Mälardalen University Licentiate Thesis 297

Advancing Continuous Model-Based Development in Industry

Robbert Jongeling

MÄLARDALEN UNIVERSITY
SWEDEN

Mälardalen University Press Licentiate Theses
No. 297

ADVANCING CONTINUOUS MODEL-BASED DEVELOPMENT IN INDUSTRY

Robbert Jongeling

2020

MÄLARDALEN UNIVERSITY
SWEDEN

School of Innovation, Design and Engineering

ISBN 978-91-7485-480-0
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

Abstract

For the development of complex software systems, two paradigms have become popular in industry: model-based development and Agile software development. In model-based development, models are the core development artifacts, particularly in early development phases such as specification and design. The short development cycles of Agile development, and in particular continuous integration, are sometimes seen as conflicting with the apparent longer development phases in model-based development. We study how software development can benefit from combining these two paradigms successfully into *continuous model-based development*.

In this licentiate thesis, we present four papers studying continuous model-based development of complex embedded systems in industry. The first two papers present investigations of the current state-of-the-art and state-of-practice of combining model-based development and continuous integration. In particular, specific challenges to the combination are identified. In the third and fourth papers, we focus on one of those challenges: model synchronization, i.e., the management of consistency between disparate development artifacts describing the same system or parts of it. We propose a lightweight approach that notifies developers of arisen inconsistency between different models. Lastly, we consider the aspect of variability among different development artifacts. In particular, we provide automated support for alleviating manual tasks in maintaining consistency across model variants organized in a product line.

Sammanfattning

Två populära paradigmer för utveckling av komplexa mjukvarusystem är modellbaserad utveckling och agil mjukvaruutveckling. I modellbaserad utveckling är modeller kärnartiklar för mjukvaruutveckling, speciellt för att uttrycka specifikation och design. De korta utvecklingscyklerna i agil utveckling, i synnerhet vid kontinuerlig integration, ses ibland som motstridiga med de längre utvecklingsfaserna i modellbaserad utveckling. Vi fokuserar på hur mjukvaruutveckling kan dra nytta av att de två paradigmerna framgångsrikt kan kombineras till kontinuerlig modellbaserad utveckling.

I denna licentiatavhandling presenterar vi fyra artiklar som studerar kontinuerlig modellbaserad utveckling av komplexa inbyggda system inom industrin. De två första artiklarna presenterar undersökningar av den aktuella situationen och specifika utmaningar för att kombinera modellbaserad utveckling och kontinuerlig integration. I den tredje och fjärde artikeln fokuserar vi på en av dessa utmaningar: modellsynkronisering, det vill säga hanteringen av konsistens mellan olika utvecklingsartefakter som beskriver samma system. Vi föreslår en metod som informerar utvecklare när inkonsistens mellan olika modeller introduceras. Slutligen undersöker vi variabilitet mellan olika utvecklingsartefakter och presenterar ett automatiskt stöd för att förenkla det manuella arbetet att upprätthålla konsistens mellan modellvarianter organiserade i en produktlinje.

Samenvatting

Voor de ontwikkeling van complexe softwaresystemen zijn twee praktijken breed omarmd: model-gebaseerde ontwikkeling en *Agile* softwareontwikkeling. In model-gebaseerde ontwikkeling staan modellen centraal, in het bijzonder in vroege ontwerpfases. De ontwikkelcycli in model-gebaseerde ontwikkeling worden vaak gezien als lang en stug, en daarmee conflicterend met de korte ontwikkelcycli in Agile methodes en nadrukkelijk *continuous integration*. We onderzoeken hoe software ontwikkeling kan profiteren van een combinatie van deze twee praktijken: *continue model-gebaseerde ontwikkeling*.

In deze licentiaat[1] thesis presenteren we vier wetenschappelijke artikelen die dit onderwerp onderzoeken in de context van ontwikkeling van complexe *embedded* softwaresystemen in bedrijven die machines ontwikkelen waarvan deze systemen deel uitmaken. De eerste twee artikelen presenteren onderzoek naar de huidige standaarden en uitdagingen, zowel in de literatuur als in de praktijk. In het derde en vierde artikel verleggen we de focus naar een van die uitdagingen: modelsynchronisatie. Daarmee wordt bedoeld, het ervoor zorgen dat verschillende artefacten (zoals modellen en programmacode), die worden ontwikkeld om een systeem te ontwerpen en te implementeren, elkaar niet tegenspreken. We stellen een “lichtgewicht” benadering voor, waarbinnen ontwikkelaars melding krijgen van ontstane tegenstrijdigheden tussen verschillende artefacten. Als laatste bijdrage in deze thesis ontwikkelen we automatische ondersteuning voor het onderhouden van consistentie tussen modellen in een productlijn, die varianten van een systeem beschrijven.

[1]Licentiaat is een Zweedse academische graad halverwege tussen een master (MSc) en een doctor (PhD).

Acknowledgment

I would like to take this opportunity to thank some of the people who were invaluable to the creation of this thesis. First of all, I would like to thank my advisors Jan Carlson, Antonio Cicchetti, and Federico Ciccozzi. Thank you for all your help during the last years; for bringing me to Sweden; for your valuable input in our research discussions; for all your patience whenever I came by your offices to talk; for the shared pearls of wisdom; for your fast and insightful feedback on drafts; and for your mentorship during conference trips (and on that note, I should also thank honorary advisor Alessio Bucaioni).

The work in this thesis was supported by Software Center. I would like to thank our contacts, interviewees, and other discussion partners in the companies we worked with, for their time and valuable input in research discussions.

I would also like to thank my IDT colleagues for many great times. I enjoyed our fika's with sweets from around the world, cinema visits, game nights, runs, and the many other activities outside work.

A few special mentions are in order. Thank you Filip for being the best office mate. Thank you Jean and Leo for accompanying me on many great adventures, disappointing or otherwise. And thank you my cycling buddy Václav for showing me many kilometers around Västerås and beyond.

Finally, but most importantly, I want to thank my family for their continued support. Especially, I want to thank my parents for everything they did for me throughout the years. *Pap en mam, bedankt voor meer dan alles.*

Robbert Jongeling
Västerås, October, 2020

List of Publications

Papers included in this thesis[2]

Paper A: Robbert Jongeling, Jan Carlson, Antonio Cicchetti, Federico Ciccozzi. *Continuous integration support in modeling tools.* In the 3rd International workshop on collaborative modelling in MDE (COMMitMDE) at the 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018).

Paper B: Robbert Jongeling, Jan Carlson, Antonio Cicchetti. *Impediments to Introducing Continuous Integration for Model-Based Development in Industry.* In the 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2019). [3]

Paper C: Robbert Jongeling, Federico Ciccozzi, Antonio Cicchetti, Jan Carlson. *Lightweight Consistency Checking for Agile Model-Based Development in Practice.* In the 15th European Conference on Modelling Foundations and Applications (ECMFA 2019)[4] and the Journal of Object Technology (JOT 2019).

Paper D: Robbert Jongeling, Antonio Cicchetti, Federico Ciccozzi, Jan Carlson. *Co-evolution of Simulink Models in a Model-Based Product Line.* In the 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS 2020).

[2]The included papers have been reformatted to comply with the thesis layout.

[3]This paper was chosen among the five best papers of the SEAA 2019 programme.

[4]This paper was awarded the Best Paper Award at the Applications Track of ECMFA 2019.

Related publications, not included in this thesis

Paper V: Robbert Jongeling. “*Considerations About Consistency Management for Industrial Model-Based Development.*” In the 12th Seminar on Advanced Techniques & Tools for Software Evolution (SATToSE 2019).

Paper W: Robbert Jongeling. “*How to live with inconsistencies in industrial model-based development practice.*” In the Doctoral Symposium (DS) at the 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS 2019).

Paper X: Robbert Jongeling, Johan Fredriksson, Federico Ciccozzi, Antonio Cicchetti, Jan Carlson. “*Towards Consistency Checking Between a System Model and its Implementation.*” In the 1st International Conference on Systems Modelling and Management (ICSMM 2020).

Paper Y: Muhammad Abbas, Robbert Jongeling, Claes Lindskog, Eduard Paul Enoiu, Mehrdad Saadatmand, Daniel Sundmark. “*Product Line Adoption in Industry: An Experience Report from the Railway Domain.*” In the 24th International Systems and Software Product Line Conference (SPLC 2020).

Paper Z: Robbert Jongeling, Antonio Cicchetti, Federico Ciccozzi, Jan Carlson. “*Towards boosting the OpenMBEE platform with model-code consistency*” In the 1st International Workshop on Open Model Based Engineering Environment (OpenMBEE) at the 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS 2020).

Contents

I

Thesis

Chapter 1

Introduction

In the development of modern embedded systems, most innovation comes from software, leading to expressions like "this car runs on code" [1]. Hence, there has been a lot of work aiming at improving the productivity of software development and the quality of the resulting artifacts. We discuss two of the most prominent paradigms that have been widely adopted to achieve those gains: Model-Based Development (MBD) and Agile software development [2].

In MBD, models are used for the design of systems, and possibly for their implementation too [3]. Within system design, it is beneficial to abstract some of the implementation details away in favor of a more human-oriented view of structure and functionality. Models can be used at all stages of the development and for different purposes, from communication to the automatic generation of code. In this work, we use the term MBD to refer to practices in which models are used as core software development artifacts, meaning that the models are expected to undergo frequent changes and the resulting implementation is expected to be consistent with these models. We exclude from our scope those MBD practices in which models are created for temporary documentation or communication between stakeholders only. Note that this scope is thus much wider than the UML-specific focus of the Agile Modeling (AM) paradigm described by Ambler [4].

Since the publication of the Agile manifesto [2], software development has increasingly focused on shortening development cycles. Ideally, customers are regularly presented with an enhanced implementation, allowing them to adjust the requirements and thereby the product to their needs. Important among the Agile practices is Continuous Integration (CI) [5], in which multiple developers collaborate on a software project and each of them integrates her work frequently into a shared repository. In the CI paradigm, an integration is followed by an automated build as well as automated execution of a test suite, giving the developers an up-to-date overview of the status of a project throughout the development. This allows early detection of errors and prevents a difficult integration period of uncertain length at the end of the project.

Benefits to the productivity of software development in industry have been reported for both CI [6] and MBD [7], separately. Yet, the application of both practices in combination in industrial development projects is sometimes met with skepticism [8]. We hypothesize that combining MBD and CI into *continuous MBD* can improve the productivity of software development. In this work, we study the state-of-the-art, and the state-of-practice in several industrial environments, to identify challenges to this combination. Thereafter, we provide approaches to alleviate tasks that currently involve a large manual effort and are thereby impeding the introduction of short development cycles.

In particular, we focus on tasks related to model synchronization, i.e., ensuring consistency across various development artifacts, which is a general challenge to MBD as well [9]. In a development setup where all artifacts are code, a build system typically notices inconsistent definitions between different portions of code. For example, the code will not compile when a class does not implement all the methods defined in an interface. In MBD, it often occurs that no formal links exist between artifacts, and consequently, these types of inconsistencies might go unnoticed for a long time. The ultimate consequence of this might be late changes to the implementation or even an incorrect implementation. When moving to Agile development, with its shorter development cycles and aim of continuously integrating the development artifacts, keeping them consistent becomes both more important and more challenging. Different artifacts sooner rely on each other, and hence, inconsistencies may be propagated faster across artifacts and can be more difficult to resolve. One way to address this challenge is by frequently checking the consistency across artifacts, through automated checking mechanisms.

Many approaches have been presented to deal with inconsistency across models. Existing approaches aim to satisfy different sets of requirements in various settings. We study the problem in two industrial settings. From the first of these, we obtain requirements for a consistency checking approach that can support continuous MBD. This has resulted in an approach that is less expressive and can not automatically resolve inconsistencies, but requires a lower effort than existing approaches for defining and maintaining consistency checks.

In the above introduction of consistency checking, we have focused on the consistency across different development artifacts that describe the same system, possibly at different levels of abstraction. An additional dimension of the problem emerges when the MBD setting also includes different variants of the to-be-developed system. Typically, variability is managed using software product lines [10], in which the development is based on structured reuse of development artifacts across different product variants. To preserve the integrity of the product line and the opportunities for reuse, those artifacts should be kept consistent with each other. We study this aspect in a second setting, a model-based software product line in which changes must be propagated between models describing various derived products. As with the previously discussed type of consistency checking, reducing the manual effort for this task is vital to allow for shorter development cycles and eventually, continuous MBD.

Thesis outline This licentiate thesis contains two parts. Part I is an overview of the thesis and is organized as follows. We first discuss our research process and introduce research goals in Chapter 2, after which background and related work to the thesis are discussed in Chapter 3. In Chapter 4, we provide an overview of the included papers and the contributions brought by each of them. In Chapter 5, we present conclusions and an outline of future work towards a doctoral thesis. Part II includes the collection of included papers.

Chapter 2

Research Overview

In this chapter, we introduce the overall research goals of the thesis and how we used both empirical and constructive research methods to achieve them.

2.1 Research Goals

The essence of MBD is to abstract from the implementation by capturing the problem space in models [3]. We consider MBD to refer to development practices in which models are created and maintained as core development artifacts. That is, we require models to explicitly play a central role during development and we require the implementation to conform to it.

Nevertheless, we include in our scope a broad range of MBD practices, since not all artifacts have to be models. We also consider those development settings in which graphical models play a smaller role, e.g. because textual code is written manually. For "models", we refer to system design, software design, or software implementation models. These models can be expressed in various modeling languages. Models may provide support for, e.g. automated analysis, simulation, or code generation. Furthermore, different models can be used to describe systems at different levels of abstraction. For example, a company may use MBD to capture the system requirements and structure in a SysML [11] model, whereas individual features are implemented in Simulink [12] models or code.

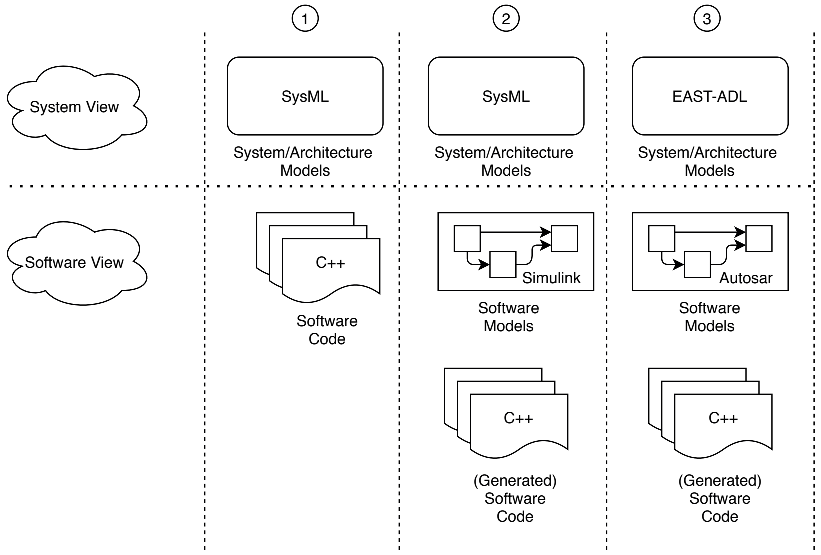
1
2
3
System View
Software View
SysML
SysML
EAST-ADL
System/Architecture Models
System/Architecture Models
System/Architecture Models
C++
Software Code
Simulink
Software Models
Autosar
Software Models
C++
(Generated) Software Code
C++
(Generated) Software Code

Figure 2.1. Three examples of possible artifacts in different model-based development projects.

More examples of MBD settings are illustrated in Figure 2.1. In the figure, we show three settings in which different models are used for the system view and the software view. In setting ①, the high-level system design is captured in a SysML model; this model does not contain any implementation details, but rather outlines the structure of the software as it is divided into components. Furthermore, the system model deals with concerns on dividing functionality across software components and hardware components. Hence, in this setting, code is not automatically generated from the model. In setting ②, we see that part of the implementation is automatically generated, from Simulink software models. The overall system design is again done in a SysML model and, additionally, Simulink models are created to design specific Software components. Among these example settings, there is a common need for the artifacts to be consistent with each other. Tool choices can vary across different settings too, as is illustrated by setting ③.

MBD promises to improve the productivity of software development [3].

Indeed, among the reported benefits is a reduced total development time of software systems [13]. However, MBD is traditionally viewed as "waterfall-like", with long development cycles and formal checks between each step in the cycle. After the publication of the Agile manifesto [2] and the more recent popularity gained by *DevOps*, short development cycles have become the norm in software development. One of the first steps in the DevOps paradigm and also one of the development practices promoted in Agile software development is CI. Our focus on CI is motivated by the state-of-practice at industrial partners. Due to the stringent safety requirements of their developed embedded systems, companies typically do not continuously deliver, let alone continuously deploy their software.

To illustrate continuous MBD and the CI activities on top of the existing MBD activities, consider setting ② from Figure 2.1. In the CI paradigm, the SysML model is subject to frequent changes. As a consequence, also the software models, in this example Simulink models, are subject to changes, following the updated system model. Then, also the code is generated again, following the changes to the Simulink models. As with code, also modeling artifacts should be integrated into a shared repository. The automation facilities of the CI pipeline can then be utilized to provide insight into the developed artifacts, for example through simulating the models or automatically analyzing them. The intended result of continuously integrating models too is to accelerate the feedback loop to developers, for example by allowing frequent inspection of the adherence of the code to the intended design.

To summarize, both the MBD and CI paradigms separately give improved productivity in software development; and their combination can yield additional benefits. Therefore, we propose to enhance existing MBD practices with CI features. To this end, we formulated our first research goal as follows:

RG$_1$: To identify impediments towards the adoption of continuous integration in model-based development.

Towards achieving this research goal, we first identified the state-of-the-art and state-of-practice of the combination of MBD and CI. In particular, we investigated existing MBD practices in the development settings at our industrial partners and identified improvement opportunities for them. As a result, we found the need to automate some of the labor-intensive manual tasks in MBD, so that more frequent development iterations could be established. This resulted in the second research goal, which instead aimed at identifying actions that are currently performed manually and that would need to be at least partially automated to eventually make more frequent development iterations possible and beneficial. Our second research goal was:

RG$_2$: To alleviate labor-intensive manual tasks that impede the adoption of short development iterations in industrial MBD settings.

2.2 Research Methodology

An old critique of software engineering research in a new guise states that "most software engineering research has the same effect on programmers that astronomy has on stars" [14]. The research community recognizes the limited practical relevance of software engineering research and suggests industry-academia collaborations as one of the means to improve it [15]. We performed our research in close collaboration with industrial partners through Software Center[1], an organization featuring 5 Swedish universities and 15 companies collaborating in software engineering research projects. The research presented in this thesis is the result of collaborations with three of the Software Center member companies, as well as two external companies.

Our research was performed in 6-month "sprints", following the commonly recommended best practice for industry-academia research collaborations of organizing the research in iterations so that research topics can frequently be fine-tuned to maximize their relevance [16]. Each of these sprints was started with a research proposal agreed upon with the partner companies. At the end of each sprint, research directions for the next sprint were proposed and results were presented in a joint workshop open to all companies in Software Center.

The iterative nature of our research process is closely related to the well-known constructive research methodology [17]. This methodology describes the common practice in software engineering research of creating knowledge by constructing solutions to well-defined problems. To identify a well-defined problem, constructive research is often preceded by empirical studies investigating

[1]https://www.software-center.se/

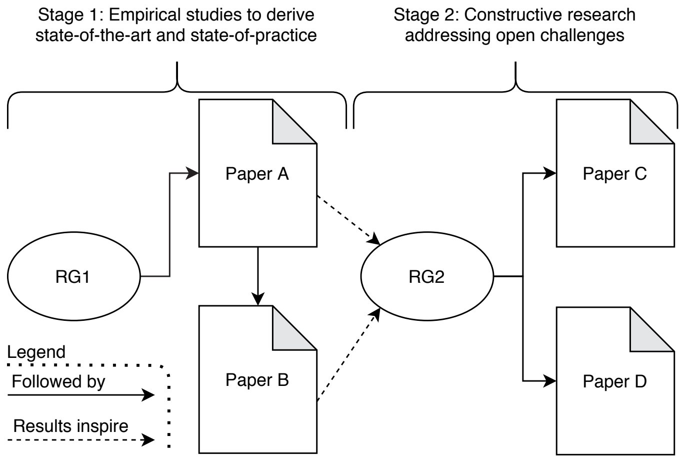
Stage 1: Empirical studies to derive state-of-the-art and state-of-practice
Stage 2: Constructive research addressing open challenges
Paper A
Paper B
Paper C
Paper D
RG1
RG2
Legend
Followed by
Results inspire

Figure 2.2. Overview of our research process

the state-of-the-art and the state-of-practice [17].

In this thesis, we present two papers (Paper A and Paper B) presenting empirical studies reporting on the state-of-the-art and state-of-practice of continuous MBD, addressing RG_1. The other two papers (Paper C and Paper D), present the construction of new approaches in collaboration with our industrial partners, addressing RG_2. Organizing the research in sprints has allowed us to use the results from Paper A and Paper B for refining RG_2 and for creating specific research projects for Paper C and Paper D. Notably, the findings of Paper A and Paper B show a lack of automated support for model synchronization, impact analysis, and co-evolution. These results have then inspired the work leading to Paper C, which targets support for consistency checking in a continuous MBD setting, and Paper D, which targets impact analysis and co-evolution in another industrial MBD setting. Figure 2.2 summarizes our research process in terms of the contributions and their interdependencies.

The aforementioned process describes the relation between our research goals and contributions. While the work addressing both research goals involved industrial partners, RG_1 aimed at identifying general research problems in the area, whereas RG_2 aimed at proposing an approach in specific industrial settings.

Therefore, in Paper A and Paper B, we used empirical methods [18] to capture the state-of-practice and identify the requirements for a new approach.

In Paper A, we describe the state-of-the-art by comparing the most-used COTS modeling tools in industry. Existing state-of-the-art literature reviews on the combination of Agile development and MBD, e.g. [19], tend to agree on high-level challenges such as tool immaturity and steep learning curve of MBD. We complement this knowledge by providing insights into available tooling, the features they support, and their shortcomings. For similar reasons, in Paper B, we conducted an interview study with practitioners. Our main objective was to identify diverse states of practice and identify open research challenges related to them. To obtain insights complementing existing literature, we involved practitioners from our partner companies. We performed semi-structured interviews [20] to provide interviewees the opportunity for personal input, while still ensuring to discuss a set of pre-defined topics.

To address RG_2, we further extended the empirical work using constructive research, in which we proposed approaches within concrete industrial settings. In this way, the collaboration differed from that in Paper B, which included interviews with engineers from several companies. Both the collaborations for Paper C and Paper D started by defining research goals for which the results of Paper A and Paper B were used as input. Upon the definition of research goals, the collaborations continued with several iterations of proposals for an approach to address the goals. Once a promising way forward had been identified, we then implemented the approach and lastly validated it.

In Paper C, we developed an approach for making developers aware of inconsistencies between models. We presented a prototype implementation and an evaluation of the approach on a limited use-case. Our evaluation indicated new requirements for a lightweight consistency checking approach. The results represent the first step towards an industry-level approach.

In Paper D, we worked in a setting in which variants of products are developed using models. We developed an approach for propagating changes from the product line to derived products. The study considered a model-based product line setting in industry, thereby making it a different setting than studied in Paper C. Hence, Paper C and Paper D are denoted in Figure 2.2 as independent papers, both originating from RG_2.

Chapter 3

Background and Related Work

This chapter contains background information and related work to the work presented in this thesis.

3.1 Model-Based Development

Several names and corresponding acronyms are in use to describe the notion of using models as key software development artifacts. Common ones include model-driven engineering (MDE), model-based development (MBD), and model-based software engineering. In our work, we refer to MBD, to emphasize that models are core development artifacts but the development includes also other artifacts such as textual documentation or code.

Figure 3.1 illustrates the four layers of the modeling stack as originating from the object management group (OMG) core specification [21]. The bottom layer (M_0, object layer) represents the real world, each of the three layers above it represents an abstraction of the layer below that layer. The first layer above the bottom, layer M_1, contains models of the real world. At this point, it should be noted that the real-world layer can also house artifacts such as code, so M_1 could contain e.g. a UML class diagram as an abstraction of some implementation. M_2, contains so-called meta-models, which denote the type of constructs that can be used to express models in M_1. UML itself is an example of a metamodel. M_3, provides meta-metamodels, the final abstraction layer since a meta-metamodel

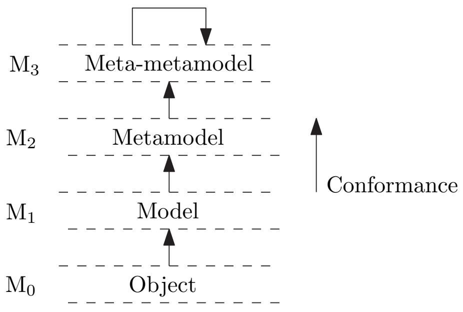

Figure 3.1. Layers of the modeling stack

describes not only instances at M_2 but also itself. Examples of meta-metamodels include Meta Object Facility (MOF)[1] and Ecore[2]. Our work is mostly concerned with instances at M_1 level and instances at M_0, but must take into account also M_2, since different instances of M_1 might conform to different instances of M_2.

Model-to-model transformations can be created to convert models conforming to one metamodel into models conforming to another metamodel. For this purpose, specialized model transformation languages have been developed, such as ATL[3] and QVTo[4]. Also, model-to-text transformations can be created, e.g. to generate code from models. Some modeling tools include such transformations and thereby support for code generation from their models.

3.1.1 Model-based systems engineering

In model-based systems engineering (MBSE), a diagrammatic *system model* is used as the central artifact containing architecture and design, thereby replacing textual documentation. The best-known language supporting this paradigm is the Systems Modeling Language (SysML), which is an extended subset of UML [11]. SysML provides a modeler with several diagrams to describe the requirements, structure, and behavior of a system [22]. Although these diagrams

[1]https://www.omg.org/mof/
[2]https://wiki.eclipse.org/Ecore
[3]https://www.eclipse.org/atl/
[4]https://projects.eclipse.org/projects/modeling.mmt.qvt-oml

may still be complemented with textual descriptions, the idea of MBSE is that the system model forms the central development artifact. This ideally enables the automatic generation of source code and documentation from the system model. In practice, automatic generation of code is not always done because it requires the model to be completed down to a very low level of abstraction, i.e., to contain a great amount of detail. Among our industrial partners, we have encountered MBSE practices in which the system model is rather a guide for the manual development of code from it. Nevertheless, also in these practices, the eventual implementation is required to be consistent with the system model, in the sense that these two descriptions of the system should not contradict each other.

3.1.2 Model-based product lines

When developing software systems, companies may need to express different versions of that system that vary on certain points. To manage this type of variability, software product line engineering (SPLE) prescribes an organization of development artifacts in product lines [10]. Various structured ways of establishing product lines are known in the literature. On the other hand, clone-and-own is an unstructured practice in which reuse is initially organized through copy-and-paste [23].

We refer to software product lines in which models are central development artifacts as model-based product lines. In general, software product lines are organized as one central development "line" from which product variants can be derived. Changes in the main product line may need to be propagated to those derived variants, for example in case of bug-fixes. The changes that need to be propagated are typically smaller than complete files. When the development artifacts are text-based, files can in most cases be merged to achieve the propagation. In model-based product lines, diagrammatic models may need to be merged, which is notoriously challenging. Moreover, the localization of the part of the model that requires propagation is not straightforward. In Paper D, we study this problem in an industrial setting.

3.1.3 Adoption of MBD in industry

Towards achieving our first research goal (RG_1), we identified challenges in combining MBD and CI. Some of the resulting challenges are shared with general challenges to the introduction of modeling practices in industry, which are reported plentiful in the literature (e.g. [24]). For example, tool interoperability, tool usability, and a steep learning curve are usual suspects among reported challenges to the industrial adoption of modeling. Furthermore, challenges are not limited to tooling issues, also human factors must be considered [25]. Despite being well-known for years, these remain open research challenges [9].

Our results offer a new perspective on these known challenges. We study those settings in which MBD has already been introduced and propose ways to make them more continuous. We expect the different perspectives to yield complementary findings and thus we also expect our results to improve the adoption of MBD in industry.

3.2 Agile software development

The manifesto for Agile software development [2] aims for customer satisfaction through frequent delivery of working software. The main effect of adopting the practices outlined in the manifesto is that development cycles become shorter, thereby allowing for frequent course adjustments. A fundamental ingredient for achieving this is Continuous Integration (CI) [5]. In the “stairway to heaven” model, CI is the third step on the evolution path from traditional engineering to continuous deployment [26].

3.2.1 Continuous Integration

In Paper A, we define CI as: “a collaborative development practice where software engineers frequently, at least daily, integrate their work into a shared repository.” Besides enabling frequent deliveries of the software to clients, CI also prevents the need for a complex integration period after the implementation of all parts, which can be hard to plan for and take exceedingly long to complete. As can be seen from the definition, CI is concerned only with the development of software. The next stage is then to frequently release versions of the software (continuous delivery). Continuous delivery in turn can be followed by frequently deploying the releases on customer devices (continuous deployment). Figure 3.2 illustrates these phases. Further extensions of the continuous development paradigm are made in DevOps (Development and Operations), in which data of the usage of the deployed software is used as input for new development iterations [27]. In this thesis, we focus only on continuous integration, not the subsequent stages.

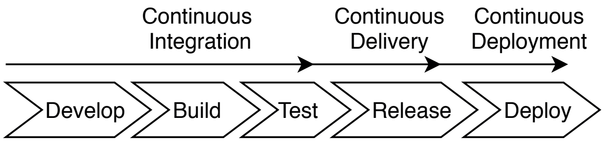

Figure 3.2. Stages of continuous development.

3.2.2 Agile model-driven development

Under the term Agile model-driven development (AMDD), several authors have presented work towards introducing Agile practices while using models as core development artifacts. Zhang et al. [28] have presented benefits of combining the two paradigms from experiences at Motorola. They present how their development processes were set up to allow for shorter development cycles, continuous integration, and frequent testing. Other case studies also find the potential benefits of applying Agile practices in MBD [29]. Rumpe has presented research results on Agile model-based software engineering using (executable) UML, presenting challenges such as model management, model composition, refactoring, and model quality [30]. Lano et al. [31] also advise a process to follow when combining Agile and modeling practices, among their tips are to do regular integration and testing. Another case study shows a successful adoption of Agile MBD and highlights the close coupling of software development with physical systems as a challenging aspect [32]. The authors address that challenge by using plant models to enable a virtual test environment, rather than relying on sparsely available physical systems, thereby contributing to shortening development increments in their model-based development. Given these experiences, there seems to be support for our hypothesis that MBD and Agile in combination can improve development productivity. This notion is supported by the MDE research community, which has identified making modeling more Agile as one of the current research challenges [9].

We limit our focus to continuous integration (CI), one of the key practices in Agile development. Some recent work has been published towards methods and processes enabling the combination of MBD and CI. The most important of those works consider more involved modeling practices, in which models are the only development artifacts and code is generated from them. Hence, the problems they identify are closely related to that way of working. Gatcía-Díaz et al. identify model versioning and incremental artifact generation as two problems in combining modeling and CI [33]. Considering a similar level of involvement of models, Garcia and Cabot propose to utilize the continuous delivery pipeline to deal with the co-evolution of models, metamodels, and model transformations [34]. The authors propose to chain existing activities and tools using the automation capabilities of Jenkins[5].

3.3 Consistency Management

The detection and resolution of inconsistencies within or between different diagrams of the same model (intra-model consistency checking) or between different models (inter-model consistency checking) have been studied extensively. In this research, we focus on inter-model consistency checking. In particular, in Paper C, we consider consistency between different views of a system, captured in different models that are potentially created using different modeling languages and in different tools. In Paper D, we consider consistency between different models describing system variants.

3.3.1 Relevance of consistency checking

The importance of consistency in the development process is undisputed [35] but despite the considerable amount of work on model synchronization, it is still considered an obstacle to industrial adoption of MBD [36]. Industrial evaluations of multi-view modeling and its consistency problems are lacking [37], perhaps

[5]https://jenkins.io/

because of the complexity and scale of those environments. Selic identifies the scale of industrial applications as one of the main challenges to overcome for a model synchronization approach to be applicable [38]. In particular, he argues that in many cases the number of consistency links is huge, resulting in a large maintenance effort that is at constant risk of being neglected in favor of more pressing issues [38]. Another often identified challenge is a lack of tool interoperability in MBD [24], which naturally complicates the type of consistency management we are interested in. Indeed, creating traceability links between different models is required for effective tool interoperability and consistency between models [39].

Several attempts have been made to define consistency. Some of them tried to mathematically define [36] or create an ontology of possible inconsistencies [40, 41]. To arrive at a common definition, we state that views that express overlapping concerns are inconsistent when they contradict each other [42].

3.3.2 Consistency checking approaches

We now discuss several categories of existing consistency checking approaches and reflect on the existence of so many approaches while at the same time, many new approaches are still proposed.

Instant Model Synchronization

A significant amount of work has been done on approaches that promise the automatic maintenance of consistency between views. Approaches based on Triple Graph Grammars (TGGs) [43] or Single-Underlying Model (SUM) [44] establish a bidirectional transformation between different diagrams, thereby ensuring instant propagation of changes across different views of the system. Furthermore, there are many other proposed mechanisms for model synchronization, such as keeping models synchronized given a synchronized situation and traceability links [45], or automatic bidirectional synchronization derived from a one-directional model transformation [46]. Also, a hybrid approach is proposed, in which model transformations are generated for change propagation between views based on model difference, based on a common underlying meta-model for all views [47].

In our studied industrial settings, these model synchronization mechanisms are typically not applicable. The first possible obstacle is the aforementioned difference in detail captured in different models, which is particularly apparent in the described case of keeping a system model and code consistent. Furthermore, high-level system models are typically “modeling the future”, i.e. the high-level models aim to describe the final product, whereas the code always represents the latest state of development. Therefore, the code is not expected to always conform to the latest version of the model. That also means that we don’t want to automatically propagate (or at least not yet) changes in the high-level model. Another key reason is that we need to support the iterative and flexible development process in industry. Changes are not always definitive or fully completed. Some developers may include temporary placeholder snippets in models or code that are known to be inconsistent with other artifacts but will be resolved in later stages. In such cases, it makes no sense to try to synchronize the models, but it does make sense to make developers aware of the introduced inconsistency so that it is not forgotten about. Also, there may be artifacts we do not control, because they are third-party, open-source, or re-used from other projects. Specifically, for the SUM approach, the consequence is that some views are “read-only”, i.e., they can not be changed. But furthermore, this reading may not be trivial at all, because the view could be expressed in any modeling or programming language. For these reasons, we try to formulate an approach not aimed at completely synchronizing models, but rather at an approach that allows inconsistencies but notifies the engineers when they are introduced. This follows established inconsistency tolerance ideas, which state that inconsistency must be to some extend tolerated during development such that development is not inhibited [48].

Other formalisms

Some other formalisms for detecting inconsistencies rely on common representations for different models. For example, Diskin et al. [49] propose to merge graph representations of heterogeneous models and then use the resulting single typed graph to detect inconsistencies. This approach is also an example of representing models as graphs. In another proposed approach, models are represented as graphs denoting logical facts about the models [50]. Similar to the first approach, the graphs are then used to derive contradictions. Other approaches have been proposed in which models are represented by the operations that are needed to construct them. After this representation step, logical rules are defined to detect inconsistencies between the models [51].

Reflection

It is noteworthy that, although inconsistency challenges have been studied for many years, there are still research articles being published on the topic, which is still considered to be very challenging to achieve in industry. The result is that all proposed approaches are created with a certain set of requirements in mind that are identified as required for adoption in some particular industrial practice. Consistently, we do not pretend that our approach is universally applicable and somehow better than all the other proposed approaches of the past. Rather, we identified our own set of requirements induced by the industrial settings under study and have proposed an approach for meeting those requirements.

Chapter 4

Research Results

In this chapter, we discuss the results of our research. We first present the contributions of the thesis and how they were validated. Then, we highlight the specific contributions brought by each of the four included papers.

4.1 Thesis Contributions

This thesis presents the following three research contributions.

- C_1: Identified challenges of combining MBD and CI.
- C_2: An approach for lightweight inter-model consistency checking in continuous model-based development.
- C_3: An approach for alleviating the change propagation process in a model-based product line.

A mapping of contributions to research goals is shown in Table 4.1.

4.1.1 C1: Identified challenges towards combining MBD and CI

C_1 is brought by Paper A and Paper B in which we identified the state-of-the-art in modeling tools, the state-of-practice at several companies, and challenges

Table 4.1. Mapping of contributions to research goals.

	RG_1	RG_2
C_1	X	
C_2		X
C_3		X

towards combining MBD and CI. In the aspects of integration, building, testing, and overall automation, several relevant practices for continuous MBD were identified. We divided the identified challenges into the following categories: human, business, non-functional, and functional. Although these challenges are not specific to *continuous* MBD, some of them become more troublesome when adopting shorter development cycles. In our research, we focused on those challenges and in particular on model synchronization.

Another interesting result was discovering some MBD projects in which the adoption of CI was not seen as a good idea. This seemed to stem mostly from the existence of many manual steps in the current process, which are not easily performed at a higher frequency. Moreover, we found that those among our industrial partners that are most mature in the adoption of MBD and CI develop all their models in one single tool. This is done to avoid some of the most intimidating challenges, like tool interoperability and model synchronization.

Validation: The two papers forming this research contribution report on empirical findings on challenges in continuous MBD. To ensure the validity of these findings, several measures were taken in the design of these studies, such as ensuring a sampling of study subjects working in different roles and at different companies. More details are provided in Paper A and Paper B.

4.1.2 C2: Lightweight Approach to Consistency Management

Model synchronization, in particular managing consistency between different artifacts, arose as one of the core challenges. Hence, we studied inter-model consistency checking in industrial settings and proposed an approach for their lightweight management, within a continuous integration pipeline. This contribution is described in Paper C, which presents an approach to manage consistency between heterogeneous artifacts as well as a tool for a CI pipeline. We have contributed to the existing state-of-the-art and practice by focusing both on the continuous aspect of the consistency checks and their required "lightweightness" for usage in industrial settings. Despite their lightweight nature, our consistency checks still give useful information on structural inconsistencies.

Validation: The approach and functionality of the tool have been evaluated using an example system commonly used in the relevant literature. To evaluate the other important aspects related to usability and applicability in practice, an evaluation with industrial partners is planned as future work. In fact, we are currently working with an industrial partner on establishing these types of consistency checks between their system model and corresponding code-base.

4.1.3 C3: Change Propagation in a Model-Based Product Line

This thesis contribution is carried by Paper D. It addresses MBD settings in which multiple variants of software are developed in a *clone-and-own* product line. In such settings, changes in the product line may need to be propagated to derived products. Significant effort is spent on the analysis of the impact of changes in the product line on derived products. Our contribution is an approach for semi-automating the change propagation. We identified two benefits of our approach: (i) the analysis and change propagation process is simplified, and (ii) the approach can be used to move from a clone-and-own product line to a more structured organization of reuse across variants.

The domain expertise of developers is required to make decisions on change propagation since these choices depend on the requirements of the different products. Therefore, we instead identified the tasks with the most manual effort and provided techniques to automate those.

Validation: We evaluated the approach using publicly available models. Also, we report on qualitative results in terms of experiences of applying the proposed approach to industrial models.

4.2 Paper Contributions

Below we list abstracts and brief descriptions of the contributions of the included papers. A mapping of research contributions to included papers is shown in Table 4.2.

Table 4.2. Mapping of research contributions to included papers.

	P_A	P_B	P_C	P_D
C_1	X	X		
C_2			X	
C_3				X

4.2.1 Personal Contributions

I have been the main author and driver of the work for all included papers. The co-authors have been involved in all works through brainstorming and discussions. Furthermore, they have provided feedback on drafts of the papers.

4.2.2 Included Papers

Paper A: Continuous integration support in modeling tools.
Abstract: Continuous Integration (CI) and Model-Based Development (MBD) have both been hailed as practices that improve the productivity of software development. Their combination has the potential to boost productivity even more. The goal of our research is to identify impediments to realizing this combination in industrial collaborative modeling practices. In this paper, we examine certain specific features of modeling tools that, due to their immaturity, may represent impediments to combining MBD and CI. To this end, we identify features of modeling tools that are relevant to enabling CI practices in MBD processes and we review modeling tools with respect to their level of support for each of these features.
Paper contributions: Although the results are not surprising, the work contributes to the body of knowledge on impediments towards adopting CI in MBD.

Further, it strengthens some conclusions made previously by others that have indicated impediments such as tool interoperability and model versioning.

Paper B: Impediments to Introducing Continuous Integration for Model-Based Development in Industry
Abstract: Model-based development and continuous integration each separately are methods to improve the productivity of development of complex modern software systems. We investigate industrial adoption of these two phenomena in combination, i.e., applying continuous integration practices in model-based development projects. Through semi-structured interviews, eleven engineers at three companies with different modeling practices share their views on perceived and experienced impediments to this adoption. We find some cases in which this introduction is undesired and expected to not be beneficial. For other cases, we find and categorize several impediments and discuss how they are dealt with in industrial practice. Model synchronization and tool interoperability are found the most challenging to overcome and the ways in which they are circumvented in practice are detrimental for introducing continuous integration.
Paper contributions: The main contribution of this work is the finding that, in some of the studied settings, current practices actively inhibit companies from developing in shorter development cycles. We identify those practices and discuss how they are impeding the adoption of CI.

Paper C: Lightweight Consistency Checking for Agile Model-Based Development in Practice.
Abstract: In model-based development projects, models at different abstraction levels capture different aspects of a software system, e.g., specification or design. Inconsistencies between these models can cause inefficient and incorrect development. A tool-based framework to assist developers creating and maintaining models conforming to different languages (i.e. *heterogeneous models*) and consistency between them is not only important but also much needed in practice. In this work, we focus on assisting developers bringing about multi-view consistency in the context of agile model-based development, through frequent, lightweight consistency checks across views and between heterogeneous models. The checks are lightweight in the sense that they are easy to create, edit, use and

maintain, and since they find inconsistencies but do not attempt to automatically resolve them. With respect to ease of use, we explicitly separate the two main concerns in defining consistency checks, being (i) which modeling elements across heterogeneous models should be consistent with each other and (ii) what constitutes consistency between them. We assess the feasibility and illustrate the potential usefulness of our consistency checking approach, from an industrial agile model-based development point-of-view, through a proof-of-concept implementation on a sample project leveraging models expressed in SysML and Simulink. A continuous integration pipeline hosts the initial definition and subsequent execution of consistency checks, it is also the place where the user can view results of consistency checks and reconfigure them.
Paper contributions: Many approaches for checking inter-model consistency exist. The contribution of this work is an approach for checking inter-model consistency that is explicitly lightweight, i.e., easy to use and deploy in industrial settings. Furthermore, the approach is generic, it can be applied to any modeling language with a hierarchical structure that can be mapped onto a tree structure. It represents a first step towards creating a lightweight consistency checking approach that supports more types of structural consistency and can deal with the evolution of the involved models.

Paper D: Co-evolution of Simulink Models in a Model-Based Product Line.
Abstract: Co-evolution of metamodels and conforming models is a known challenge in model-driven engineering. A variation of co-evolution occurs in model-based software product line engineering, where it is needed to efficiently co-evolve various products together with the single common platform from which they are derived. In this paper, we aim to alleviate manual efforts during this co-evolution process in an industrial setting where Simulink models are partially reused across various products. We propose and implement an approach providing support for the co-evolution of reusable model fragments. A demonstration on a realistic example model shows that our approach yields a correct co-evolution result and is feasible in practice, although practical application challenges remain. Furthermore, we discuss insights from applying the approach within the studied industrial setting.
Paper contributions: To handle variability across different versions of devel-

oped software, product lines are adopted in industrial practice, often through clone-and-own reuse. This causes a lack of traceability and systematic re-use between variants. In this paper, we aid the hitherto manually performed process of propagating changes made in the product line to derived products. We do not anticipate the process to become completely automated in the future, but we expect this to be a step towards providing more automated means of analysis to help domain experts in their design decisions.

Chapter 5

Conclusion

In industrial practice, a broad range of MBD settings can be encountered. Across these settings, models are used at different levels of abstraction and in combination with various other types of artifacts. By advancing continuous MBD, we aim to allow companies to adopt shorter development iterations and faster feedback on their model-based designs.

This thesis presents three contributions to this goal. We first identified the challenges of combining MBD and CI. Then, we presented an approach for lightweight inter-model consistency checking in continuous model-based development. Finally, we presented an approach for alleviating the change propagation process in a model-based product line.

Both proposed approaches address model synchronization, a key challenge in establishing continuous MBD. However, holistic support for continuous MBD requires more improvements to functional aspects such as model synchronization and tool interoperability, as well as to process aspects such as collaborating on models and insight into quality metrics.

5.1 Future Work

In future research, we aim at extending and supplementing the presented approaches, and at addressing more of the identified challenges in various MBD settings, thereby further advancing continuous MBD in industry.

Extensions of approaches in this thesis. We are currently working on an industrial evaluation of the approach presented in Paper C. This might also lead to further refinements of the approach to ensure its lightweight nature. One of the starting points of Paper C is that the manual definition of traceability links across artifacts requires too much manual effort. We plan to further improve on our proposed approach in Paper C by establishing automated traceability link discovery methods. In early work towards this goal, we are working on an approach that uses information on the known structure of the model and naming conventions in model and code. Using these inputs, we aim to provide accurate suggestions for traceability links across artifacts.

The work of Paper D is planned to be extended with automated support for suggesting changes to test cases upon a change to a model. In the current setting, the majority of the test development effort is spent on ensuring test cases are still up to date after the model they cover is updated. To alleviate that effort, support for assessing the impact of model changes on test cases is needed in the first place. In the second place, for specific kinds of changes, these assessments may be improved by suggesting changes to the test cases that would synchronize them with the model again. We aim to combine this with our current work to establish a faster development cycle for models, their variants, and their test cases. This is an important step towards establishing continuous MBD in the model-based product line setting.

Support the functional aspects of continuous MBD. Additionally, we plan to address challenges identified in Paper A and Paper B that were left unsolved. In general, we work towards the goal of supporting continuous MBD in various settings. To do so, our current results need to be extended in the area of model synchronization and model management tasks. The latter includes e.g. automated analysis for change impact analysis, and model differencing and merging for parallel development by multiple, possibly geographically distributed, developers. Approaches to those challenges should furthermore be compatible with the continuous integration paradigm.

In Paper D, we incorporated the aspect of developing software variants. Variability is typically organized in software product lines, which can be used in MBD too. Models and parts of models can be re-used across different product variants. An emerging challenge is then to manage both the dimensions of variants and revisions of these models. This is particularly pressing in component-based systems, where variation occurs on three levels since 1) the components themselves exist in different variants (or alternatives), 2) across systems, various configurations of multiple components are in use, and 3) there are various systems expressed using these components and configurations. On top of that, these variants all exist in different revisions through time. A first challenge is then to optimize the way of modeling such systems to avoid duplication of volatile information. Furthermore, consistency checking methods are needed that are aware of these dimensions and can appropriately check consistency between the appropriate versions of different artifacts.

Supporting the continuous MBD process. Supporting continuous MBD would require, in addition to solutions to technical challenges, also improvements to tooling that supports the development process of teams of developers. For code-based software development, mature tools are available for supporting activities such as code reviews and issue reporting. Mature continuous MBD practices require such supporting tooling too, that furthermore is model-aware.

Another need in industrial practice is to get insight into the quality development artifacts. In continuous MBD, these metrics should be defined and measured across different artifacts, instead of being scoped to single models or code. In this context, consistency is merely one quality metric among the typically six: correctness, completeness, consistency, comprehensibility, confinement, and changeability [52]. The continuous integration pipeline could be a good host for the calculation and presentation of quality metrics, as is typically done in dashboards for code-based software development projects.

5.2 Summary

In this thesis, we have presented contributions that are aimed at advancing continuous MBD. We have shown different setups of continuous MBD, one in which the continuous integration pipeline is utilized to include inter-model consistency checking. In Paper A and Paper B, we identified challenges to continuous MBD and in Paper C and Paper D, we presented approaches to alleviating two of these challenges. The challenges have been identified through an investigation of state-of-the-art modeling tools and their support for continuous integration, as well as interviews with industrial MBD practitioners.

The proposed approaches are a lightweight consistency checking method and an approach to assist in change propagation within a product line. The two approaches have been defined in diverse MBD settings and in collaboration with industrial partners from different domains. Both approaches reduce the needed manual effort, which otherwise inhibits the adoption of short development cycles and, ultimately, continuous MBD.

Bibliography

[1] Robert N Charette. This car runs on code. *IEEE spectrum*, 46(3):3, 2009.

[2] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for Agile Software Development, 2001.

[3] Douglas C Schmidt. Model-driven engineering. *IEEE Computer*, 39(2):25, 2006.

[4] Scott Ambler. *Agile modeling: effective practices for extreme programming and the unified process*. John Wiley & Sons, 2002.

[5] Martin Fowler and Matthew Foemmel. Continuous integration. `http://martinfowler.com/articles/continuousIntegration.html`, 2006.

[6] Daniel Ståhl and Jan Bosch. Experienced benefits of continuous integration in industry software product development: A case study. In *The 12th IASTED International Conference on Software Engineering (Innsbruck, Austria, 2013)*, pages 736–743, 2013.

[7] Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, and Miguel A. Fernandez. An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical Software Engineering*, 18(1):89–116, Feb 2013.

[8] Robbert Jongeling, Jan Carlson, and Antonio Cicchetti. Impediments to introducing continuous integration for model-based development in industry. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 434–441. IEEE, 2019.

[9] Antonio Bucchiarone, Jordi Cabot, Richard F Paige, and Alfonso Pierantonio. Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, pages 1–9, 2020.

[10] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[11] Matthew Hause. The SysML Modelling Language. In *Fifteenth European Systems Engineering Conference*, volume 9, pages 1–12, 2006.

[12] MATLAB Simulink. The MathWorks, Natick, MA, USA.

[13] Manfred Broy, Sascha Kirstan, Helmut Krcmar, and Bernhard Schätz. What is the benefit of a model-based design of embedded software systems in the car industry? In *Emerging Technologies for the Evolution and Maintenance of Software Models*, pages 343–369. IGI Global, 2012.

[14] Greg Wilson. Software engineering revisited. https://third-bit.com/2019/05/30/software-engineering-revisited.html, 2019.

[15] Vahid Garousi, Markus Borg, and Markku Oivo. Practical relevance of software engineering research: synthesizing the community's voice. *Empirical Software Engineering*, pages 1–68, 2020.

[16] Vahid Garousi, Kai Petersen, and Baris Özkan. Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review. *Information and Software Technology*, 79, 07 2016.

[17] Gordana Dodig Crnkovic. Constructive research and info-computational knowledge generation. In *Model-Based Reasoning in Science and Technology*, pages 359–380. Springer, 2010.

[18] Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*, pages 7–23. Springer, 2003.

[19] Hessa Alfraihi and Kevin Lano. The integration of agile development and model driven development - a systematic literature review. In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development*, MODELSWARD 2017, pages 451–458. SCITEPRESS, 2017.

[20] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131, 2009.

[21] Object Management Group. Meta-object facility 2.0 core specification. https://www.omg.org/spec/MOF/2.0/PDF, 2006.

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

[23] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. Managing cloned variants: a framework and experience. In *Proceedings of the 17th International Software Product Line Conference*, pages 101–110, 2013.

[24] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Software & Systems Modeling*, 17(1):91–113, 2018.

[25] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical Assessment of MDE in Industry. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 471–480. IEEE, 2011.

[26] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch. Climbing the 'Stairway to Heaven'–A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of

Software. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 392–399. IEEE, 2012.

[27] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *Ieee Software*, 33(3):94–100, 2016.

[28] Yuefeng Zhang and Shailesh Patel. Agile model-driven development in practice. *IEEE software*, 28(2):84–91, 2011.

[29] Sylvia Ilieva, Iva Krasteva, Gorka Benguria, and Brian Elvesæter. Enhance your model-driven modernization process with agile practices. In *Proceedings of the 1st International Workshop in Software Evolution and Modernization–SEM 2013*, pages 95–102. Angers Loire Valley, France, 2013.

[30] Bernhard Rumpe. *Agile Modeling with UML: Code Generation, Testing, Refactoring*. Springer, 2017.

[31] Kevin Lano, Hessa Alfraihi, S Yassipour-Tehrani, and Howard Haughton. Improving the application of agile model-based development: Experiences from case studies. In *The Tenth International Conference on Software Engineering Advances*, pages 213–219, 2015.

[32] Ulf Eliasson, Rogardt Heldal, Jonn Lantz, and Christian Berger. Agile model-driven engineering in mechatronic systems-an industrial case study. In *International Conference on Model Driven Engineering Languages and Systems*, pages 433–449. Springer, 2014.

[33] Vicente García-Díaz, Jordán Pascual Espada, Edward Rolando Núnez-Valdéz, G Pelayo, B Cristina Bustelo, and Juan Manuel Cueva Lovelle. Combining the continuous integration practice and the model-driven engineering approach. *Computing and Informatics*, 35(2):299–337, 2016.

[34] Jokin Garcia and Jordi Cabot. Stepwise adoption of continuous delivery in model-driven engineering. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 19–32. Springer, 2018.

[35] ISO/IEC/IEEE. ISO/IEC/IEEE 42010:2011(E) Systems and software engineering – Architecture description. Technical report, Dec 2011.

[36] Jan Reineke, Christos Stergiou, and Stavros Tripakis. Basic problems in multi-view modeling. *Software & Systems Modeling*, pages 1–35, 2017.

[37] Antonio Cicchetti, Federico Ciccozzi, and Alfonso Pierantonio. Multi-view approaches for software and system modelling: a systematic literature review. *Software & Systems Modeling*, pages 1–27, 2019.

[38] Bran Selic. What will it take? a view on adoption of model-based methods in practice. *Software & Systems Modeling*, 11(4):513–526, 2012.

[39] Francis Bordeleau, Benoit Combemale, Romina Eramo, Mark van Den Brand, and Manuel Wimmer. Tool-support of socio-technical coordination in the context of heterogeneous modeling: A research statement and associated roadmap. 2018.

[40] Dimitrios Kolovos, Richard Paige, and Fiona Polack. Detecting and Repairing Inconsistencies Across Heterogeneous Models. In *2008 1st International Conference on Software Testing, Verification, and Validation*, pages 356–364. IEEE, 2008.

[41] Sebastian Herzig, Ahsan Qamar, Axel Reichwein, and Christiaan JJ Paredis. A conceptual framework for consistency management in model-based systems engineering. In *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, IDETC/CIE 2011; Washington, DC, United States, 28-31 August, 2011*, pages 1329–1339. ASME, 2011.

[42] Richard Paige, Phillip Brooke, and Jonathan Ostroff. Metamodel-Based Model Conformance and Multi-view Consistency Checking. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(3):11, 2007.

[43] Andy Schürr. Specification of graph translators with triple graph grammars. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163. Springer, 1994.

[44] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. Orthographic software modeling: A practical approach to view-based development. In *Evaluation of Novel Approaches to Software Engineering*, pages 206–219. Springer Berlin Heidelberg, 2010.

[45] Igor Ivkovic and Kostas Kontogiannis. Tracing evolution changes of software artifacts through model synchronization. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, pages 252–261. IEEE, 2004.

[46] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, and Hong Mei. Towards automatic model synchronization from model transformations. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 164–173. ACM, 2007.

[47] Antonio Cicchetti, Federico Ciccozzi, and Thomas Leveque. A hybrid approach for multi-view modeling. *Electronic Communications of the EASST*, 50, 2012.

[48] Anthony CW Finkelstein, Dov Gabbay, Anthony Hunter, Jeff Kramer, and Bashar Nuseibeh. Inconsistency handling in multiperspective specifications. *IEEE Transactions on Software Engineering*, 20(8):569–578, 1994.

[49] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. Specifying overlaps of heterogeneous models for global consistency checking. In *International Conference on Model Driven Engineering Languages and Systems*, pages 165–179. Springer, 2010.

[50] Sebastian Herzig, Ahsan Qamar, and Christiaan Paredis. An approach to Identifying Inconsistencies in Model-Based Systems Engineering. *Procedia Computer Science*, 28:354–362, 2014.

[51] Jerome Le Noir, Olivier Delande, Daniel Exertier, Marcos Aurélio Almeida da Silva, and Xavier Blanc. Operation based model representation: experiences on inconsistency detection. In *European Conference on Modelling Foundations and Applications*, pages 85–96. Springer, 2011.

[52] Parastoo Mohagheghi, Vegard Dehlen, and Tor Neple. Definitions and approaches to model quality in model-based software development–a review of literature. *Information and software technology*, 51(12):1646–1669, 2009.