. IEEE, 2015, pp. 36–40. DOI: 10.1109/MESOCA.2015.7328124.

[112] Robert Heinrich et al. "Architecture-based change impact analysis in cross-disciplinary automated production systems". In: *Journal of Systems and Software* 146 (2018), pp. 167–185. ISSN: 0164-1212. DOI: 10.1016/j.jss.2018.08.058.

[113] Robert Heinrich et al. "Challenges in the Evolution of Palladio—Refactoring Design Smells in a Historically-Grown Approach to Software Architecture Analysis". In: *Composing Model-Based Analysis Tools*. Ed. by Robert Heinrich et al. Springer, 2021, pp. 235–257. ISBN: 978-3-030-81915-6. DOI: 10.1007/978-3-030-81915-6_11.

[114] Robert Heinrich et al. *Composing Model-Based Analysis Tools*. Ed. by Robert Heinrich et al. Springer, 2021. ISBN: 978-3-030-81915-6. DOI: 10.1007/978-3-030-81915-6.

[115] Robert Heinrich et al. "Integrating business process simulation and information system simulation for performance prediction". In: *Software & Systems Modeling* 16.1 (2017), pp. 257–277. ISSN: 1619-1366. DOI: 10.1007/s10270-015-0457-1.

[116] Robert Heinrich et al. “Integration and Orchestration of Analysis Tools”. In: *Composing Model-Based Analysis Tools*. Ed. by Robert Heinrich et al. Springer, 2021, pp. 71–95. ISBN: 978-3-030-81915-6. DOI: 10.1007/978-3-030-81915-6_11.

[117] Robert Heinrich et al. “The Palladio-Bench for Modeling and Simulating Software Architectures”. In: *40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 2018, pp. 37–40. DOI: 10.1145/3183440.3183474.

[118] Nikolas Roman Herbst et al. “Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning”. In: *4th ACM/SPEC International Conference on Performance Engineering*. ACM, 2013, pp. 187–198. DOI: 10.1145/2479871.2479899.

[119] Sebastian Herold et al. “CoCoME - The Common Component Modeling Example”. In: *The Common Component Modeling Example: Comparing Software Component Models*. Ed. by Andreas Rausch et al. Springer, 2008, pp. 16–53. ISBN: 978-3-540-85289-6. DOI: 10.1007/978-3-540-85289-6_3.

[120] Bernhard Hoisl, Stefan Sobernig, and Mark Strembeck. “Modeling and enforcing secure object flows in process-driven SOAs: an integrated model-driven approach”. In: *Software & Systems Modeling* 13.2 (2014), pp. 513–548. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-012-0263-y.

[121] Katrin Hölldobler, Bernhard Rumpe, and Andreas Wortmann. “Software language engineering in the large: towards composing and deriving languages”. In: *Computer Languages, Systems & Structures* 54 (2018), pp. 386–405. DOI: 10.1016/j.cl.2018.08.002.

[122] Kevin Soo Hoo, Andrew W Sudbury, and Andrew R Jaquith. “Tangible ROI through Secure Software Engineering”. In: *Secure Business Quarterly* 1.2 (2001), pp. 1–3.

[123] André van Hoorn. “Model-Driven Online Capacity Management for Component-Based Software Systems”. PhD thesis. Kiel University, Department of Computer Science, 2014. ISBN: 978-3-7357-5118-8.

[124] IEEE 1278.2-1995. *Standard for Distributed Interactive Simulation - Communication Services and Profiles*. 1995. DOI: 10.1109/IEEESTD.1996.80824.

[125] IEEE 1516-2010. *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) (Revision of IEEE Std 1516-2000)*. 2010. DOI: 10.1109/IEEESTD.2010.5553440.

[126] Center for Internet Security. *The SolarWinds Cyber-Attack: What You Need to Know*. 2021. URL: https://www.cisecurity.org/solarwinds/ (visited on 02/02/2022).

[127] Jim Isaak and Mina J Hanna. "User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection". In: *Computer* 51.8 (2018), pp. 56–59. DOI: 10.1109/MC.2018.3191268.

[128] ISO/IEC 25010:2011. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. 2011.

[129] ISO/IEC 27000:2018(E). *Information technology – Security techniques – Information security management systems – Overview and vocabulary*. 2018.

[130] ISO/IEC/IEEE 42010:2011(E). *Systems and software engineering - Architecture description (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*. 2011. DOI: 10.1109/IEEESTD.2011.6129467.

[131] ISO/TC 176. *Quality Concepts and Terminology, part 1: Generic Terms and Definitions, Document ISO/TC 176/SC 1 N 93*. 1992.

[132] Dragan Ivanovic, Manuel Carro, and Manuel Hermenegildo. "Constraint-based runtime prediction of SLA violations in service orchestrations". In: *International Conference on Service-Oriented Computing*. Springer, 2011, pp. 62–76. DOI: 10.1007/978-3-642-25535-9_5.

[133] Pooyan Jamshidi and Claus Pahl. "Business process and software architecture model co-evolution patterns". In: *2012 4th International Workshop on Modeling in Software Engineering*. IEEE, 2012, pp. 91–97. DOI: 10.1109/MISE.2012.6226021.

[134] Hans Jense, N.H.L. Kuijpers, and A.C.M. Dumay. "DIS and HLA: connecting people, simulations and simulators in the military, space and civil domains". In: *48th International Astronautical Congress*. 1997.

[135] Reiner Jung and Marc Adolf. "Extracting Realistic User Behavior Models". In: *Software Engineering Workshops 2018*. Vol. 2066. CEUR Workshop Proceedings. 2018, pp. 47–50.

[136] Jan Jürjens. *Secure Systems Development with UML*. Springer, 2005. ISBN: 978-3-540-00701-2. DOI: 10.1007/b137706.

[137] Jan Jürjens. "UMLsec: Extending UML for Secure Systems Development". In: *UML 2002 — The Unified Modeling Language*. Vol. 2460, Lecture Notes in Computer Science. Springer, 2002, pp. 412–425. ISBN: 978-3-540-44254-7. DOI: 10.1007/3-540-45800-X_32.

[138] Huzefa Kagdi, Jonathan I. Maletic, and Andrew Sutton. "Context-Free Slicing of UML Class Models". In: *21st IEEE International Conference on Software Maintenance*. IEEE, 2005, pp. 635–638. ISBN: 0769523684. DOI: 10.1109/ICSM.2005.34.

[139] Kuzman Katkalov et al. "Model-Driven Development of Information Flow-Secure Systems with IFlow". In: *2013 International Conference on Social Computing*. IEEE, 2013, pp. 51–56. DOI: 10.1109/SocialCom.2013.14.

[140] Lennart C. L. Kats and Eelco Visser. "The Spoofax Language Workbench. Rules for Declarative Specification of Languages and IDEs". In: *25th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM, 2010, pp. 444–463. DOI: 10.1145/1869459.1869497.

[141] R. Kazman et al. "SAAM: A Method for Analyzing the Properties of Software Architectures". In: *16th International Conference on Software Engineering*. IEEE Computer Society Press, 1994, pp. 81–90. DOI: 10.1109/ICSE.1994.296768.

[142] Rick Kazman et al. "The Architecture Tradeoff Analysis Method". In: *4th International Conference on Engineering of Complex Computer Systems*. IEEE, 1998, pp. 68–78. DOI: 10.1109/ICECCS.1998.706657.

[143] Udo Kelter, Jürgen Wehren, and Jörg Niere. "A generic difference algorithm for UML models". In: *Software Engineering 2005*. Ed. by Peter Liggesmeyer, Klaus Pohl, and Michael Goedicke. Gesellschaft für Informatik e.V., 2005, pp. 105–116. ISBN: 3-88579-393-8.

[144] Emre Kiciman and Benjamin Livshits. "AjaxScope: A Platform for Remotely Monitoring the Client-Side Behavior of Web 2.0 Applications". In: *ACM Transactions on the Web* 4.4 (2010). ISSN: 1559-1131. DOI: 10.1145/1841909.1841910.

[145] Yves Richard Kirschner. "Model-Driven Reverse Engineering of Technology-Induced Architecture for Quality Prediction". In: *Companion Proceedings of the 15th European Conference on Software Architecture*. Vol. 2978. CEUR Workshop Proceedings. 2021.

[146] Jóakim v. Kistowski, Nikolas Roman Herbst, and Samuel Kounev. "Modeling Variations in Load Intensity over Time". In: *3rd International Workshop on Large Scale Testing*. ACM, 2014, pp. 1–4. DOI: 10.1145/2577036.2577037.

[147] Heiko Klare et al. "Enabling consistency in view-based system development - The Vitruvius approach". In: *Journal of Systems and Software* 171 (2021). ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110815.

[148] Sandro Koch et al. "Feature-based Investigation of Simulation Structure and Behaviour". In: *16th European Conference on Software Architecture*. Springer, 2022.

[149] Marco Konersmann. "Explicitly Integrated Architecture - An Approach for Integrating Software Architecture Model Information with Program Code". PhD thesis. University of Duisburg-Essen, Paluno, 2018. URL: https://nbn-resolving.org/urn:nbn:de:hbz:464-20180509-094231-3.

[150] Marco Konersmann and Jens Holschbach. "Automatic Synchronization of Allocation Models with Running Software". In: *SoftwaretechnikTrends* 36.4 (2016). ISSN: 0720-8928.

[151] Barbara Kordy, Ludovic Pièvre-Cambacédès, and Patrick Schweitzer. "DAG-based attack and defense modeling: Don't miss the forest for the attack trees". In: *Computer Science Review* 13-14 (2014), pp. 1–38. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2014.07.001.

[152] Samuel Kounev, Fabian Brosig, and Nikolaus Huber. *The Descartes Modeling Language*. Tech. rep. Department of Computer Science, University of Wuerzburg, 2014, p. 91. URL: http://www.descartes-research.net/dml/.

[153] M. Kradolfer and A. Geppert. "Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration". In: *Fourth IFCIS International Conference on Cooperative Information Systems*. IEEE, 1999, pp. 104–114. DOI: 10.1109/COOPIS.1999.792162.

[154] Klaus Krogmann. "Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis". PhD thesis. Karlsruhe Institute of Technology (KIT), 2012. DOI: 10.5445/KSP/1000025617.

[155] Klaus Krogmann, Michael Kuperberg, and Ralf Reussner. "Using Genetic Search for Reverse Engineering of Parametric Behavior Models for Performance Prediction". In: *IEEE Transactions on Software Engineering* 36.6 (2010), pp. 865–877. ISSN: 0098-5589. DOI: 10.1109/TSE.2010.69.

[156] Thomas Kühn, Walter Cazzola, and Diego Mathias Olivares. "Choosy and picky: configuration of language product lines". In: *19th International Conference on Software Product Line*. ACM. 2015, pp. 71–80. DOI: 10.1145/2791060.2791092.

[157] Tri A. Kurniawan et al. "Relationship-Preserving Change Propagation in Process Ecosystems". In: *10th International Conference on Service-Oriented Computing*. Springer, 2012, pp. 63–78. ISBN: 9783642343209. DOI: 10.1007/978-3-642-34321-6_5.

[158] Jan Ladiges, Alexander Fay, and Winfried Lamersdorf. "Automated Determining of Manufacturing Properties and Their Evolutionary Changes from Event Traces". In: *Intelligent Industrial Systems* 2.2 (2016), pp. 163–178. DOI: 10.1007/s40903-016-0048-7.

[159] M. Langhammer et al. "Automated Extraction of Rich Software Models from Limited System Information". In: *13th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2016, pp. 99–108. DOI: 10.1109/WICSA.2016.35.

[160] Michael Langhammer. "Automated Coevolution of Source Code and Software Architecture Models". PhD thesis. Karlsruhe Institute of Technology (KIT), 2019. 339 pp. DOI: 10.5445/KSP/1000081447.

[161] Kevin Lano and Shekoufeh Kolahdouz Rahimi. "Slicing Techniques for UML Models". In: *Journal of Object Technology* 10 (2011), pp. 1–49. ISSN: 1660-1769. DOI: 10.5381/jot.2011.10.1.a11.

[162] Juan de Lara, Hans Vangheluwe, and Manuel Alfonseca. "Meta-modelling and graph grammars for multi-paradigm modelling in AToM 3". In: *Software & Systems Modeling* 3 (2004), pp. 194–209. DOI: 10.1007/s10270-003-0047-5.

[163] Manuel Leduc, Thomas Degueule, and Benoit Combemale. "Modular Language Composition for the Masses". In: *11th ACM SIGPLAN International Conference on Software Language Engineering*. ACM, 2018, pp. 47–59. ISBN: 9781450360296. DOI: 10.1145/3276604.3276622.

[164] Meir M. Lehman. “On understanding laws, evolution, and conservation in the large-program life cycle”. In: *Journal of Systems and Software* 1 (1980), pp. 213–221. DOI: 10.1016/0164-1212(79)90022-0.

[165] Steffen Lehnert. *A review of software change impact analysis*. Tech. rep. Ilmenau University of Technology, Department of Software Systems / Process Informatics, 2011. URL: https://www.db-thueringen.de/receive/dbt_mods_00019544.

[166] Steffen Lehnert. “A Taxonomy for Software Change Impact Analysis”. In: *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*. ACM, 2011, pp. 41–50. ISBN: 9781450308489. DOI: 10.1145/2024445.2024454.

[167] Sebastian Lehrig. “Applying Architectural Templates for Design-Time Scalability and Elasticity Analyses of SaaS Applications”. In: *2nd International Workshop on Hot Topics in Cloud Service Scalability*. ACM, 2014. DOI: 10.1145/2649563.2649573.

[168] Bennet P. Lientz and Burton E. Swanson. *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley, 1980. ISBN: 0201042053.

[169] Yuehua Lin, Jeff Gray, and Frédéric Jouault. “DSMDiff: a differentiation tool for domain-specific models”. In: *European Journal of Information Systems* 16.4 (2007), pp. 349–361. ISSN: 1476-9344. DOI: 10.1057/palgrave.ejis.3000685.

[170] K.C. Liu, L.L. Sun, and K. Bennett. “Co-design of business and IT systems - Introduction by guest editors.” In: *Information Systems Frontiers* 4.3 (2002), pp. 251–256. DOI: 10.1023/A:1019942501848.

[171] Robert von Massow, André van Hoorn, and Wilhelm Hasselbring. “Performance Simulation of Runtime Reconfigurable Component-Based Software Architectures.” In: *5th European Conference on Software Architecture*. Vol. 6903. Lecture Notes in Computer Science. Springer, 2011, pp. 43–58. ISBN: 978-3-642-23797-3. DOI: 10.1007/978-3-642-23798-0_5.

[172] Manar Mazkatli et al. “Incremental Calibration of Architectural Performance Models with Parametric Dependencies”. In: *2020 IEEE International Conference on Software Architecture*. IEEE, 2020, pp. 23–34. DOI: 10.1109/ICSA47634.2020.00011.

[173] Gary McGraw. *Software Security - Building Security In*. Addison-Wesley Professional, 2006. ISBN: 9780321356703.

[174] Daniel A. Menascé et al. "A Methodology for Workload Characterization of E-commerce Sites". In: *1st ACM Conference on Electronic Commerce*. ACM, 1999, pp. 119–128. DOI: 10.1145/336992.337024.

[175] David Méndez-Acuña et al. "Leveraging software product lines engineering in the development of external dsls: A systematic literature review". In: *Computer Languages, Systems & Structures* 46 (2016), pp. 206–235.

[176] David Méndez-Acuña et al. "Puzzle: A Tool for Analyzing and Extracting Specification Clones in DSLs". In: *Software Reuse: Bridging with Social-Awareness*. Springer, 2016, pp. 393–396. ISBN: 978-3-319-35122-3. DOI: 10.1007/978-3-319-35122-3_26.

[177] Philipp Merkle and Jörg Henss. "EventSim – An Event-driven Palladio Software Architecture Simulator". In: *Palladio Days 2011*. Karlsruhe Reports in Informatics. 2011, pp. 15–22. URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000025188.

[178] Marjan Mernik. "An Object-oriented Approach to Language Compositions for Software Language Engineering". In: *Journal of Systems and Software* 86 (2013), pp. 2451–2464. DOI: 10.1016/j.jss.2013.04.087.

[179] Ningfang Mi et al. "Burstiness in multi-tier applications: symptoms, causes, and new models". In: *Middleware 2008 – ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Ed. by Valerie Issarny and Richard Schantz. Springer, 2008, pp. 265–286. ISBN: 3-540-89855-7. DOI: 10.1007/978-3-540-89856-6_14.

[180] Microsoft Corporation and iSEC Partners, Inc. *Microsoft SDL: Return-on-Investment*. accessed 2022-03-25. 2009. URL: https://web.archive.org/web/20210925085942/https://www.nccgroup.com/globalassets/our-research/us/whitepapers/isec-partners---microsoft-sdl-return-on-investment.pdf.

[181] Object Management Group (OMG). *MOF 2.5.1 Core Specification (formal/2016-11-01)*. 2016. URL: https://www.omg.org/spec/MOF/2.5.1/.

[182] Angela Monaghan. *Timeline of trouble: how the TSB IT meltdown unfolded*. The Guardian, UK. accessed 2022-04-21. June 2018. URL: https://www.theguardian.com/business/2018/jun/06/timeline-of-trouble-how-the-tsb-it-meltdown-unfolded.

[183] David Monschein. "Enabling Consistency between Software Artefacts for Software Adaption and Evolution". Master Thesis. Karlsruher Institut für Technologie (KIT), 2020.

[184] David Monschein et al. "Enabling Consistency between Software Artefacts for Software Adaption and Evolution". In: *2021 IEEE 18th International Conference on Software Architecture*. IEEE, 2021, pp. 1–12. DOI: 10.1109/ICSA51549.2021.00009.

[185] B. Morin et al. "Models@Run.time to Support Dynamic Adaptation". In: *IEEE Computer* 42.10 (2009), pp. 44–51. DOI: 10.1109/MC.2009.327.

[186] Pieter J. Mosterman and Hans Vangheluwe. "Computer Automated Multi-Paradigm Modeling: An Introduction". In: *SIMULATION* 80.9 (2004), pp. 433–450. DOI: 10.1177/0037549704050532.

[187] G.C. Murphy, D. Notkin, and K.J. Sullivan. "Software reflexion models: bridging the gap between design and implementation". In: *IEEE Transactions on Software Engineering* 27.4 (2001), pp. 364–380. DOI: 10.1109/32.917525.

[188] Matthias Naab. "Enhancing architecture design methods for improved flexibility in long-living information systems". PhD thesis. Fraunhofer IESE, Kaiserslautern, 2012. 232 pp. ISBN: 978-3-8396-0477-9.

[189] James O'Brien and George Marakas. *Introduction to Information Systems*. 15th ed. McGraw-Hill, Inc., 2012. ISBN: 0070167087.

[190] Object Management Group (OMG). *Business Process Model and Notation (BPMN) Version 2.0 (formal/11-01-03)*. 2011. URL: http://www.omg.org/spec/BPMN/2.0.

[191] Object Management Group (OMG). *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems (formal/2009-11-02)*. 2009. URL: https://www.omg.org/spec/MARTE/1.0/PDF.

[192] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. "Runtime Software Adaptation: Framework, Approaches, and Styles". In: *Companion of the 30th International Conference on Software Engineering*. ACM, 2008, pp. 899–910. DOI: 10.1145/1370175.1370181.

[193] Karl J. Ottenstein and Linda M. Ottenstein. “The Program Dependence Graph in a Software Development Environment”. In: *First ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*. ACM, 1984, pp. 177–184. ISBN: 0897911318. DOI: 10.1145/800020.808263.

[194] Nathaniel Palmer. “Workflow Schema”. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Springer, 2009, pp. 3558–3558. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_819.

[195] Behrooz Parhami. “From Defects to Failures: A View of Dependable Computing”. In: *ACM SIGARCH Computer Architecture News* 16.4 (1988), pp. 157–168. ISSN: 0163-5964. DOI: 10.1145/54331.54345. URL: https://doi.org/10.1145/54331.54345.

[196] Daniel J. Paulish and Len Bass. *Architecture-Centric Software Project Management: A Practical Guide*. Addison-Wesley, 2001. ISBN: 0201734095.

[197] C. Pietsch et al. “SiPL – A Delta-Based Modeling Framework for Software Product Line Engineering”. In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2015, pp. 852–857. DOI: 10.1109/ASE.2015.106.

[198] Nikolaos Polatidis, Michalis Pavlidis, and Haralambos Mouratidis. “Cyber-attack path discovery in a dynamic supply chain maritime risk management system”. In: *Computer Standards & Interfaces* 56 (2018), pp. 74–82. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2017.09.006.

[199] Nikolaos Polatidis et al. “From product recommendation to cyber-attack prediction: generating attack graphs and predicting future attacks”. In: *Evolving Systems* 11 (2020), pp. 479–490. DOI: 10.1007/s12530-018-9234-z.

[200] Tobias Pöppke. “Design Space Exploration for Adaptation Planning in Cloud-based Applications”. Master Thesis. Karlsruher Institut für Technologie (KIT), 2017.

[201] Herbert Prähofer et al. “Feature-oriented development in industrial automation software ecosystems: Development scenarios and tool support”. In: *2016 IEEE 14th International Conference on Industrial Informatics*. IEEE, 2016, pp. 1218–1223. DOI: 10.1109/INDIN.2016.7819353.

[202] Cui Qin and Holger Eichelberger. "Impact-minimizing Runtime Switching of Distributed Stream Processing Algorithms". In: *EDBT/ICDT 2016 Workshops*. Vol. 1558. CEUR Workshop Proceedings. 2016.

[203] M. Reichert et al. "Adaptive process management with ADEPT2". In: *21st International Conference on Data Engineering*. IEEE, 2005, pp. 1113–1114. DOI: 10.1109/ICDE.2005.17.

[204] Manfred Reichert and Peter Dadam. "ADEPT flex - Supporting Dynamic Changes of Workflows Without Loosing Control". In: *Journal of Intelligent Information Systems* 10 (1998), pp. 93–129. DOI: 10.1023/A:1008604709862.

[205] Thomas Reps. "Optimal-Time Incremental Semantic Analysis for Syntax-Directed Editors". In: *9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 1982, pp. 169–176. ISBN: 0897910656. DOI: 10.1145/582153.582172.

[206] Ralf H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, 2016. 408 pp. ISBN: 9780262034760. URL: https://mitpress.mit.edu/books/modeling-and-simulating-software-architectures.

[207] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. "Correctness criteria for dynamic changes in workflow systems––a survey". In: *Data & Knowledge Engineering* 50.1 (2004), pp. 9–34. ISSN: 0169-023X. DOI: https://doi.org/10.1016/j.datak.2004.01.002.

[208] Stefanie Rinderle, Andreas Wombacher, and Manfred Reichert. "Evolution of Process Choreographies in DYCHOR". In: *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*. Ed. by Robert Meersman and Zahir Tari. Vol. 4275. Lecture Notes in Computer Science. Springer, 2006, pp. 273–290. DOI: 10.1007/11914853_17.

[209] Stefanie Rinderle, Andreas Wombacher, and Manfred Reichert. "On the Controlled Evolution of Process Choreographies". In: *22nd International Conference on Data Engineering*. Ed. by Ling Liu et al. IEEE, 2006, p. 124. DOI: 10.1109/ICDE.2006.108.

[210] Kiana Rostami et al. "Architecture-based Assessment and Planning of Change Requests". In: *11th International ACM SIGSOFT Conference on Quality of Software Architectures*. ACM, 2015, pp. 21–30. ISBN: 978-1-4503-3470-9. DOI: 10.1145/2737182.2737198.

[211] Kiana Rostami et al. “Architecture-based Change Impact Analysis in Information Systems and Business Processes”. In: *2017 IEEE International Conference on Software Architecture*. IEEE, 2017, pp. 179–188. ISBN: 978-1-5090-5729-0. DOI: 10.1109/ICSA.2017.17.

[212] Kiana Rostami et al. “Reconstructing Development Artifacts for Change Impact Analysis”. In: *Softwaretechnik-Trends* 37.2 (2017). ISSN: 0720-8928.

[213] G. Ruffo et al. “WALTy: a user behavior tailored tool for evaluating Web application performance”. In: *3rd IEEE International Symposium on Network Computing and Applications*. IEEE, 2004, pp. 77–86. DOI: 10.1109/NCA.2004.1347765.

[214] Bernhard Rumpe. *Agile Modeling with UML: Code Generation, Testing, Refactoring*. Springer, 2017. ISBN: 978-3319588612.

[215] Bernhard Rumpe. “Towards model and language composition”. In: *1st Workshop on the Globalization of Domain Specific Languages*. ACM, 2013, pp. 4–7. DOI: 10.1145/2489812.2489814.

[216] Bernhard Rumpe, Katrin Hölldobler, and Oliver Kautz. *MontiCore Language Workbench and Library Handbook: Edition 2021*. Shaker Verlag, 2021. DOI: 10.2370/9783844080100.

[217] Tobias Runge et al. “Lattice-Based Information Flow Control-by-Construction for Security-by-Design”. In: *8th International Conference on Formal Methods in Software Engineering*. ACM, 2020, pp. 44–54. DOI: 10.1145/3372020.3391565.

[218] Barbara G. Ryder and Frank Tip. “Change impact analysis for object-oriented programs”. In: *2001 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis For Software Tools and Engineering*. Ed. by John Field and Gregor Snelting. ACM, 2001, pp. 46–53. DOI: 10.1145/379605.379661.

[219] George Rzevski. “On conceptual design of intelligent mechatronic systems”. In: *Mechatronics* 13 (2003), pp. 1029–1044. DOI: 10.1016/S0957-4158(03)00041-2.

[220] A. Sabelfeld and A.C. Myers. “Language-based information-flow security”. In: *IEEE Journal on Selected Areas in Communications* 21.1 (2003), pp. 5–19. DOI: 10.1109/JSAC.2002.806121.

[221] Shazia Sadiq and Maria Orlowska. “Architectural Considerations in Systems Supporting Dynamic Workflow Modification”. In: *Workshop on Software Architectures for Business Process Management*. University of Karlsruhe, 1999, pp. 118–134.

[222] Wasim Sadiq, Olivera Marjanovic, and Maria Orlowska. “Managing Change And Time In Dynamic Workflow Processes”. In: *International Journal of Cooperative Information Systems* 9 (2000), pp. 93–116. DOI: 10.1142/S0218843000000077.

[223] Navid Sani, Shokoofeh Ketabchi, and Kecheng Liu. “The Co-design of Business and IT Systems: A Case in Supply Chain Management”. In: *2012 International Conference on Information Systems, Technology and Management*. Vol. 285, Communications in Computer and Information Science. Springer, 2012, pp. 13–27. ISBN: 978-3-642-29165-4. DOI: 10.1007/978-3-642-29166-1_2.

[224] B. Schmerl et al. “Discovering Architectures from Running Systems”. In: *IEEE Transactions on Software Engineering* 32.7 (2006), pp. 454–466. ISSN: 0098-5589. DOI: 10.1109/TSE.2006.66.

[225] Eric Schmieders and Andreas Metzger. “Preventing performance violations of service compositions using assumption-based run-time verification”. In: *Towards a Service-Based Internet*. Vol. 6994, Lecture Notes in Computer Science. Springer, 2011, pp. 194–205. DOI: 10.1007/978-3-642-24755-2_19.

[226] Stephan Seifermann. “Architectural Data Flow Analysis for Detecting Violations of Confidentiality Requirements”. PhD thesis. Karlsruhe Institute of Technology (KIT), 2022.

[227] Stephan Seifermann, Robert Heinrich, and Ralf H. Reussner. “Data-Driven Software Architecture for Analyzing Confidentiality”. In: *IEEE International Conference on Software Architecture*. IEEE, 2019, pp. 1–10. DOI: 10.1109/ICSA.2019.00009.

[228] Stephan Seifermann et al. “Detecting Violations of Access Control and Information Flow Policies in Data Flow Diagrams”. In: *Journal of Systems and Software* 184 (2022). ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2021.111138.

[229] Adam Shostack. *Threat Modeling: Designing for Security*. 1st edition. Wiley Publishing, 2014. ISBN: 1118809998.

[230] Laurens Sion et al. “Solution-aware data flow diagrams for security threat modeling”. In: *ACM Symposium on Applied Computing*. ACM, 2018, pp. 1425–1432. ISBN: 978-1-4503-5191-1. DOI: 10.1145/3167132.3167285.

[231] Connie U. Smith. *Performance engineering of software systems*. Software Engineering Institute series in software engineering. Addison-Wesley, 1990. ISBN: 978-0-201-53769-7.

[232] Gregor Snelting et al. “Checking probabilistic noninterference using JOANA”. In: *it-Information Technology* 56.6 (2014), pp. 280–287. DOI: 10.1515/itit-2014-1051.

[233] Teodor Sommestad, Mathias Ekstedt, and Hannes Holm. “The Cyber Security Modeling Language: A Tool for Assessing the Vulnerability of Enterprise System Architectures”. In: *IEEE Systems Journal* 7.3 (2013), pp. 363–373. DOI: 10.1109/JSYST.2012.2221853.

[234] Hui Song et al. “Supporting runtime software architecture: A bidirectional-transformation-based approach”. In: *Journal of Systems and Software* 84.5 (2011), pp. 711–723. ISSN: 0164-1212. DOI: 10.1016/j.jss.2010.12.009.

[235] Martín Soto. “Delta-P: Model Comparison Using Semantic Web Standards”. In: *Softwaretechnik-Trends* 27.2 (2007). ISSN: 0720-8928.

[236] Simon Spinner et al. “Evaluating approaches to resource demand estimation”. In: *Performance Evaluation* 92 (2015), pp. 51–71. ISSN: 0166-5316. DOI: https://doi.org/10.1016/j.peva.2015.07.005.

[237] Simon Spinner et al. “LibReDE: A Library for Resource Demand Estimation”. In: *5th ACM/SPEC International Conference on Performance Engineering*. ACM, 2014, pp. 227–228. DOI: 10.1145/2568088.2576093.

[238] Simon Spinner et al. “Online model learning for self-aware computing infrastructures”. In: *Journal of Systems and Software* 147 (2019), pp. 1–16. DOI: https://doi.org/10.1016/j.jss.2018.09.089.

[239] Johannes Josef Stammel. “Architekturbasierte Bewertung und Planung von Änderungsanfragen”. PhD thesis. Karlsruhe Institute of Technology (KIT), 2015. DOI: 10.5445/IR/1000053953.

[240] Christian Stier. “Adaptation-Aware Architecture Modeling and Analysis of Energy Efficiency for Software Systems”. PhD thesis. Karlsruhe Institute of Technology (KIT), 2018. DOI: 10.5445/IR/1000083402.

[241] Misha Strittmatter. "A Reference Structure for Modular Metamodels of Quality-Describing Domain-Specific Modeling Languages". PhD thesis. Karlsruhe Institute of Technology (KIT), 2020. DOI: 10.5445/KSP/1000098906.

[242] Misha Strittmatter et al. "Challenges in the Evolution of Metamodels: Smells and Anti-Patterns of a Historically-Grown Metamodel". In: *10th International Workshop on Models and Evolution*. CEUR Workshop Proceedings Vol-1706. 2016, pp. 30–39.

[243] Daniel Strüber, Matthias Selter, and Gabriele Taentzer. "Tool Support for Clustering Large Meta-Models". In: *Workshop on Scalability in Model Driven Engineering*. ACM, 2013. ISBN: 9781450321655. DOI: 10.1145/2487766.2487773.

[244] Daniel Strüber et al. "Splitting Models Using Information Retrieval and Model Crawling Techniques". In: *17th International Conference on Fundamental Approaches to Software Engineering*. Springer, 2014, pp. 47–62. ISBN: 9783642548031. DOI: 10.1007/978-3-642-54804-8_4.

[245] Sagar Sunkle, Vinay Kulkarni, and Suman Roychoudhury. "Analyzing Enterprise Models Using Enterprise Architecture-Based Ontology". In: *ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems*. Springer, 2013, pp. 622–638. ISBN: 978-3-642-41532-6. DOI: 10.1007/978-3-642-41533-3_38.

[246] Carolyn Talcott et al. "Composition of Languages, Models, and Analyses". In: *Composing Model-Based Analysis Tools*. Ed. by Robert Heinrich et al. Springer, 2021, pp. 45–70. ISBN: 978-3-030-81915-6. DOI: 10.1007/978-3-030-81915-6_11.

[247] Emre Taspolatoglu and Robert Heinrich. "Context-based Architectural Security Analysis". In: *13th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 2016, pp. 281–282. DOI: 10.1109/WICSA.2016.55.

[248] Richard N Taylor, Nenad Medvidovic, and Eric M Dashofy. *Software architecture: foundations, theory, and practice*. Wiley Publishing, 2009. ISBN: 0470167742.

[249] Yong Meng Teo and Claudia Szabo. "CODES: An integrated approach to composable modeling and simulation". In: *41st Annual Simulation Symposium*. IEEE, 2008, pp. 103–110. DOI: 10.1109/ANSS-41.2008.24.

[250] K.C. Thramboulidis. "Using UML in control and automation: a model driven approach". In: *2nd IEEE International Conference on Industrial Informatics*. IEEE, 2004, pp. 587–593. DOI: 10.1109/INDIN.2004.1417414.

[251] Kleanthis Thramboulidis. "The 3+1 SysML View-Model in Model Integrated Mechatronics". In: *Journal of Software Engineering and Applications* 3.2 (2010), pp. 109–118. DOI: 10.4236/jsea.2010.32014.

[252] Frank Tip. "A Survey of Program Slicing Techniques". In: *Journal of Programming Languages* 3 (1995), pp. 121–189.

[253] K. Tuma, R. Scandariato, and M. Balliu. "Flaws in Flows: Unveiling Design Flaws via Information Flow Analysis". In: *IEEE 16th International Conference on Software Architecture*. IEEE, 2019, pp. 191–200. DOI: 10.1109/ICSA.2019.00028.

[254] Katja Tuma et al. "Automating the Early Detection of Security Design Flaws". In: *23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. ACM, 2020, pp. 332–342. ISBN: 9781450370196. DOI: 10.1145/3365438.3410954.

[255] Fatih Turkmen et al. "Analysis of XACML Policies with SMT". In: *4th International Conference on Principles of Security and Trust*. Springer, 2015, pp. 115–134. ISBN: 9783662466650. DOI: 10.1007/978-3-662-46666-7_7.

[256] I.T.P. Vanderfeesten et al. "Quality metrics for business process models". In: *2007 BPM and workflow handbook*. Ed. by L. Fischer. Future Strategies, 2007, pp. 179–190. ISBN: 978-0-9777527-1-3.

[257] Valeria Vittorini et al. "The OsMoSys approach to multi-formalism modeling of systems". In: *Software & System Modeling* 3 (2004), pp. 68–81. DOI: 10.1007/s10270-003-0039-5.

[258] Thomas Vogel and Holger Giese. "Adaptation and Abstract Runtime Models". In: *2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2010, pp. 39–48. ISBN: 978-1-60558-971-8. DOI: 10.1145/1808984.1808989.

[259] B. Vogel-Heuser et al. *Researching Evolution in Industrial Plant Automation: Scenarios and Documentation of the Pick and Place Unit. Report No. TUM-AIS-TR-01-14-02*. Tech. rep. 2014. URL: https://mediatum.ub.tum.de/doc/1208973/1208973.pdf.

[260] B. Vogel-Heuser et al. “Towards a common classification of changes for information and automated production systems as precondition for maintenance effort estimation”. In: *2016 IEEE 14th International Conference on Industrial Informatics*. IEEE, 2016, pp. 166–172. DOI: 10.1109/INDIN.2016.7819152.

[261] Birgit Vogel-Heuser. “Usability Experiments to Evaluate UML/SysML-Based Model Driven Software Engineering Notations for Logic Control in Manufacturing Automation”. In: *Journal of Software Engineering and Applications* 7.11 (2014), pp. 943–973. DOI: 10.4236/jsea.2014.711084.

[262] Birgit Vogel-Heuser et al. “Evolution of software in automated production systems: Challenges and research directions”. In: *Journal of Systems and Software* 110 (2015), pp. 54–84. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2015.08.026.

[263] Christian Vögele et al. “Modeling Complex User Behavior with the Palladio Component Model”. In: *Softwaretechnik-Trends* 35.3 (2015). ISSN: 0720-8928.

[264] Christian Vögele et al. “WESSBAS: extraction of probabilistic workload specifications for load testing and performance prediction—a model-driven approach for session-based application systems”. In: *Software & Systems Modeling* 17.2 (2018), pp. 443–477. ISSN: 1619-1374. DOI: 10.1007/s10270-016-0566-5.

[265] Markus Völter. “Language and IDE Modularization, Extension and Composition with MPS”. In: *2011 International Summer School on Generative and Transformational Techniques in Software Engineering*. Vol. 7680, Lecture Notes in Computer Science. Springer, 2011, pp. 383–430. ISBN: 978-3-642-35991-0. DOI: 10.1007/978-3-642-35992-7_11.

[266] Markus Völter et al. “Mbeddr: An extensible C-based programming language and IDE for embedded systems”. In: *3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*. ACM, 2012, pp. 121–140. DOI: 10.1145/2384716.2384767.

[267] Michael Waidner and Michael Kasper. “Security in industrie 4.0 - challenges and solutions for the fourth industrial revolution”. In: *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1303–1308. DOI: 10.3850/9783981537079_1005.

[268] Jürgen Walter et al. “An Expandable Extraction Framework for Architectural Performance Models”. In: *8th ACM/SPEC International Conference on Performance Engineering Companion*. ACM, 2017, pp. 165–170. DOI: 10.1145/3053600.3053634.

[269] Maximilian Walter, Robert Heinrich, and Ralf Reussner. “Architectural Attack Propagation Analysis for Identifying Confidentiality Issues”. In: *19th IEEE International Conference on Software Architecture*. IEEE, 2022, pp. 1–12. DOI: 10.1109/ICSA53651.2022.00009.

[270] Maximilian Walter et al. “Architectural Optimization for Confidentiality under Structural Uncertainty”. In: *15th European Conference on Software Architecture Post-Proceedings*. Springer, 2022.

[271] B.C. Warboys, R.M. Greenwood, and P. Kawalek. “Modelling the Co-Evolution of Business Processes and IT Systems”. In: *Systems Engineering for Business Process Change: Collected Papers from the EPSRC Research Programme*. Springer, 2000, pp. 10–23. ISBN: 978-1-4471-1146-7. DOI: 10.1007/978-1-4471-0457-5_2.

[272] Matthias Weidlich, Mathias Weske, and Jan Mendling. “Change Propagation in Process Models Using Behavioural Profiles”. In: *2009 IEEE International Conference on Services Computing*. IEEE, 2009, pp. 33–40. DOI: 10.1109/SCC.2009.58.

[273] Monika Weidmann et al. “Business process change management based on process model synchronization of multiple abstraction levels”. In: *2011 IEEE International Conference on Service-Oriented Computing and Applications*. IEEE, 2011, pp. 1–4. DOI: 10.1109/SOCA.2011.6166253.

[274] Herb Weisbaum. *Trust in Facebook has dropped by 66 percent since the Cambridge Analytica scandal*. accessed 2021-08-20. Apr. 2018. URL: https://web.archive.org/web/20210820004535/https://www.nbcnews.com/business/consumer/trust-facebook-has-dropped-51-percent-cambridge-analytica-scandal-n867011.

[275] Mark Weiser. “Program Slicing”. In: *5th International Conference on Software Engineering*. IEEE, 1981, pp. 439–449. ISBN: 0897911466.

[276] Christian Wende, Nils Thieme, and Steffen Zschaler. “A Role-Based Approach towards Modular Language Engineering”. In: *2nd International Conference on Software Language Engineering*. Vol. 5969, Lecture Notes in Computer Science. Springer, 2009, pp. 254–273. ISBN: 978-3-642-12106-7. DOI: 10.1007/978-3-642-12107-4_19.

[277] Alexander Wert, Henning Schulz, and Christoph Heger. "AIM: Adaptable Instrumentation and Monitoring for Automated Software Performance Analysis". In: *2015 IEEE/ACM 10th International Workshop on Automation of Software Test*. IEEE, 2015, pp. 38–42. ISBN: 978-1-4673-7022-6. DOI: 10.1109/AST.2015.15.

[278] Felix Willnecker et al. "Comparing the accuracy of resource demand measurement and estimation techniques". In: *European Workshop on Performance Engineering*. Vol. 9272, Lecture Notes in Computer Science. Springer. 2015, pp. 115–129. DOI: 10.1007/978-3-319-23267-6_8.

[279] Katharina Wolter, Thorsten Krebs, and Lothar Hotz. "Ontology-based Model Comparison". In: *Softwaretechnik-Trends* 27.2 (2007). ISSN: 0720-8928.

[280] Murray Woodside, Greg Franks, and Dorina C. Petriu. "The Future of Software Performance Engineering". In: *2007 Future of Software Engineering*. IEEE, 2007, pp. 171–187. ISBN: 0769528295. DOI: 10.1109/FOSE.2007.32.

[281] Workflow Management Coalition. *The Workflow Management Coalition Specification – Terminology & Glossary (WFMC-TC-1011)*. 1999.

[282] Y. Brun et al. "Engineering Self-Adaptive Systems Through Feedback Loops". In: *Software Engineering for Self-Adaptive Systems*. Vol. 5525, Lecture Notes in Computer Science. Springer, 2009, pp. 48–70. DOI: 10.1007/978-3-642-02161-9_3.

[283] Mark Yampolskiy et al. "Systematic analysis of cyber-attacks on CPS-evaluating applicability of DFD-based approach". In: *2012 5th International Symposium on Resilient Control Systems*. IEEE, 2012, pp. 55–62. DOI: 10.1109/ISRCS.2012.6309293.

[284] Sanghyun Yoo et al. "Rule-based Dynamic Business Process Modification and Adaptation". In: *2008 International Conference on Information Networking* (2008), pp. 1–5. DOI: 10.1109/ICOIN.2008.4472793.

[285] Bintao Yuan et al. "An Attack Path Generation Methods Based on Graph Database". In: *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference*. IEEE, 2020, pp. 1905–1910. DOI: 10.1109/ITNEC48623.2020.9085039.

[286] Bernard P. Zeigler, Alexandre Muzy, and Ernesto Kofman. *Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations*. 3rd edition. Academic Press, Inc., 2018. ISBN: 0128133708.

[287] Steffen Zschaler and Francisco Durán. "GTSMorpher: Safely Composing Behavioural Analyses Using Structured Operational Semantics". In: *Composing Model-Based Analysis Tools*. Ed. by Robert Heinrich et al. Springer, 2021, pp. 189–215. ISBN: 978-3-030-81915-6. DOI: 10.1007/978-3-030-81915-6_9.

The Karlsruhe Series on Software Design and Quality

ISSN 1867-0067

Band 1 Steffen Becker
Coupled Model Transformations for QoS Enabled Component-Based Software Design.
ISBN 978-3-86644-271-9

Band 2 Heiko Koziolek
Parameter Dependencies for Reusable Performance Specifications of Software Components.
ISBN 978-3-86644-272-6

Band 3 Jens Happe
Predicting Software Performance in Symmetric Multi-core and Multiprocessor Environments.
ISBN 978-3-86644-381-5

Band 4 Klaus Krogmann
Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis.
ISBN 978-3-86644-804-9

Band 5 Michael Kuperberg
Quantifying and Predicting the Influence of Execution Platform on Software Component Performance.
ISBN 978-3-86644-741-7

Band 6 Thomas Goldschmidt
View-Based Textual Modelling.
ISBN 978-3-86644-642-7

Band 7 Anne Koziolek
Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes.
ISBN 978-3-86644-973-2

Band 8 Lucia Happe
Configurable Software Performance Completions through Higher-Order Model Transformations.
ISBN 978-3-86644-990-9

Band 9 Franz Brosch
Integrated Software Architecture-Based Reliability Prediction for IT Systems.
ISBN 978-3-86644-859-9

Band 10 Christoph Rathfelder
Modelling Event-Based Interactions in Component-Based Architectures for Quantitative System Evaluation.
ISBN 978-3-86644-969-5

Band 11 Henning Groenda
Certifying Software Component Performance Specifications.
ISBN 978-3-7315-0080-3

Band 12 Dennis Westermann
Deriving Goal-oriented Performance Models by Systematic Experimentation.
ISBN 978-3-7315-0165-7

Band 13 Michael Hauck
Automated Experiments for Deriving Performance-relevant Properties of Software Execution Environments.
ISBN 978-3-7315-0138-1

Band 14 Zoya Durdik
Architectural Design Decision Documentation through Reuse of Design Patterns.
ISBN 978-3-7315-0292-0

Band 15 Erik Burger
Flexible Views for View-based Model-driven Development.
ISBN 978-3-7315-0276-0

Band 16 **Benjamin Klatt**
Consolidation of Customized Product Copies into Software Product Lines.
ISBN 978-3-7315-0368-2

Band 17 **Andreas Rentschler**
Model Transformation Languages with Modular Information Hiding.
ISBN 978-3-7315-0346-0

Band 18 **Omar-Qais Noorshams**
Modeling and Prediction of I/O Performance in Virtualized Environments.
ISBN 978-3-7315-0359-0

Band 19 **Johannes Josef Stammel**
Architekturbasierte Bewertung und Planung von Änderungsanfragen.
ISBN 978-3-7315-0524-2

Band 20 **Alexander Wert**
Performance Problem Diagnostics by Systematic Experimentation.
ISBN 978-3-7315-0677-5

Band 21 **Christoph Heger**
An Approach for Guiding Developers to Performance and Scalability Solutions.
ISBN 978-3-7315-0698-0

Band 22 **Fouad ben Nasr Omri**
Weighted Statistical Testing based on Active Learning and Formal Verification Techniques for Software Reliability Assessment.
ISBN 978-3-7315-0472-6

Band 23 **Michael Langhammer**
Automated Coevolution of Source Code and Software Architecture Models.
ISBN 978-3-7315-0783-3

Band 24 Max Emanuel Kramer
Specification Languages for Preserving Consistency between Models of Different Languages.
ISBN 978-3-7315-0784-0

Band 25 Sebastian Michael Lehrig
Efficiently Conducting Quality-of-Service Analyses by Templating Architectural Knowledge.
ISBN 978-3-7315-0756-7

Band 26 Georg Hinkel
Implicit Incremental Model Analyses and Transformations.
ISBN 978-3-7315-0763-5

Band 27 Christian Stier
Adaptation-Aware Architecture Modeling and Analysis of Energy Efficiency for Software Systems.
ISBN 978-3-7315-0851-9

Band 28 Lukas Märtin
Entwurfsoptimierung von selbst-adaptiven Wartungsmechanismen für software-intensive technische Systeme.
ISBN 978-3-7315-0852-6

Band 29 Axel Busch
Quality-driven Reuse of Model-based Software Architecture Elements.
ISBN 978-3-7315-0951-6

Band 30 Kiana Busch
An Architecture-based Approach for Change Impact Analysis of Software-intensive Systems.
ISBN 978-3-7315-0974-5

Band 31 Misha Strittmatter
A Reference Structure for Modular Metamodels of Quality-Describing Domain-Specific Modeling Languages.
ISBN 978-3-7315-0982-0

Band 32 **Markus Frank**
Model-Based Performance Prediction for Concurrent Software on Multicore Architectures. A Simulation-Based Approach.
ISBN 978-3-7315-1146-5

Band 33 **Manuel Gotin**
QoS-Based Optimization of Runtime Management of Sensing Cloud Applications.
ISBN 978-3-7315-1147-2

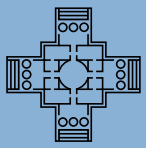
Band 34 **Heiko Klare**
Building Transformation Networks for Consistent Evolution of Interrelated Models.
ISBN 978-3-7315-1132-8

Band 35 **Roman Pilipchuk**
Architectural Alignment of Access Control Requirements Extracted from Business Processes.
ISBN 978-3-7315-1212-7

Band 36 **Stephan Seifermann**
Architectural Data Flow Analysis for Detecting Violations of Confidentiality Requirements.
ISBN 978-3-7315-1246-2

Band 37 **Sofia Ananieva**
Consistent View-Based Management of Variability in Space and Time.
ISBN 978-3-7315-1241-7

Band 38 **Robert Heinrich**
Architecture-based Evolution of Dependable Software-intensive Systems.
ISBN 978-3-7315-1294-3



The Karlsruhe Series on Software Design and Quality

Edited by Prof. Dr. Ralf Reussner

Ensuring dependability of increasingly heterogeneous and complex software-intensive systems requires to address the following challenges. There is (i) little support for reasoning about dependability in early development, (ii) little flexibility for evolving modelling languages and analysis techniques, and (iii) a gap of abstraction between data of the operation phase, architectural models and source code of the development phase which results in loss of architectural knowledge and deprecates models.

This work, proposes concepts for (i) modelling and analysing dependability based on architectural models of software-intensive systems early in development, (ii) decomposition and composition of modelling languages and analysis techniques to enable more flexibility in evolution, and (iii) bridging the divergent levels of abstraction between data of the operation phase, architectural models and source code of the development phase.

ISSN 1867-0067
ISBN 978-3-7315-1294-3

Gedruckt auf FSC-zertifiziertem Papier