



Hasso Plattner Institute for Software Systems Engineering
System Analysis and Modeling Group

Thesis

Evolution of Model-Driven Engineering Settings in Practice

Dissertation
zur Erlangung des akademischen Grades
"doctor rerum naturalium"
(Dr. rer. nat.)
in der Wissenschaftsdisziplin "Praktische Informatik"

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

von
Regina Hebig

Potsdam, June 20, 2014

This work is licensed under a Creative Commons License:
Attribution 4.0 International
To view a copy of this license visit
<http://creativecommons.org/licenses/by/4.0/>

Published online at the
Institutional Repository of the University of Potsdam:
URL <http://opus.kobv.de/ubp/volltexte/2014/7076/>
URN urn:nbn:de:kobv:517-opus-70761
<http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-70761>

Abstract

Nowadays, software systems are getting more and more complex. To tackle this challenge most diverse techniques, such as design patterns, service oriented architectures (SOA), software development processes, and model-driven engineering (MDE), are used to improve productivity, while time to market and quality of the products stay stable. Multiple of these techniques are used in parallel to profit from their benefits. While the use of sophisticated software development processes is standard, today, MDE is just adopted in practice.

However, research has shown that the application of MDE is not always successful. It is not fully understood when advantages of MDE can be used and to what degree MDE can also be disadvantageous for productivity. Further, when combining different techniques that aim to affect the same factor (e.g. productivity) the question arises whether these techniques really complement each other or, in contrast, compensate their effects. Due to that, there is the concrete question how MDE and other techniques, such as software development process, are interrelated. Both aspects (advantages and disadvantages for productivity as well as the interrelation to other techniques) need to be understood to identify risks relating to the productivity impact of MDE.

Before studying MDE's impact on productivity, it is necessary to investigate the range of validity that can be reached for the results. This includes two questions. First, there is the question whether MDE's impact on productivity is similar for all approaches of adopting MDE in practice. Second, there is the question whether MDE's impact on productivity for an approach of using MDE in practice remains stable over time. The answers for both questions are crucial for handling risks of MDE, but also for the design of future studies on MDE success.

This thesis addresses these questions with the goal to support adoption of MDE in future. To enable a differentiated discussion about MDE, the term "MDE setting" is introduced. MDE setting refers to the applied technical setting, i.e. the employed manual and automated activities, artifacts, languages, and tools. An MDE setting's possible impact on productivity is studied with a focus on changeability and the interrelation to software development processes. This is done by introducing a taxonomy of changeability concerns that might be affected by an MDE setting. Further, three MDE traits are identified and it is studied for which manifestations of these MDE traits software development processes are impacted. To enable the assessment and evaluation of an MDE setting's impacts, the Software Manufacture Model language is introduced. This is a process modeling language that allows to reason about how relations between (modeling) artifacts (e.g. models or code files) change during application of manual or automated development activities. On that basis, risk analysis techniques are provided. These techniques allow identifying changeability risks and assessing the manifestations of the MDE traits (and with it an MDE setting's impact on software development processes).

To address the range of validity, MDE settings from practice and their evolution histories were captured in context of this thesis. First, this data is used to show that MDE settings cover the whole spectrum concerning their impact on changeability or interrelation to software development processes. Neither it is seldom that MDE settings are neutral for processes nor is it seldom that MDE settings have impact on processes. Similarly, the impact on changeability differs relevantly. Second, a taxonomy of evolution of MDE settings is introduced. In that context it is discussed to what extent different types of changes on an MDE setting can influence this MDE setting's impact on changeability and the interrelation to processes. The category of structural evolution, which can change these characteristics of an MDE setting, is identified. The captured MDE settings from practice are used to show that structural evolution exists and is common. In addition, some examples of structural evolution steps are collected that actually led to a change in the characteristics of the respective MDE settings. Two implications are: First, the assessed diversity of MDE settings evaluates the need for the analysis techniques that shall be presented in this thesis. Second, evolution is one explanation for the diversity of MDE settings in practice.

To summarize, this thesis studies the nature and evolution of MDE settings in practice. As a result support for the adoption of MDE settings is provided in form of techniques for the identification of risks relating to productivity impacts.

Zusammenfassung

Um die steigende Komplexität von Softwaresystemen beherrschen zu können, werden heutzutage unterschiedlichste Techniken gemeinsam eingesetzt. Beispiele sind, Design Pattern, Serviceorientierte Architekturen, Softwareentwicklungsprozesse oder modellgetriebene Entwicklung (MDE). Ziel dabei ist die Erhöhung der Produktivität, so dass Entwicklungsduer und Qualität stabil bleiben können. Während hoch entwickelte Softwareentwicklungsprozesse heute schon standardmäßig genutzt werden, fangen Firmen gerade erst an MDE einzusetzen.

Jedoch zeigen Studien, dass der erhoffte Erfolg von MDE nicht jedes Mal eintritt. So scheint es, dass noch kein ausreichendes Verständnis dafür existiert, inwiefern MDE auch Nachteile für die Produktivität bergen kann. Zusätzlich ist bei der Kombination von unterschiedlichen Techniken damit zu rechnen, dass die erreichten Effekte sich gegenseitig negieren anstatt sich zu ergänzen. Hier entsteht die Frage wie MDE und andere Techniken, wie Softwareentwicklungsprozesse, zusammenwirken. Beide Aspekte, der direkte Einfluss auf Produktivität und die Wechselwirkung mit anderen Techniken, müssen aber verstanden werden um den Risiken für den Produktivitätseinfluss von MDE zu identifizieren. Außerdem, muss auch die Generalisierbarkeit dieser Aspekte untersucht werden. Das betrifft die Fragen, ob der Produktivitätseinfluss bei jedem Einsatz von MDE gleich ist und ob der Produktivitätseinfluss über die Zeit stabil bleibt. Beide Fragen sind entscheidend, will man geeignete Risikobehandlung ermöglichen oder künftige Studien zum Erfolg von MDE planen.

Diese Dissertation widmet sich der genannten Fragen. Dafür wird zuerst der Begriff "MDE Setting" eingeführt um eine differenzierte Betrachtung von MDE-Verwendungen zu ermöglichen. Ein MDE Setting ist dabei der technische Aufbau, inklusive manueller und automatische Aktivitäten, Artefakten, Sprachen und Werkzeugen. Welche Produktivitätseinflüsse von MDE Settings möglich sind, wird in der Dissertation mit Fokus auf Änderbarkeit und die Wechselwirkung mit Softwareentwicklungsprozessen betrachtet. Dafür wird einerseits eine Taxonomie von "Changeability Concerns" (potentiell betroffene Aspekte von Änderbarkeit) vorgestellt. Zusätzlich, werden drei "MDE Traits" (Charakteristika von MDE Settings die unterschiedlich ausgeprägt sein können) identifiziert. Es wird untersucht welche Ausprägungen dieser MDE Traits Einfluss auf Softwareentwicklungsprozesse haben können. Um die Erfassung und Bewertung dieser Einflüsse zu ermöglichen wird die Software Manufaktur Modell Sprache eingeführt. Diese Prozessmodellierungssprache ermöglicht eine Beschreibung, der Veränderungen von Artefaktbeziehungen während der Anwendung von Aktivitäten (z.B. Codegenerierung). Weiter werden auf Basis dieser Modelle, Analysetechniken eingeführt. Diese Analysetechniken erlauben es Risiken für bestimmte Changeability Concerns aufzudecken sowie die Ausprägung von MDE Traits zu erfassen (und damit den Einfluss auf Softwareentwicklungsprozesse).

Um die Generalisierbarkeit der Ergebnisse zu studieren, wurden im Rahmen der Arbeit mehrere MDE Settings aus der Praxis sowie teilweise deren Evolutionshistorien erhoben. Daran wird gezeigt, dass MDE Settings sich in einem breiten Spektrum von Einflüssen auf Änderbarkeit und Prozesse bewegen. So ist es weder selten, dass ein MDE Setting neutral für Prozesse ist, noch, dass ein MDE Setting Einschränkungen für einen Prozess impliziert. Ähnlich breit gestreut ist der Einfluss auf die Änderbarkeit. Zusätzlich, wird diskutiert, inwiefern unterschiedliche Evolutionstypen den Einfluss eines MDE Settings auf Änderbarkeit und Prozesse verändern können. Diese Diskussion führt zur Identifikation der "strukturellen Evolution", die sich stark auf die genannten Charakteristika eines MDE Settings auswirken kann. Mithilfe der erfassten MDE Settings, wird gezeigt, dass strukturelle Evolution in der Praxis üblich ist. Schließlich, werden Beispiele aufgedeckt bei denen strukturelle Evolutionsschritte tatsächlich zu einer Änderung der Charakteristika des betreffenden MDE Settings geführt haben. Einerseits bestärkt die ermittelte Vielfalt den Bedarf nach Analysetechniken, wie sie in dieser Dissertation eingeführt werden. Zum Anderen erscheint es nun, dass Evolution zumindest zum Teil die unterschiedlichen Ausprägungen von MDE Settings erklärt.

Zusammenfassend wird studiert wie MDE Settings und deren Evolution in der Praxis ausgeprägt sind. Als Ergebnis, werden Techniken zur Identifikation von Risiken für Produktivitätseinflüsse bereitgestellt um den Einsatz von MDE Settings zu unterstützen.

Acknowledgments

First of all, I like to thank my supervisor Professor Dr. Holger Giese, for valuable feedback, for sharing his experiences and know-how, and for encouraging and demanding me to follow my ideas when it was necessary. I also appreciate the support of the HPI research school, which enabled me to work and concentrate on my research during the last four years. Especially I am thankful to the professors of the research school for their sincere and constructive feedback.

For fruitful teamwork, I would like to thank my colleagues. I am grateful to Florian Stallmann for his early feedback on the study design; to Andreas Seibel and Gregor Berg for uncounted discussions, feedback, and close cooperation; and to Thomas Beyhl and Marco Kuhrmann for their constructive feedback on the thesis.

For their assistance in organizational and bureaucratic matters, I thank Kerstin Miers and Sabine Wagner. I also thank the students, who refined the tool support and who accompanied me during some of the interviews, namely Sebastian Stamm, Alexander Teibrich, Kinom Batoulis, Robert Boehme, Philipp Langer, Mychajlo Wolowyk, Gary Yao, and Armin ZamaniFarahani. For proofreading, I thank Richard Holmes.

Last but not least, I am grateful to the many engaged interviewees, who provided valuable, interesting, and often surprising insights into their everyday working practices.

“Science is organized knowledge.”
Immanuel Kant
German philosopher (1724 - 1804)

Contents

I. Introduction and Preliminaries	1
1. Introduction	3
1.1. Focus	4
1.2. Challenges and Thesis' Goals	6
1.3. Contributions	8
1.4. Structure	10
2. Preliminaries	13
2.1. Model-Driven Engineering	13
2.1.1. Comparing Development with and without MDE	16
2.1.2. Model Management and Megamodels	17
2.2. Productivity	18
2.2.1. Influences on Productivity	19
2.2.2. Internal Quality Attributes	20
2.2.3. MDE Mechanisms	20
2.2.4. Interrelation of Productivity Terminologies used in this Thesis	21
2.3. Software Development Processes	22
2.4. Patterns	23
2.5. Empirical Research Methods	23
2.6. Example MDE setting	26
II. MDE in Practice	27
3. Analytical Insights	29
3.1. MDE Settings	29
3.2. Analytical Insights: Changeability	30
3.2.1. Basic Influences on Changeability	30
3.2.2. MDE's Influence on Changeability	31
3.2.3. Interrelation of Influences on Changeability	34
3.2.4. Obstacles for Creating a Global Measure for Changeability	34
3.3. Analytical Insights: Process Interrelation	35
3.3.1. Reference Model for Combinations of MDE Settings and Software Development Processes	35
3.3.2. Assumptions and Adaptation within Literature Combinations	38
3.3.3. MDE Traits	39
3.4. Analytical Insights: Structural Evolution	41
3.4.1. Types of Changes in Evolution of MDE Settings	42
3.4.2. Possible Impact of Evolution on Changeability	42
3.4.3. Possible Impact of Evolution on the Manifestation of MDE Traits	44
3.5. Hypotheses	44
3.5.1. Hypotheses on Changeability	44
3.5.2. Hypotheses on Process Interrelation	45
3.5.3. Hypotheses on Evolution	45
4. Studies	47
4.1. Initial Study of MDE Settings in Practice	47
4.1.1. Study Design	48

Contents

4.1.2. Overview of Case Studies from SAP	50
4.2. Meta Study of Evolution	52
4.2.1. Review Process	52
4.2.2. Overview of Cases Studies	53
4.3. Follow-up Study of MDE Settings in Practice and their Evolution	54
4.3.1. Study Design	54
4.3.2. Overview of Cases Studies	55
4.4. Threats to Validity	58
4.4.1. Construction Validity	58
4.4.2. External Validity	60
4.4.3. Conclusion Validity	61
5. Study Results: MDE in Practice	63
5.1. MDE Settings in Practice	63
5.1.1. General Results	63
5.1.2. Observations on MDE Settings	66
5.2. Nature of Structural Evolution	67
5.2.1. Existence and Relevance	68
5.2.2. Observations on Structural Evolution	72
III. Modeling and Analysis	81
6. Modeling	83
6.1. Language Design Concepts	83
6.1.1. Object Flow	84
6.1.2. Pre und Postconditions	86
6.1.3. Degree of Automation	88
6.1.4. Hierarchy	89
6.2. Software Manufacture Models	90
6.2.1. Language Description	90
6.2.2. Notation of Complex Activities	96
6.2.3. Activity Semantic	97
6.2.4. Activity Orders: Predecessors and Successors	98
6.3. Software Manufacture Model Patterns	99
6.3.1. Pattern Language	100
6.3.2. Pattern Matching	102
6.3.3. Discussion of Pattern Matching	104
7. Analysis	107
7.1. Analysis of Changeability	107
7.1.1. Description Structure of Patterns	107
7.1.2. Identification of Proto-Pattern	108
7.1.3. Software Manufacture Model Proto-Pattern Subsequent Adjustment	109
7.1.4. Software Manufacture Model Proto-Pattern Creation Dependence	113
7.1.5. Software Manufacture Model Proto-Pattern Split Manufacture	116
7.1.6. Software Manufacture Model Proto-Pattern Anchor	118
7.2. Analysis of Process Interrelation	119
7.2.1. Analysis Method for Phases and Synchronization Points	119
7.2.2. Analysis Method for Manual Information Propagations	121
7.2.3. Analysis Method for Complexity of Activity Chains	123
7.3. Analysis Approach	125
8. Implementation	129
8.1. Editors	129
8.2. Pattern Matcher	129

8.3. Automation Support for Order-Based Analysis Techniques	131
8.3.1. Automated Identification of Predecessor Sets, Predecessors, and Successors	131
8.3.2. Automation Support for Analysis of Phases and Synchronization Points	137
8.3.3. Automation Support for Analysis of Complexity of Activity Chains	139
9. Study Results: Employing Modeling and Analysis	143
9.1. Preparation of Models for Analysis	143
9.1.1. Model Preparation	144
9.1.2. Resulting Threats to Validity of Analysis Results	145
9.2. Findings on Changeability	147
9.2.1. Changeability Analysis of Case Studies	147
9.2.2. Changeability Analysis of an Open Source Example	149
9.2.3. Evaluation of Hypothesis $H_{changeability}$	151
9.2.4. Threats to Validity	152
9.3. Findings on Process Interrelation	152
9.3.1. Phases and Synchronization Points in Case Studies	152
9.3.2. Manual Information Propagation in Case Studies	159
9.3.3. Complexity of Activity Chains in Case Studies	162
9.3.4. Evaluation of Hypothesis H_{traits}	165
9.3.5. Threats to Validity	167
9.4. Findings on the Influence of Evolution on Changeability and Process Interrelation	167
9.4.1. Method	167
9.4.2. Effects of Evolution on Changeability	168
9.4.3. Effects of Evolution on Process Interrelation	170
9.4.4. Discussion of Evolution Effects	172
9.4.5. Threats to Validity	172
10. Evaluation of Modeling and Analysis	173
10.1. Evaluation of Software Manufacture Model Language	173
10.1.1. Applicability	173
10.1.2. Comparability	173
10.1.3. Analysis Support	175
10.1.4. Discussion	176
10.2. Evaluation of Analysis Techniques	176
10.2.1. Applicability of Analysis	176
10.2.2. Need of Analysis Techniques	177
10.2.3. Discussion	177
IV. Roundup	179
11. Related Work	181
11.1. Related Work on Studies about MDE in Practice	181
11.2. Related Work on Modeling and Analysis of MDE Settings	182
11.2.1. Modeling of MDE Development	182
11.2.2. Analysis Techniques for MDE Settings	183
11.3. Related Work on MDE's Influence of Changeability and Productivity	185
11.3.1. Theoretical Explanations on MDE's Changeability Influence	185
11.3.2. Empirical View on MDE Productivity	186
11.4. Related Work on Process Interrelation	188
11.4.1. Combination of MDE and Software Development Processes	188
11.4.2. Impacts between MDE and Software Development Processes	188
11.5. Related Work on MDE Evolution	189
12. Conclusion	191
12.1. Summary	191

Contents

12.2. Thesis' Goals	192
12.3. Implications	193
12.3.1. Changeability & Process Interrelation	193
12.3.2. Structural Evolution	195
Bibliography	197
Selected Publications	211
Additional Publications	213
 V. Appendix	 215
A. Languages used in Captured Case Studies	A-1
B. Case Studies on Evolution of MDE Settings	B-1
B.1. BO Evolution History	B-1
B.2. Ableton Evolution History	B-2
B.3. Cap1 Evolution History	B-3
B.4. Cap2a and Cap2b Evolution Histories	B-3
B.5. VCat Evolution History	B-3
B.6. Carmeq Evolution History	B-3
C. Language Simplification	C-1
D. Notation of Software Manufacture Model Language in Graphical Editor	D-1
E. WSDL Example	E-1
F. CodeListings	F-1
F.1. Identifcication of Precessor Sets	F-1
F.2. Identification of Phases and Synchronization Points	F-3
F.3. Identification of Lengths of Activity Chains	F-8
G. Analysis Details for Changeability Patterns and MDE Traits	G-1
G.1. Pattern Occurrences for the 11 Case Studies	G-1
G.2. Manual Information Propagation within the 11 Case Studies	G-3
G.3. Complexity of Activity Chains of the 11 Case Studies	G-5
G.4. Phases within the 11 Case Studies	G-9
H. Literature on Support for Evolution in Context of MDE	H-1
I. Overview of Publications	I-1

Part I.

Introduction and Preliminaries

1. Introduction

Today, software is ubiquitous as part of automotive systems, aviation systems, mobile devices, telecommunication systems, or enterprise management systems. Software systems within the same domain often have commonalities in functionality and are confronted with similar challenges. For example, embedded systems usually need to deal with resources constraints. Between the domains, requirements on software systems functionality and quality can differ considerably. However, what is common to all domains is that associated software systems are getting more and more complex. In consequence, there is a growing need to tackle complexity of software and to improve productivity of development without decreasing the quality of the resulting system. To reach this goal, diverse approaches and techniques are in use.

One of these approaches is to base software development on modeling languages (e.g. UML [156]) in addition to (or even instead of) traditional programming languages. The core motivation for the use of models is to provide a more abstract view of the required software system. This more abstract view supports communication as well as documentation or even automated analysis of systems in context of quality assurance. Often this approach is accompanied by the use of tools that automatically consume models in order to generate source code or other models (which is often called model transformation). Techniques that implement this approach to use models and automation have various names, e.g. model-driven engineering (MDE), model-based software engineering (MBSE), model-driven design (MDD), or model-driven architecture (MDA) [154]. The different names partly refer to the spectrum of goals that are targeted. While the term “model-based” is mainly used to refer to the aim of improving communication through the use of models, the term “model-driven” often refers to the additional aim of saving effort through the generation of code from models. The vision behind these techniques is to disburden developers from recurring and possibly error-prone coding tasks [79], [111].

In this thesis, the term model-driven engineering (MDE) is used to refer to each technique that implements a facet of the approach to use models and automation. To be able to reason about differences between applications of MDE, the more detailed term MDE setting is defined as

the set of manual, semi-automated, and automated activities that are employed during development, the set of artifacts that are consumed or produced by these activities, the set of languages used to describe the artifacts, as well as the set of tools that allow the editing of used languages or that implement automated activities.

Artifacts can be models or code files. Examples for automated activities are transformations or generation steps, while examples for manual activities are modeling and coding. MDE tools, which automate development activities, capture crucial knowhow and domain knowledge and are consequently a valuable asset of a company [179]. Further, manual and automated are often intermixed (e.g. as in [52, 72, 103, 116, 171]).

To allow developers to focus on domain aspects during development it is possible to create domain-specific languages (DSLs) [110, 200]. Due to the restrictions on the domain, it is then often possible to completely generate the source code or even interpret the specification directly. Often, most computation specific aspects are encapsulated in generators, compilers and interpreters. If a DSL is a modeling language such approaches are called domain-specific modeling (DSM) (e.g. [50] or [73], further examples for DSMs are listed at the website of the tool MetaCase¹). DSMs that allow specification of a system completely within the domain-specific modeling language (DSML) are examples for the most extreme vision of MDE. In this facet of MDE it is no longer necessary to manipulate program code. Further, in extreme cases the DSML is designed in a way that only few software engineering skills are required to specify the system. This ultimate vision of MDE is that software can be developed in close collaboration with domain experts or even by domain experts themselves.

However, conclusive evidence that MDE improves productivity is missing. On the contrary, there are indications that there are strong differences in the degree of success of MDE in practice. While there are some reports on projects that showed large productivity gains, other reports document a

¹<http://www.metacase.com/cases/> (last access at November 9th, 2013)

1. Introduction

loss of productivity. So far, explanations for these differences are rare. A main part of this problem is that there is only partial knowledge about how MDE influences productivity. As described above, this knowledge focuses mainly on potential positive effects of MDE. However, there is little research on potential negative effects of MDE. Similarly, the interplay of MDE and other productivity influencing factors can lead to variations in the productivity impact. While literature on the interplay of MDE and organizational factors exists, research on the interplay of MDE and other factors, such as software development processes, is still rare. A second part of the problem is the lack of knowledge about how the structures of MDE settings differ in practice. As a consequence, there is often a focus on the quality of single tools when searching for explanations for the differences in the degree of success of MDE in practice. However, it is unclear whether these differences might also be explained by differences in the structure of MDE settings. A final part of this problem is that there is little knowledge about whether the productivity impact of an MDE setting stays stable over time or can change (e.g. in the context of evolution of this MDE setting).

In this chapter, an overview is given of factors that can influence productivity and might interfere with MDE. In addition, a focus is defined for this thesis. Subsequently, the goals of this thesis are introduced in detail. Finally, an overview of the contributions of this thesis is provided, followed by an outline of the structure of this thesis.

1.1. Focus

One goal of research in software engineering is to enable software engineers to build complex systems with a high quality. If the quality of the software and the time to produce the software should stay stable while handling an increasing size and complexity of software, it is necessary to apply techniques that increase the productivity. Consequently, research focuses on the question how productivity can be influenced. A detailed list of factors that can influence productivity is provided by Balzert in [17] (referred to as productivity influencing factors in the following). This list splits the productivity influencing factors into five groups: *organization*, *management*, *employees*, *product influences*, and *process influences*. *Process influences* can be restrictions on development time, duration of the project, stability of requirements, technological support, methods or process quality, the programming languages used, or reuse [31, 134, 187].

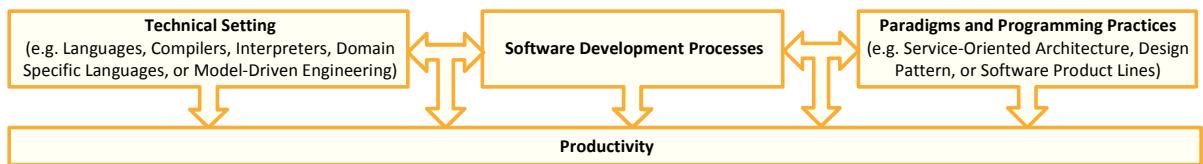


Figure 1.1.: Groups of techniques and practices that are used to affect productivity via process influences.

The different process influences can be affected with various techniques and practices. As summarized in Figure 1.1, such techniques and practices can be split into different groups. First, there are *software development processes* that coordinate development (e.g. Rational Unified Process (RUP) [100], V-Modell XT [82], or Scrum [176]). Processes often build on practices like rapid prototyping or incremental development [31, 148]. Second, there is the *technical setting* that is used to develop the software. This technical setting includes used languages and tools, like compilers or interpreters. Research on more productive languages for the development of software focuses on domain specific languages (DSLs) or modeling languages, amongst others. Further, model-driven engineering (MDE) and DSLs aim at enhancing reuse, by providing techniques for automation of code creation, documentation, or quality assurance [92, 206]. Finally, there are modern *programming practices and paradigms*. Such practices aim for example at eliminating rework. This can include the use of design patterns [86] or service-oriented architecture (SOA) [127]. Software product lines (SPLs) [163] are an example of a technique that aims at supporting massive reuse of software components. The development of software product lines is often supported by the chosen technical setting (e.g. feature trees might be used to model a planned SPL). Still, SPLs are, comparable to SOAs, considered as (architectural) paradigm here.

All these techniques and practices are not used in isolation, but may be employed simultaneously. Thus, it is possible that they interfere with each other. This multitude of techniques and possible interferences between these techniques makes it difficult to study all influences on productivity at once.

Techniques and Practices

From Balzert's five groups of productivity influencing factors (organization, management, employees, product influences, and process influences), *process influences*, which concern how development techniques and practices can affect productivity (as summarized in Figure 1.1), are considered in this thesis. Within the range of techniques and practices that address process influences on productivity, the main object of this thesis is model-driven engineering (MDE) as technical setting for development. As explained above, the aspects of an MDE approach that constitute the technical setting are referred to as MDE setting in this thesis.

An MDE setting affects productivity directly, but also by interrelation with the software development process used as well as the paradigms and programming practices that are applied. While the application of different *paradigms and programming practices*, such as SOAs or SPLs, is not necessarily part of each software project, some kind of *software development process* is always applied (whether explicitly specified or not). Thus, MDE is always used in combination with a *software development process*. Therefore, the interrelation between MDE and *software development processes* is in focus of this thesis, too, while the interrelation between MDE and *paradigms and programming practices* remains a topic for future research.

To summarize, the productivity influencing factors studied in this work include MDE settings as well as their interrelation to software development processes.

Software Quality Attributes

Productivity manifests in several aspects of development. Internal software quality attributes, like reusability, maintainability, portability, changeability, or interoperability, can be seen as perspectives that cover productivity for different use cases [1]. For example, changeability can be considered as productivity when changing a software system. Consequently, techniques that affect productivity can also affect these internal software quality attributes.

Similar to the multitude of productivity influencing factors, the way in which the different internal software quality attributes are affected can vary strongly. For this reason, one internal software quality attribute will be studied. This is changeability, i.e. "the amount of effort necessary to change a system" [1], since it plays an important role at all phases of the software life cycle. For example, changeability is required, when developing a system in an iterative manner, during maintenance, and when a system is subject to extensive changes.

Focus Summary

To summarize, as shown in Figure 1.2, this thesis focuses on how MDE as well as the interrelation of MDE and software development processes can affect productivity (especially changeability).

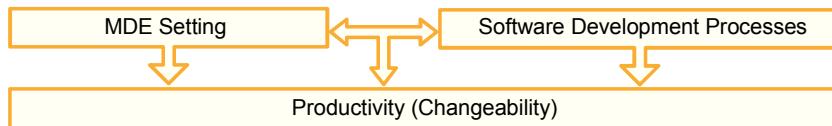


Figure 1.2.: The focus of this research is on the influence of model-driven engineering on productivity, especially changeability, as well as on the interrelation and potential interferences of MDE with software development processes.

1.2. Challenges and Thesis' Goals

The question under which conditions improvements can be achieved by adopting MDE is still open and subject to research (e.g. in [146, 213]). While in some domains the application of MDE is established, the industry in other domains is only just beginning to adopt MDE in practice. However, there is so far no proof that MDE always improves productivity. In fact, there are reports of successful MDE applications as well as reports on fruitless MDE applications [143].

For this motivation this thesis addresses three challenges that are summarized in Figure 1.3. First, in order to study the characteristics on MDE settings it is necessary to capture examples from practice. Second, it is a challenge to identify risks relating to the productivity impacts of MDE settings. Therefore, it needs to be understood how MDE settings affect changeability and whether they define constraints on the used software development processes.

Finally, a risk analysis implies additional effort for a company. Consequently, the question arises whether such evaluations are necessary for all MDE settings. For example, when specific strengths and weaknesses are similar for all MDE settings used in practice, there is no need to analyze risks of a specific MDE setting. Therefore, the second challenge is to determine the generalizability of knowledge about risks for the productivity impact of MDE settings.

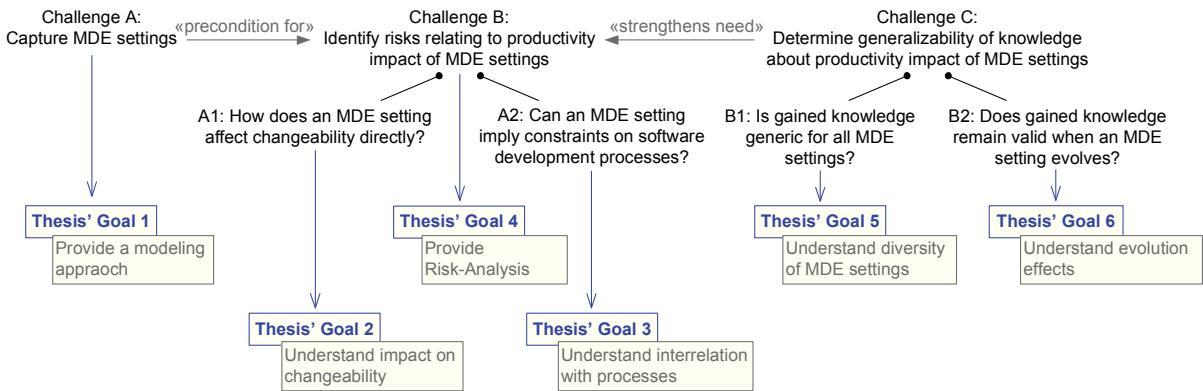


Figure 1.3.: Overview of challenges that are addressed in this thesis.

Capture MDE Settings

A precondition for studying MDE settings is the ability to capture examples of them. Modeling languages to capture MDE development either aim at documenting the process of development activities (e.g. [2, 24, 25, 26, 40, 51, 58, 117, 129, 164], or [183]) or at documenting and maintaining the set of development artifacts and their interrelation ([P5], e.g. [65, 66], or [204]).

However, a connection of both views is missing. Thus, current modeling techniques do not allow documenting how the set of artifacts and their relations change, when development activities are performed. This lack makes it difficult to reason about the characteristics of MDE settings. Therefore the following goal is formulated for this thesis:

Thesis' Goal 1 *It is a goal of this thesis to provide a modeling approach that enables capturing of MDE settings, such that the effects of activities on artifacts and their interrelations can be documented.*

Identify Risks Relating to Productivity Impact of MDE Settings

As already discussed above in Section 1.1, the question how MDE affects productivity has various aspects. In order to develop risk analysis techniques for the productivity impact of MDE settings, two questions will be addressed: Firstly, how MDE settings directly affect changeability, and secondly whether an MDE setting can imply constraints on software development processes.

So far, the main effort in understanding why MDE supports productivity has gone into in understanding why generation of code can help to reduce development effort (e.g. [110]) and in understanding how models and modeling languages support developers in working more productively (e.g. [206]). For example, Heijstek et al. [95] investigated how model size and complexity influence the development effort and quality of software. Existing studies on MDE in practice only seldom focus on the structure of MDE settings [15, 37, 41, 71, 77, 94, 95, 99, 107, 114, 119, 133, 135, 136, 143, 145, 146, 152, 190, 195, 196, 205, 210, 211]. However, there is little knowledge about the effect that the structure of an MDE setting as a whole (e.g. the interplay of manual and automated activities with different artifacts) can have on productivity (as illustrated in Figure 1.4). As a consequence, it is difficult to exclude negative effects on productivity when an MDE setting is established. More knowledge on an MDE setting's effects would be a prerequisite for the creation of appropriate analysis techniques to identify risks for productivity. As described in Section 1.1, the perspective on productivity that is in focus of this thesis is changeability. Therefore, the following goal is formulated for this thesis:

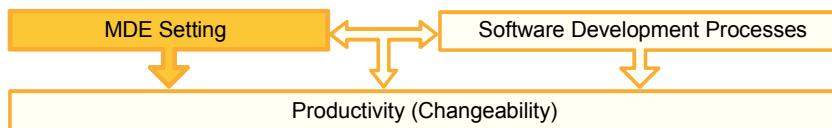


Figure 1.4.: The first focus of this thesis is the question how an MDE setting can influence changeability.

Thesis' Goal 2 *A goal of this thesis is to enhance the knowledge about how an MDE setting can influence the changeability of the software that is built with that MDE setting.*

Different studies have shown that software development processes and MDE settings can define constraints on each other (e.g. [8, 96, 190]). For example, Mahe et al. [130] highlight that short iteration cycles have to be supported by an MDE approach when it is applied with an agile development process. However, the interrelation of MDE settings and software development processes (as illustrated in Figure 1.5) has not yet been investigated in full depth. Consequently, it is not clear whether and how MDE settings and software development processes can complement or interfere with each other. More knowledge on the interrelations of MDE settings and software development processes would be prerequisite for educated decisions on how to combine both techniques, when introducing MDE within a company. Therefore, the following goal is formulated:

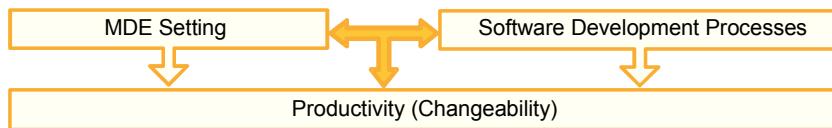


Figure 1.5.: The second focus of this thesis is the question whether MDE settings can interfere with software development processes.

Thesis' Goal 3 *It is a goal of this thesis to enhance the knowledge about how an MDE setting can interfere with a software development process.*

Gaining knowledge about how MDE settings can influence changeability and interrelate with software development processes is the precondition for the development of risk analysis techniques. Such techniques can make it possible to identify and handle risks for changeability during development. Similarly, an analysis of the interrelation to software development processes can help to tailor the process appropriately. Such analysis techniques are still rare. In consequence, there are no systematic methods to make

1. Introduction

developers aware of changeability risks or a disadvantageous combination with a software development process. For this reason a next goal is formulated for this thesis:

Thesis' Goal 4 *A goal of this thesis is to provide analysis techniques for MDE settings that support the identification of risks for changeability as well as constraints on software development processes.*

Determine Generalizability of Knowledge about Productivity Impact of MDE Settings

An analysis of the question whether MDE settings can imply risks on productivity can be expensive. The question arises whether it is possible to generalize knowledge that is gained during such an analysis. It might turn out that MDE settings in practice have differing characteristics. This leads to the additional question whether the knowledge gained about an MDE setting remains valid when this MDE setting changes over time and evolves.

As mentioned above, some applications of MDE in practice are successful and others not. In theory, it is possible that this difference is partly caused by the way MDE settings affect productivity (directly or indirectly via their interrelation with software development processes). In practice, this might only be the case when MDE settings differ in their productivity impact. However, it is not known whether MDE settings that include risks on changeability occur in practice. Similarly, it is not clear how often MDE settings in practice affect software development processes if at all. If MDE settings in practice do not differ in their influence on changeability and processes, these influences cannot be candidates to explain the diversity of the degree of success of MDE applications. However, if MDE settings in practice do differ in their influence on changeability and processes, a need arises for analysis techniques to identify whether the trait manifestations and changeability risks occur in a given MDE settings. This leads to following goal for this thesis:

Thesis' Goal 5 *A goal of this thesis is to evaluate whether the way an MDE setting influences the changeability and the way an MDE setting interferes with a software development process are potential explanations for the differences in the degree of success of applications of MDE in practice. Note that it is not a goal to prove that these characteristics of an MDE setting explain the differences in the degree of success. Instead, it shall be tested whether it is plausible that these characteristics can lead to differences in the degree of success in practice.*

There are several reports about the direct introduction of specific MDE approaches created in research (e.g. as summarized in [94]). However, after being introduced in practice, MDE settings might be subject to change. This evolution of modeling languages or model transformations has been studied in research (e.g. in [12, 97, 87, 126, 150, 203, 216]). However, there is a lack of knowledge about what kind of evolution MDE settings actually undergo in practice. Consequently, it is also not known whether evolution can change an MDE setting, so that its impact on changeability or its interrelation to software development processes changes, too. Thus, it is not understood whether evolution can invalidate knowledge about these characteristics of an MDE setting. Therefore a final goal is formulated for this thesis:

Thesis' Goal 6 *A goal of this thesis is to understand whether evolution in practice might change an MDE setting's impact on changeability and software development processes.*

1.3. Contributions

There are three groups of contributions in this thesis: insights that result from a systematic analysis of the problem domain, insights that result from empirical data, and a modeling language with analysis methods for MDE settings. Based on systematic analyses of the problem domain, i.e. the definitions of MDE settings, changeability, and software development processes, several analytical insights on MDE are presented in Chapter 3.

First, three traits of MDE settings are identified that can be manifested, such that they imply constraints on the software development process that is used. These traits are *phases*, which influence how

developers coordinate their work, *manual information propagation*, which influences for which artifacts additional quality assurance is required, and *complex activity chains*, which can make an MDE setting inappropriate for the application within an agile process. Second, it is established that MDE settings can include risks for changeability. MDE settings can lead to unexpected loss or preservation of content within artifacts and relations between artifacts, when changes are applied.

A taxonomy of possible types of changes that can occur to an MDE setting during evolution is introduced (as illustrated in Figure 1.6). The identified class of structural changes affects the number of artifacts, tools, languages, as well as the number of automated and manual activities in the MDE setting or the order of manual and automated activities) are distinguished from non-structural changes (i.e. evolution of languages, transformations, or generators). Evolution that consists of structural changes is referred to as structural evolution. This taxonomy of change types is accompanied by an analysis of how each change type affects an MDE setting's influence on different aspects of productivity. It is shown that there are change types that can alter how an MDE setting influences changeability. Similarly, it is analyzed how each change type alters how an MDE setting interferes with software development processes.

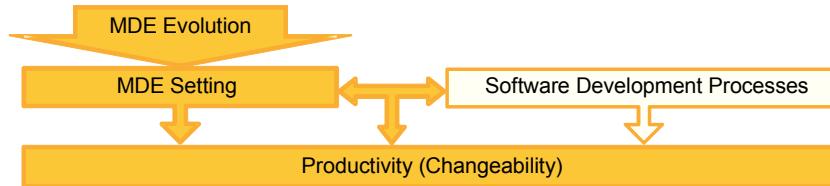


Figure 1.6.: Part of this thesis is the question what forms of evolution occurs on MDE setting in practice and how does this influence the characteristics of the MDE setting.

To improve the knowledge base about MDE settings in practice, three studies were performed (described in Chapter 4). First, an exploratory observational field study was performed to capture MDE settings from practice. In addition to the six case studies captured, the exploratory character of this field study made it possible to capture some additional observations. One of these was that most of the MDE settings considered are the result of evolution of older MDE settings that were in use before. To gain more data about this evolution, a meta study was performed to identify whether reports on MDE settings in practice include hints on structural evolution, too.

Structural evolution is studied in Chapter 5, based on these two data sets (the records from the MDE settings that were captured in the observational field study, on the one hand and the literature reports on the other hand). The insight is that structural evolution exists in practice and seems to be common. Thus, it is shown that types of evolution that can alter an MDE setting's influence on changeability actually do occur in practice. To gain more information about the reasons for the occurrence of structural evolution, the observational descriptive field study was extended to capture additional MDE settings in practice together with their evolution history. As a result this thesis contributes observations on the combination of change types within an evolution step as well as on the motivations, trade-offs, and triggers for structural evolution in practice.

The analytical insights reveal that MDE settings can lead to risks for changeability. In addition, MDE settings can imply constraints on the software development process. Further the empirical insights show that structures that cause these risks and constraints might be introduced into the MDE setting through evolution. To support developers in identifying these properties, this thesis introduces techniques for modeling and analyzing MDE settings (in Chapters 6 and 7).

First, the Software Manufacture Model language is introduced, to better capture processes of activities from MDE settings. This modeling language is an extended process modeling language that can be used to capture how the relations between artifacts (i.e. overlaps in content or references), change when manual or automated activities are applied. Further, the applicability of the language to practical MDE settings is evaluated by using it to capture the case studies that were collected in the context of this thesis. Second, the Software Manufacture Model Pattern language is introduced, which makes it possible to formulate patterns of one or more Software Manufacture Model activity. This can be used to capture knowledge about structures in MDE settings that lead to risks for changeability or other concerns.

1. Introduction

For the analysis, simple methods are introduced in Chapter 7. These make it possible to identify MDE traits that can imply constraints on software development processes. Based on the Software Manufacture Model Pattern language, four proto-patterns are introduced (i.e. prospective patterns for which a regular occurrence in practice is not yet shown [7]). Two of these proto-patterns describe structures that can lead to risks for changeability, while the other two describe solutions to prevent or remove risks for changeability.

Modeling and analysis techniques are supported by tools, as described in Chapter 8. To use the modeling language, an editor for Software Manufacture Models is provided. In addition, an editor for the Software Manufacture Model Pattern language and an automated pattern matcher are also provided. The pattern matcher automatically identifies occurrences of patterns within Software Manufacture Models. Further, the thesis provides tool support for the assessment of the manifestation of an MDE setting's traits and thus on the potential to affect software development processes.

To gain further knowledge on the characteristics of MDE settings in practice, the analysis methods are applied to the case studies that were collected in the context of this thesis (see Chapter 9). As a result this thesis provides the additional insight that manifestations of MDE traits that imply constraints on software development processes occur in MDE settings in practice. However, these manifestations of MDE traits do not occur in all MDE settings. Similarly, occurrences of the proto-patterns are identified within these case studies. This shows that risks on changeability occur in practice as well as solutions to prevent these risks. Both results show that MDE settings in practice differ strongly in their impact on changeability and in the constraints they imply on software development processes. Subsequently, the Software Manufacture Model language and the analysis methods are evaluated on the basis of their application to the case studies from practice (see Chapter 10).

These analysis techniques can be applied during the set up of a software development projects. When different MDE settings are available, the results of the analysis can help to decide which MDE setting should be used. Further, if only one MDE setting is available the analysis might help to decide whether and how to evolve an MDE setting. Such an analysis can also influence the choice or tailoring of the software development process. Finally, the analysis of how an MDE setting influences changeability can be used during development or maintenance to make developers aware of risks.

1.4. Structure

The chapters of this thesis are now briefly outlined:

Chapter 2 This chapter provides foundations on concepts of model-driven engineering, model management, software development processes and productivity in software development. Literature that explains how MDE influences productivity is summarized. Further research methods are discussed. This chapter is partially based on [P3, P5], [P2], and [P4].

Chapter 3 In this chapter the conceptual domain of MDE settings is analyzed. A set of MDE traits is presented that can imply constraints of software development processes. Additional changeability concerns are presented which are associated with MDE settings. Finally, a taxonomy of possible changes and evolution steps of MDE settings is introduced. For each change type it is discussed how productivity aspects like changeability might be altered. This chapter is partially based on [P2], and [P4].

Chapter 4 This chapter introduces the studies that were performed in the context of this thesis, gives an overview about the captured data, and discusses threats to validity. This chapter is partially based on [P3], [P6], [P2], and [P4].

Chapter 5 In this chapter results from the studies are presented. This includes insights on the structure of MDE settings in practice, insights on the evolution of MDE settings in practice, as well as observations on the motivations and triggers of this evolution. This chapter is partially based on [P3] and [P4].

Chapter 6 This chapter discusses requirements for a modeling language to capture MDE settings, introduces the Software Manufacture Model language as well as the Software Manufacture Model Pattern language. This chapter is partially based on [P2] and [P6].

Chapter 7 This chapter discusses requirements for an analysis of MDE settings. Following, analysis methods to identify the MDE traits presented in Chapter 3 are presented. Further, four proto-patterns on changeability risks and evolution are presented. This chapter is partially based on [P2].

Chapter 8 This chapter introduces tool support for the modeling languages and analysis methods. This chapter is partially based on [P2].

Chapter 9 In this chapter the analysis techniques are applied to the case studies from the examples. It is discussed to what extent the analyzed MDE traits and proto-patterns can be found in these case studies. This chapter is partially based on [P2], [P6], [P4] and [P3].

Chapter 10 In this chapter the modeling language and the analysis techniques are evaluated on the basis of their application to the case studies. This chapter is partially based on [P6].

Chapter 11 In this chapter related work is summarized and discussed. This chapter is partially based on [P3], [P2], [P6], and [P4].

Chapter 12 Finally, the results of this thesis are summarized and directions for future work are outlined.

2. Preliminaries

In this chapter preliminaries and foundations that are used during this thesis are described. This includes following topics: model-driven engineering, software development processes, productivity, pattern, and empirical research methods. Further, a running example for this thesis is introduced. This chapter is partially based on [P5].

2.1. Model-Driven Engineering

One approach to cope with complexity of software is to use models to represent abstract views on a system. Depending on the captured aspect of the system under study, a model can be used to support different engineering tasks.

First, models can be used to support communication between different stakeholders and developers of the system. Therefore, implementation specific aspects of the system are abstracted away. This way the tasks of requirements engineering and validation can be supported by models. Second, models can be used as a basis for creation and verification of a system design. Models that are used for this purpose reflect different aspects of the structural and behavioral system design. Through the concept of abstraction, models can provide overviews on large and complex systems. In addition, different techniques, such as model checking, model-based testing, model simulation, and formal methods can be used to support quality assurance (e.g. to verify whether a system will meet required safety properties [80]). This way some errors in system design can be identified and corrected before the implementation starts. Finally, models that are used for system design can be a basis to support coordination and division of work within teams. Using these two benefits from models during software development is the basic idea of “Model-Based Software Engineering” (MBSE) [175].

When models are created to design and verify a system, they often already include design decisions. Thus, the third benefit from using models is that they can be basis for the (partial or complete) generation of source code. In some approaches program execution is even directly done by interpreting models. The main benefit when generating source code out of models is reuse. First a manual translation of the design from the models to the code might be an error prone task. When generating source code, this manual task is no longer necessary. This has the additional benefit that, assuming that the generator works correctly, model verifications remain valid for the code. Second, generators can encapsulate knowledge on how specific system aspects can be implemented optimally. This knowledge is then reused as a side effect of generation. In contrast to MBSE, techniques like “Model-Driven Architecture” (MDA)[154] and “Model-Driven Engineering” (MDE)[111] include the possibility to use models for generation of code. MDA is an OMG¹ standard and specifies three model types (computation independent model (CIM), platform independent model (PIM), and platform specific model (PSM)). The CIM is created first and then transformed (manually or automatically) into the PIM. The PIM is enriched with details for implementation of the system. Finally, the PIM is transformed to the PSM, which can be used for code generation. The transformation from PIM to PSM enriches the PSM with platform specific aspects of the implementation.

As introduced in [111], MDE has a wider scope than MDA. Thus, MDE does not specify what combination of models will be used during development. Although, the automation of code creation on the basis of models is sometimes seen as the core property of MDE [206], the terms MBE and MDE are often used as synonyms (e.g. as in [180]). Other common synonyms are “Model-Driven Software Engineering” (MDSE) (e.g. in [192]) and “Model-Driven Development” (MDD). Throughout this thesis, “Model-Driven Engineering” (MDE) refers to all these facets. Also MDA is seen a special form of MDE.

How the use of models within practice might differ is illustrated by Brown’s spectrum of model use (shown in Figure 2.1, the illustration bases on a diagram originally drawn by John Daniels [37]). On one end of the spectrum no models are used (code only). Alternatively, models might be used to visualize the structure of code, which can be beneficial for verification reasons (code visualization). In round

¹<http://www.omg.org/> (last access at November 9th, 2013)

2. Preliminaries

trip engineering changes are propagated between models and code. Then, code might be created out of models (model centric). Finally, at the other end of the spectrum only models are used during development (model only).

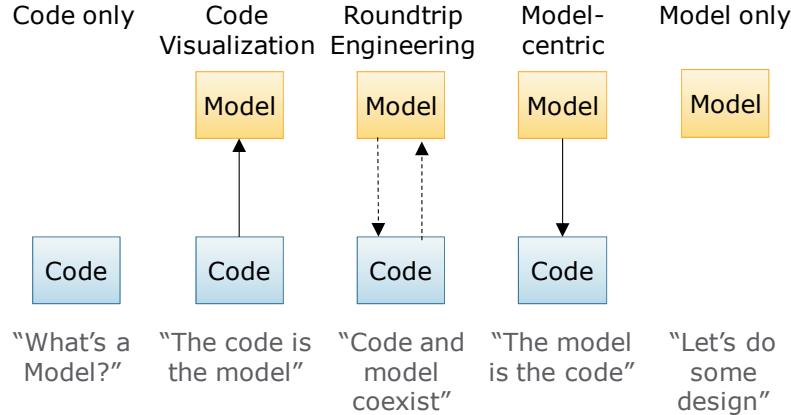


Figure 2.1.: Brown's spectrum of model use [37]

Languages MDE approaches often combine diverse languages, such as modeling languages (e.g. Unified Modeling Language (UML)[156], Business Process Model and Notation (BPMN) [157], and Systems Modeling Language (SysML) [3]), domain specific languages (DSLs) and domain specific modeling languages (DSMs) ([110], e.g. the DSM for development of Inventory Tracking Systems presented in [50]; an overview of different DSLs and DSMs in given in [200]), general purpose languages (GPLs) (e.g. Java [90]), scripting languages (e.g. JavaScript [212] or Python [165]), languages to support interface and protocol descriptions (e.g. SOAP [36] or WSDL [42]), languages to manage data base systems (e.g. structured query language (SQL) [35]), configuration files languages (often based on XML [215]), build script languages (e.g. Make [67] or Ant [14]), languages for layouts (e.g. style sheet languages like CSS [207]), languages for the specification of transformations (e.g. ATL [105] or XSLT [44]) languages for the specification of other languages (e.g. MOF [153], some XML schema languages as summarized in [149], or Backus-Naur Form (BNF) [13]) unstructured or structured natural languages (e.g. English within a requirements documentation), or simple ad hoc notations (e.g. sketches).

Two categorizations for these languages are interesting, when reasoning about MDE settings. First, languages differ in their expressiveness. On the one hand, there are general purpose languages, which can be used for specification of arbitrary programs. On the other hand, there are domain specific languages (DSL). The expressiveness of a DSL is restricted to a specific domain in favor of the ability to implement solutions for this domain in a more efficient manner (e.g. less effort for the description of the solution is required). These domains can be chosen with a focus on technical aspects (e.g. SQL) or with focus on the application domain. Although there is a multitude of languages in the first group (e.g. Make, CSS, ATL, or WSDL), the term DSL is sometimes used to refer to languages of the second group, only.

Second, it is differentiated between programming languages and modeling languages. Although most languages can be clearly assigned to the one or the other group, there is no ultimate differentiation of both groups. In most cases programming languages base on a grammar that describes the concrete syntax and is basis for parsing program files. In contrast, the abstract syntax of most modeling languages is specified using metamodels. The characteristic of models that is often used as distinction to program code is their graphical syntax. This graphical syntax should support a more intuitive recognition of complex problem and program structures, compared to the textual syntax of code. Although this differentiation between graphical and textual character fits for the most languages, there are also textual modeling languages (e.g. efactory²).

Similarly, there are further characteristics that differentiate most programming languages from most modeling languages. However, for each characteristic also exceptions exist. For example, the use of

²<http://code.google.com/a/eclipselabs.org/p/efactory/> (last access at November 9th, 2013)

models is most often motivated by the desire to gain a more abstract view on a system compared to code. However, declarative programming languages (especially declarative DSLs) can provide a very abstract view on the system, while UML class diagrams are sometimes used on a very low level of abstraction, e.g. to reflect the structure of object oriented program code. While programming languages are most often prescriptive, modeling languages can be descriptive and prescriptive. Similarly, while programming languages are most often executable (i.e. a compiler or interpreter is provided), there are modeling languages that are executable as well as modeling languages that are not executable. Often behavioral and structural aspects are split clearly between different diagram types in modeling languages (e.g. UML activity diagrams describe behavior, while UML class diagrams describe a structure). While in object oriented programming languages structure and behavior are intermixed within the same documents. However, there are also programming languages that abstract from behavior (e.g. in declarative languages, such as Prolog [45], only the problems are described). Finally, many of the languages listed above are neither clearly associated to the group of programming languages nor clearly associated to the group of modeling languages (e.g. WSDL).

To summarize, there are general purpose languages, general purpose modeling languages, domain specific languages, and domain specific modeling languages. All of these languages can occur within MDE settings. It can be expected that different languages have significantly different potential to support development of different systems and system aspects.

Automation Automation of development is in its different forms an important part of MDE. In [138] and [48] classifications of model transformations are presented. The terms *model transformation* and *model operation* are used to describe a multitude of automated operations, such as code generation (out of models), model-to-model transformations, model merge, or synchronization of models (e.g. to preserve consistency between two models). Following, a short introduction is given into the categories that are relevant for this thesis.

Amongst other categories, Mens et al. differentiates between endogenous transformations (between artifacts of the same language) and exogenous transformations (between artifacts of different languages) [138]. Further, Mens et al. differentiate between horizontal and vertical transformations. Input and output artifacts of a horizontal transformation are on the same level of abstraction. Such transformation might be used for refactoring. In contrast, input and output artifacts of a vertical transformation are on different levels of abstraction, i.e. the output artifact contains more details than the input artifact, or the other way around [138].

Czarnecki et al. further differentiate transformations, according to the output artifact. The output artifact might be created when the transformation is executed (new target), the output artifact might be one of input artifacts (in-place), a former version of the output artifact might be overwritten (destructive update), or a former version of the output artifact might be extended (extension only) [48]. Finally, Czarnecki et al. differentiate between unidirectional and bidirectional transformations. Unidirectional transformations can only transform the input artifacts into the output artifacts. In comparison, bidirectional transformations can be executed in both directions (i.e. the role of input and output artifacts can be exchanged) [48].

Such automations of development might be implemented with a GPL like Java or with a DSL that is specifically created for the implementation of model transformations (e.g. Xpand³ (for code generation out of models), Triple-Graph Grammars (TGGs) for model to model transformations [89], ATLAS Transformation Language (ATL) [105], or Query View Transformation (MOF QVT) [155]).

Transformations and generators can be very complex. This is especially the case when they are implemented, such that they can handle all possible valid instances on one or more languages as input. To support a more modular design of transformations, frameworks like UniTI [202] support the definitions and automated execution of transformation chains. These transformation chains can be compared to pipe and filter systems. The output artifacts of one transformation are input for the next transformation. Other approaches allow in addition so called context compositions [A11]. There, a transformation between two artifacts is a composition of multiple transformations that focus on different specific parts of the input and output artifact. Also the QVT standard supports context compositions [155]. Most of these approaches allow that the composed transformations are implemented in different languages.

³<http://wiki.eclipse.org/Xpand> (last access at November 9th, 2013)

2. Preliminaries

Evolution Lehman's laws [124] as well as taxonomies on software change [139] led to a common understanding that software is subject to change and evolution.

However, as Favre stated in [63] languages are subject to evolution, too. This does not only hold for, metamodels, but also for metamodels such as MOF. Since then, many forms of evolution were studied. For example, it was studies how languages evolve (e.g. [87]), how language evolution affects evolution of software, including the topic of co-evolution of metamodels and models (e.g. [39] or [43]), or how evolution of languages entails evolution of automations, like transformations or generations (e.g. [126] or [216]).

2.1.1. Comparing Development with and without MDE

To investigate the benefits of MDE it is necessary to understand its differences and commonalities with traditional code-centric software development. Therefore, the automation of implementation tasks (e.g. through generation of code) as well as the use of languages is discussed in the following.

Automation of Development

As mentioned above, one of the core ideas of MDE is to generate code on the basis of models. However, the idea to automatically create code has a long history. Already in 1958 the vision of *automatic programming* was established. In this vision, end users create a brief description of their requirements on a system and the automatic programming system generates an effective program that fulfills these requirements. An automatic programming system has three features: it can be used by the end users, it is general purpose, and it is fully automated [167].

Twenty years later, in 1988, Rich et al. summarize what has been achieved so far and discuss to what extent the vision of automatic programming is reachable [167]. They argue that the three features cannot be reached together and found that each successful automated programming system focuses on two of the three features, only. Consequently, Rich et al. split automated programming systems into three groups. *Bottom up* approaches neglect the goal to be usable by end-users. These approaches aim at increasing the degree of automation (e.g. substituting machine-level languages by high-level languages). *Narrow domain* approaches neglect the goal that the automatic programming system is general purpose (e.g. database query systems). Finally, *assistant* approaches neglect the goal that the programming system is fully automated. Tools provide support for e.g. high level design manipulation, without subsequent automated generation of code (e.g. the tool Excelerator from Index Technology Corporation that supported structured analysis and design methodologies).

When comparing automated programming with MDE, the similarities are obvious. The three features of automated programming (end users as developers, general purpose, and full automation) are also part of the MDE vision. Similar to automated programming also MDE approaches might be characterized by what features are in focus and what features are neglected. For example, general purpose modeling languages, like UML [156], focus on being general purpose. Especially, UML is in between a *bottom up* and an *assistant* approach in automated programming. On the one hand, UML is meant to support high-level system design, but (without using the concept of UML profiles) does not reach a level, where it can be used by end users to specify the required system. On the other hand, UML is used for partial generation of code frames, but cannot be used for automatic generation of the whole code (again, without using the concept of UML profiles). When using UML profiles, both features can be improved, on the costs that the UML profile is no longer general purpose.

DSLs and DSMs explicitly neglect the goal to be general purpose and are therefore comparable to *narrow domain* approaches in automatic programming. In fact, *narrow domain* approaches like database query systems are often considered as examples for DSLs, today. Finally, the idea of "Model-Based Software Engineering" (MBSE) as introduced in [175] is comparable to *assistant* approaches in automatic programming, where the automated creation of code is not in focus.

As indicated above, also high-level programming languages base on automation (bottom up approach), which is well known as the process of compilation and linking. This process, to automatically translate high-level programming languages to machine-level code, is also called software manufacture ([123, 34]). Although this term fell into oblivion, the most used software manufacture tool, Make [67], is still known by nearly every software developer, today. Software manufacture tools are often replaced by build script languages like Ant [14].

Build script languages as well as software manufacture tools fulfill a similar role like transformation chains in MDE. Although in some MDE approaches the results of a transformation chain are further manipulated by developers, researchers from both areas become aware of the similarities, nowadays. For example, Jezequel et al. discuss what both communities might learn from each other [102].

Use of Languages

Besides a comparison of the automation aspects, it is also worth to have a more detailed comparison of the language use. As discussed above in MDE often a combination of multiple languages is used. These languages are not necessarily only modeling languages, but often also normal programming languages or XML-based DSLs. Actually, “non-MDE development” today is not necessarily much different. Often different languages are combined and often these languages are not only traditional code languages.

Hutchinson et al. provide in [99] a transcript, where such a development setting is shortly described. Languages that need to be handled in this setting include B+ trees, HTML, and Java. In addition, parts of the user interface seem to be implemented with a help of a graphical tool. Also Pfeiffer et al. [162] collected a set of amazing practical examples how language combinations in software development can look like. For example, they report that for the development of the open-source ERP system OFBiz⁴ more than 30 languages are in use. This includes GPLs and DSLs as well. They also extract a simpler example from the open-source bug-tracking system JTrac. This example shows that development with multiple languages even happens, when no modeling language is involved. The three languages used for JTrac are HTML, Java, and a simple properties file. Pfeiffer et al. [162] call software systems that are developed using different languages “multi-language software systems” (MLSS). They further substantiate the commonness of MLSS by citing a survey performed by Zend Technologies Ltd. [220], where it was found that PHP developers often use languages like JavaScript, Java, or C along with PHP. Thus, there are many development settings that are – from the perspective of used languages – neither pure MDE nor code-centric.

Summary

Considering both aspects, automation and used languages, it becomes clear that MDE and non-MDE approaches have much in common. Whether developers call their approach model-driven or not, often seems to be a subjective decision. Obviously there are many approaches that are clearly code-centric (e.g. when only java is used) or clearly model-driven (e.g. when different models are used generate parts of the source code). However, there is a broad spectrum of approaches between code-centric and model-driven, like the example that is discussed in [99]. Concluding, MDE can be seen as a part of an ongoing process towards a simpler and more productive software development.

However, there is also an aspect that is special to MDE and only seldom in non-MDE approaches. This is the fact that, within MDE, manual and automated tasks can be intermixed, i.e. automatically created artifacts are manipulated manually. Further, due to the graphical nature of models, they have, compared to code, a greater potential to support communication.

Due to the commonalities, many techniques to improve development that are developed within the MDE community, aim at supporting non-MDE development, too. For example, research in model management on megamodels [19] and MLSS [162] is not exclusively motivated with MDE. Similarly, in this thesis the term model-driven engineering is used to refer to the whole spectrum of forms of development, where multiple (modeling) languages or automation of development tasks are used.

2.1.2. Model Management and Megamodels

Although MDE approaches can be simple for small DSMs as described in the example above, they can get very complex for other examples. In such cases, developers have to deal with a multitude of artifacts that are formulated in different languages and have the most different relations among each other.

The aim of model management is to support developers in coping with this additional complexity. The main tool of model management is the megamodel or the related concept “macromodel”.

As summarized in [P5], the definitions of megamodels and macromodels differ slightly. For example, Bézivin is one of the first founders of the term megamodel. His view of the term megamodel grew

⁴<http://ofbiz.apache.org> (last access at November 9th, 2013)

2. Preliminaries

during the recent years. The first definition was given in the year 2003: “*A megamodel is a model with other models as elements*” [27]. In 2007 this definition is further complemented: “*A megamodel contains relationships between models*” [18].

In contrast, Salay introduces macromodels, which fit the definition of megamodels, but have a different intention. A first definition is provided in 2007: “*A macromodel is a graphical model whose elements denote models and whose edges denote model relations*” [173]. Later, he defines the term macromodel as: “*A macromodel consists of elements denoting models and links denoting intended relationships between these models with their internal details abstracted away*” [174].

Though definitions of Bézivin, Salay, and many other authors differ slightly, they have a common core. This core definition was extracted in [P5]: A megamodel is “*a model that contains models and relations between them*”. In some approaches megamodels are hierarchical, e.g. models can contain models and relations can be context of other relations. In a broader sense, models that are contained by a megamodel can be of all kinds of artifacts, such as requirements specification that are written in structured natural language, models of the system design, or even source code files.

In Figure 2.2 an extensible metamodel for megamodels is shown [P5].

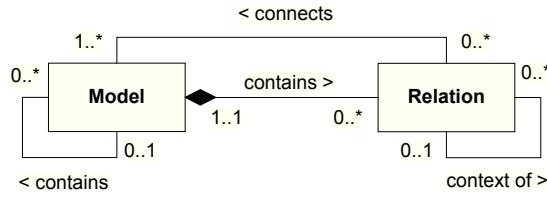


Figure 2.2.: Core metamodel for megamodels introduced in [P5] (as UML class diagram [156]).

There are basically three types of relations that are captured only within megamodels. First, there is the “*overlap*”-relation that indicates that two models have overlapping content. This relation is, for example, used by Salay [173, 174] and Perovich [160] to indicate that two models show different views of the same system. Seibel [178] uses this relation as basis for identifying consistency and inconsistency between two models. Bézivin uses “*overlap*”-relations in [18], where megamodels are used to handle for traceability issues. Second, the “*model operation*”-relation is a representative of a model operation that was, can be, or will be performed with models as input and output. For example, Favre [62, 64, 66] introduces the “*isTransformedIn*” relation, which indicates that one model was transformed to another model. Similarly, Bézivin [27] and Perovich [160] support this relation type. Seibel [178] uses the “*model operation*”-relation for automatically restoring the consistency between two models.

The last type of relation is called “*static*”-relation. Examples for “*static*”-relations are “*conformsTo*”, which represents the relation between a model and a reference model, or “*contains*”, which indicates that a model is contained by another model. Another example is the “*represents*” relation between a models and the represented system, which is used by Favre [62, 64, 66]. Most approaches, like Bézivin [27] and Perovich [160], work with the “*conformsTo*” relation. Seibel [178] uses the “*isOfType*” relation for links. Also Salay [173, 174] uses to show the “*instanceOf*” relation in his example models.

Reasons for the use of megamodels in model management are diverse. The first motivation is to make the relations between existing artifacts explicit, such that developers can get and maintain an overview during a project (e.g. in [27, 160], or [178]). A megamodel helps to trace, which artifacts are created on the basis of which other artifacts. Second, megamodels are used to declare what artifacts and artifact relations should be created during development. In such cases the megamodel of the current situation is compared to the target megamodel (e.g. in [173, 174]). A third motivation is to support automatic execution of transformation chains (e.g. in [52]), or to automatically maintain consistency between artifacts (e.g. in [178]).

2.2. Productivity

In this section, different general and MDE specific perspectives on *productivity* are introduced. Subsequently, the interrelation of the different perspectives is shortly discussed.

2.2.1. Influences on Productivity

A main motivation in research on software development techniques, technologies, and languages is the desire to enhance the productivity of a software company. As collected in [16, 17] Balzert's Lehrbuch der Softwaretechnik much research was done to understand and define the term productivity. This includes the exploration of attributes, such as quality and quantity of software, development effort, and development time.

In [185] Sneed presents his devil's square (Figure 2.3) to illustrate that it is hard to enhance productivity in the short term. The devil's square illustrates productivity as a plane, spanned by the parameters quality, quantity, costs, and time. Sneed assumes that the surface area of the plane does not change easily. For example, actions applied to enhance quantity of the produced software with stable time and costs will reduce quality of the product. Enhancing quantity of the software without decreasing quality can – in short term – only be reached by adding costs and/or time.

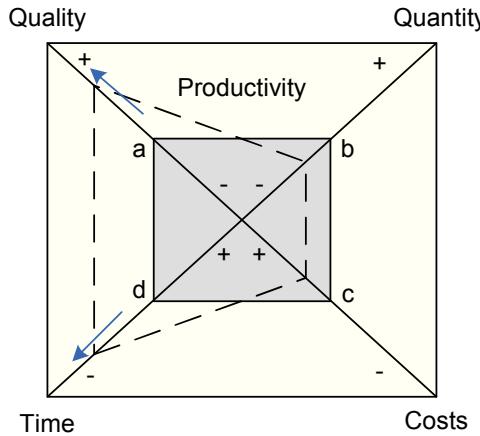


Figure 2.3.: Sneed's devil's square after [185]

Consequently, further work was done, to determine how productivity can be enhanced, i.e. how one of the four parameters can be optimized, while keeping the others stable. As cited in [17], Maxwell et al. sums up that intensive use of tools and modern methods can increase productivity [134]. Further, research was done on whether reuse of software can reduce required time and resources without reducing quality and quantity of produced software [91]. Boehm presents several actions to enhance productivity in [31]. Finally, Balzert subsumes that process improvements might improve productivity [16].

As indicated with Sneed's devil's square, productivity is influenced by other factors, which are presented in [17] on the basis of Basili (see Figure 2.4). Productivity is influenced by value of the produced software (i.e. quality and quantity or the software) and costs of the software (which is among other things influenced by development time and personnel costs). Further, complexity of the software to be build influences the level of difficulty, which further influences the costs.

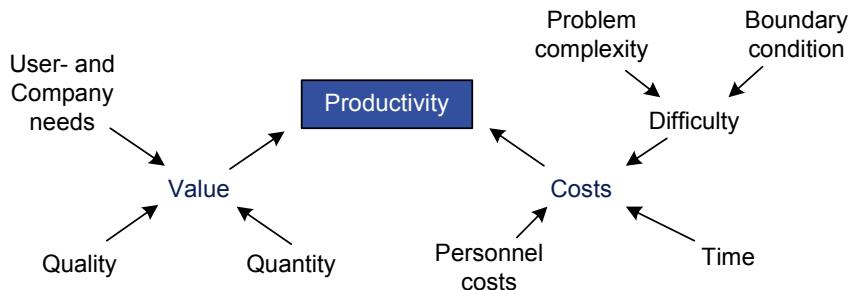


Figure 2.4.: Influencing factors on productivity after Basili (cited from Balzert [17]).

2. Preliminaries

Balzert provides a more detailed list of factors that can influence productivity (referred to as productivity influencing factors in the following) and splits them into five groups [17]: organization (e.g. corporate culture), management (e.g. working environment or team size), employees (e.g. salary, professional experience, or experience in application domain), product influences (e.g. quality, size, requirements on operating times, memory limitations, or complexity of the product), and process influences. Process influences are restrictions on development time (or time to market) [134], duration of the project [134], stable requirements [31], technological support [187] (addressed by, e.g. tool integration [31]), methods or process quality [187], amount of rework (addressed by, e.g. front-end aids, information hiding, and modern programming practices [31]), used programming languages, and, finally, reuse (addressed by, e.g. libraries, generators, automated documentation and quality assurance, or automated programming [31]).

As already mentioned in the introduction, different techniques and practices can be applied to affect these productivity influencing factors. For illustration, techniques and practices that can be used to affect factors in the group of process influences can be split into three groups here: the technical setting, which covers used languages (including DSLs), compilers, interpreters, or model-driven engineering, the software development process (e.g. RUP [100]), and the applied paradigms and programming practices (e.g. Service-Oriented Architecture or Design Pattern) (summarized in Figure 2.5).

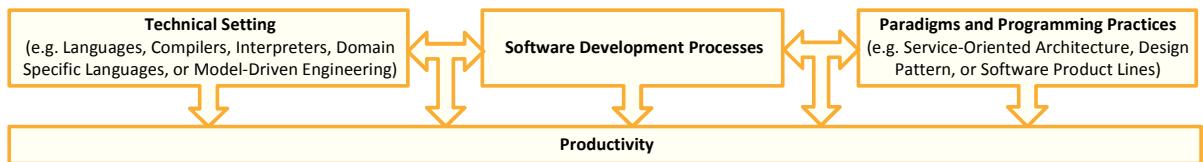


Figure 2.5.: Groups of techniques and practices that are used to affect productivity via process influences.

2.2.2. Internal Quality Attributes

Software quality attributes represent another perspective on productivity. The ISO standard defines diverse software quality attributes, for example, reliability or changeability [1].

It is differentiated between external software quality attributes and internal software quality attributes. While external software quality attributes, such as reliability and efficiency describe the quality of a software product that can be experienced by the customers and users, internal software quality attributes are independent of the software execution.

Important internal quality attributes are reusability, maintainability, changeability, and portability.

As McIlroy's law tells us: "Software reuse reduces cycle time and increases productivity and quality" [56]. Thus, reuse mainly helps to reduce development time, without reducing quality, quantity, or enhancing costs. Further, reuse can help to improve quality, since each reuse increases the probability that faults in reused models or code parts are discovered. Consequently, reused model and code parts are more mature.

Due to the need that software can be used over a long time, main aspects of productivity are maintainability and changeability. Thus, reducing time and costs for changes and maintenance is important for productivity. Further, maintenance helps to preserve quality of a software product. In addition, *customizability*, a special form of changeability, helps to enhance the validity of the software for different customers. Portability enhances productivity, if the same system has to be built on different platforms, since costs and time can be saved.

2.2.3. MDE Mechanisms

The structure of MDE approaches are in main focus within this thesis. The introduction of MDE was associated with many well founded hopes concerning improvement of quality, quantity, costs, or time and thus improvement of productivity. There are different arguments why MDE (or MDA in particular) should break through the devil's square, by improving one parameter without worsen the others.

Model-driven engineering comes up with three mechanisms, which are *separation of concerns*, *automation*, and *abstraction*. For example, in the OMG's MDA standard [154] the separated concerns are business aspects, computational aspects, and platform aspects. Separation of concerns is reached by using different models/artifacts for expressing the different concerns. Abstraction is often associated with easing the definition of certain aspects. These three concepts influence productivity by influencing internal software quality attributes, e.g. *reusability*, *maintainability*, *changeability*, and *portability*. In literature theoretical considerations to argue why the MDE concepts separation of concerns, automation, and abstraction influence software quality attributes can be found. In the following, some examples are given.

Stahl et al. [188] argue that *reusability* can be supported by a meaningful separation of concerns, since a model that is separated from certain aspects can be reused if these aspects change. For example, a main motivation of OMG's MDA [154] is the reuse of the platform independent model (PIM) for building the software for multiple different platforms, since platform aspects are not specified in the PIM. They further argue that automation leads to reuse, since implementation knowledge is reused in automated transformations or generations. Gruhn [92] argues that raising the level of abstraction on a domain specific layer leads, in addition to reuse of technical knowledge, to reuse of domain knowledge.

Portability is one of the main declared goals of OMG's MDA [154]. Kleppe et al. [115] argue that separation of concerns and automation enable changes of the underlying platform.

A declared goal of OMG's MDA [154] is *interoperability*. Kleppe et al. [115] argue that this is addressed in MDA since platform aspects are automatically generated by transformations, and thus automated generation of platform bridges, such as “PSM bridges” or “Code bridges”, is possible on the same information.

Further, Kleppe et al. [115] and Stahl et al. [188] argue that abstraction directly helps to handle *complexity* and they argue that automation has a direct influence on the time required, as each generated part of code, does not need to be written by hand [115, 188]. Finally, as Kelley et al. [110] argue, quality is supported, since DSLs reduce through abstraction the possible mistakes that can be made during implementation.

2.2.4. Interrelation of Productivity Terminologies used in this Thesis

Following, it is introduces how the different perspectives on productivity are interrelated from the viewpoint of this thesis.

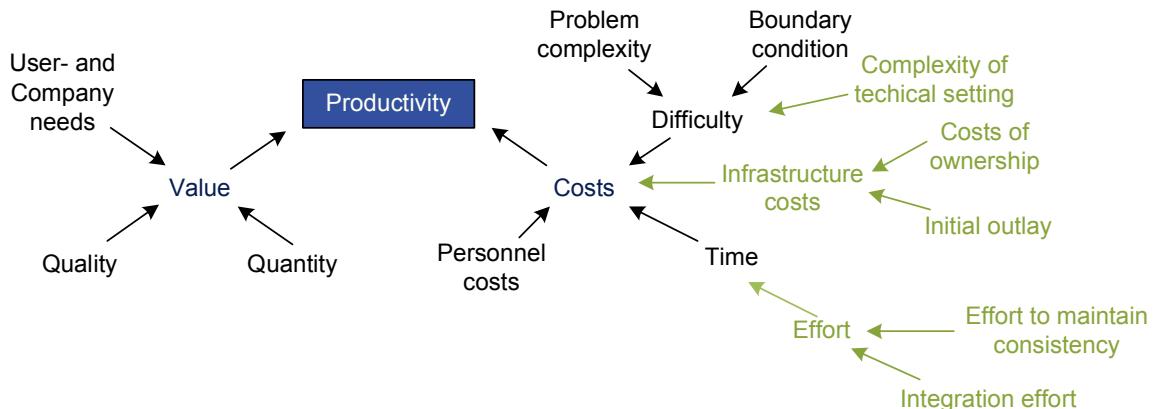


Figure 2.6.: Influencing factors after Basili (after Balzert [17]), extended here with exemplary additional productivity influencing factors that become relevant when MDE is applied (extension emphasized in green).

First of all, the influencing factors sketched by Basili (Figure 2.4) might be supplemented by further factors that become important when studying MDE settings. In Figure 2.6 some examples are shown. MDE comes with a more complex tool setting. Especially, in the case of DSLs, transformations and generators need to be maintained. Thus, the costs of the infrastructure (the initial outlay and the costs

2. Preliminaries

of ownership) become important. Further, the combined use of different languages makes the effort to maintain consistency between different artifacts important. Finally, the complexity of the technical setting (i.e. of the concrete MDE approach) is a factor that influences the daily work of developers and with it the difficulty of development. *Techniques and practices*, such as the used MDE setting or process affect these productivity influencing factors.

Internal software quality attributes are considered in this thesis as views on productivity. These views represent manifestations of productivity for different use cases (e.g. changeability can be seen as the productivity in case of changes). Thus, internal software quality attributes, such as changeability, are impacted by the productivity influencing factors, too. Consequently, MDE settings and the used software development process affect the internal software quality attributes, too.

In literature, all these perspectives (productivity influencing factors, mechanisms of the technical setting, like abstraction and automation, and internal software quality attributes) are used to describe effects on the productivity. For example, these effects are not always traced back to or discussed on the level of the corresponding productivity influencing factors. Instead, effects are often described more abstractly as effects on different internal software quality attributes. For example, as summarized in Section 2.2.3 the effects of MDE are often described as effects on changeability or portability. Sometimes it is even considered as sufficient, when it is described how the manifestation of an MDE mechanism changes. For example, it is common sense that an increased degree of automation is associated to a positive tendency for the productivity.

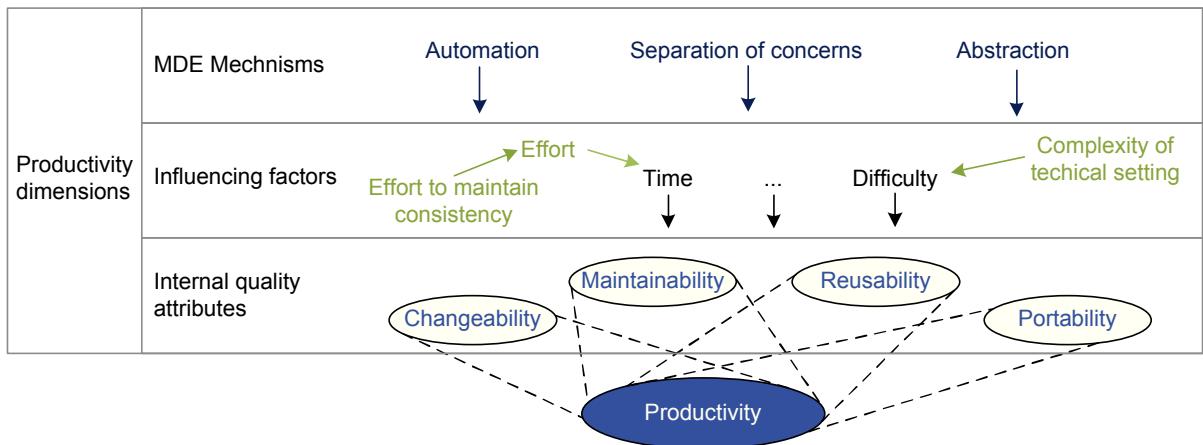


Figure 2.7.: The introduced term “*productivity dimensions*” covers productivity influencing factors, MDE mechanisms or mechanisms of the technical setting, and internal software quality attributes.

To capture this diversity, the term *productivity dimension* is introduced here as embracing concept. A *productivity dimension* is a property whose concrete manifestation can be associated to a positive or negative impact on productivity. Thus, as summarized in Figure 2.7, all productivity influencing factors, the MDE mechanisms or mechanisms of the technical setting, and internal software quality attributes are referred to as *productivity dimensions* within this thesis .

2.3. Software Development Processes

Software development processes are used to organize development of software. Processes focus on specifying an order of development activities or phases. Further, documents and artifacts that are used and produced in these phases are specified. Additional aspects that are addressed by software development processes are, for example, the way teams interact (e.g. division of labor or frequency of team meetings) or what quality assurance activities are performed. For example, an explicit goal of RUP [118] is to “direct the tasks of individual developers and the team as a whole”, whereas Scrum [176] limits the scrum master’s authority concerning the question how members of the development team should interact. Likewise, the V-Modell XT [82] and RUP [118] prescribe and schedule quality assurance activities.

Most modern software development processes define iterations of evolution cycles for the development of the software. These cycles have diverse names (e.g. sprint or iteration) and are of different durations. For example, iterations in the spiral model [32] usually last longer than a *sprint* in Scrum [176].

In general, it is differentiated between agile and “rich” software development processes. Following the manifesto for agile software development⁵, agile processes often base on shorter cycles and smaller teams. The main focus of the processes is to support customer collaboration, the ability to respond to changes, and to provide working software in early stages of development, already. In addition, these processes are often accompanied by techniques that aim at guiding interaction of developers, e.g. pair-programming. Examples for agile processes are Scrum [176] or XP [23]. Processes that are not agile (sometimes also called “rich” processes) often have a stronger focus on planning and documentation. Examples for rich processes are V-Modell XT [82] or the Rational Unified Process (RUP) [100]. As Sommerville concludes in [186] the question what process is best depends on the organization, the character of the developed software, and the skills of the developers. For example, agile processes are often not appropriate in safety critical projects or when a huge team of developers needs to be coordinated. In contrast, many small projects can benefit from agile approaches.

Processes, as they are described in the specification, are often not directly applied. First, processes are most often tailored. *Process tailoring* is the adaptation of a process to the specific needs of an organization, the environment, or project needs [21]. As collected in a survey of literature about software process tailoring by Kalus and Kuhrmann, process tailoring can affect the emphasis that is put on different actions in the project, such as integration and testing [106]. Other tailoring activities can concern the organization of the project (including communication and the number of iterations) or the knowledge management. However, in the survey the authors conclude that is not yet understood what concrete tailoring actions can be applied when certain tailoring criteria are fulfilled.

Further, processes of a company – even if no standard process is applied – change over time, are refined, and can mature [84]. The Capability Maturity Model (CMM [159]) specifies five levels of process maturity, from the use of ad hoc processes to a level, where the process is continuously improved. More mature processes come along with a better predictability of performance of development.

2.4. Patterns

Patterns are a commonly used tool to capture and communicate knowledge, experiences, and solutions in software engineering. The probably best known examples for patterns are the design patterns on software architecture that were presented by Gamma et al. [86]. However, there are not only patterns that concern the architecture of software, but also patterns for many other areas of software development and maintenance. Some of them are collected in a pattern almanac [169]. For example, a pattern that concerns the handling of automation during development is the generation gap pattern [78].

Besides “positive” patterns that describe solutions, there are also so called *anti-patterns* that capture situations that are known to be risky or problematic [7, 38]. For example, a well known anti-pattern is the god class (as discussed in [54]).

Since patterns are meant to represent experience and praxis relevant situations, the creation of a pattern does not only consist of describing a structure and the associated solutions or risks. It is necessary that a pattern is recurring. Consequently, there is a rule-of-thumb that at least three occurrences of a pattern should be documented, before this pattern can be called a pattern. As long as this rule is not fulfilled a pattern is called *proto-pattern* [7]. Sometimes, even more rigorous rules are applied. For example, in some communities there are complex review processes, before a proto-pattern is accepted as pattern. For this reason, this thesis only introduced proto-patterns and discusses their potential to be accepted as patterns, on the basis of the identified occurrences.

2.5. Empirical Research Methods

The process of knowledge building requires data from practice to recognize phenomena, to build hypothesis to explain phenomena, or to evaluate such hypotheses. The required data is captured in empirical studies. In this section, a short overview about different techniques and categorizations of empirical studies is given. It is described which techniques are used within this thesis.

⁵<http://www.agilemanifesto.org> (last access at November 9th, 2013)

2. Preliminaries

Experiments vs. observational studies Basically, there are experiments and observational studies [20]. Within experiments researchers manipulate specific variables in order to measure the effects of these manipulations. In contrast, during observational study, the researchers do not apply treatments or manipulations, but derive their insights based on what can be observed. Observational studies that capture one or multiple projects from practice are called case study or field study [219][20]. Yin defines a case study as “an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” [217]. Within this thesis case study research is applied.

Qualitative vs. quantitative data The second important differentiation is made between qualitative and quantitative data. While, quantitative data can be used to reason about frequencies of phenomena or to prove cause-effect relationships, qualitative data is rather useful to study complex phenomena or to build hypotheses. Studies can contain qualitative as well as quantitative aspects at once [22]. Seaman states that the question whether data is qualitative or quantitative, can neither be answered based on the method that was used to capture the data, nor on the objectivity or subjectivity of the data. Instead, these terms describe the representation of data (mainly numerical in case of quantitative data and mainly textual in case of qualitative data) and with it the possible kinds of analyses that can be done [177]. Also other opinions on this differentiation can be found. For example, in [20] Basili describes quantitative data as objective and qualitative data as subjective. However, this thesis bases on Seaman’s understanding, since this view fits to the fact that qualitative as well as quantitative can be captured in semi-structured interviews.

Data collection The collection of data can happen in different forms. While questionnaires allow capturing and analyzing big amounts of data, interviews are more costly. However, it is difficult to capture qualitative data within questionnaires, since only open text fields might be used for that. In contrast, interviews can be used for capturing both qualitative and quantitative data. Interviews can be structured, semi-structured, or unstructured [177]. While structured interviews allow capturing quantitative data, unstructured interviews are most appropriate to capture qualitative data. In practice, often semi-structured interviews are used to combine advantages of both approaches [177]. This makes it feasible to capture quantitative data about the parts of the phenomena that are already understood, while maintaining the possibility to collect qualitative insights.

Consequently, interviews are the appropriate choice when complex phenomena need to be studied [214]. Since this is the case in this thesis, interviews were used.

Study results Depending on the type of results that are aim of a study, the data is analyzed differently. There are different classifications for study goals and results. For example, Basili distinguished between three aims for a study: a study can be descriptive (i.e. describe observed phenomena), correlational (in order to identify correlations between different variables), or focus on cause-effect relations between variables [20]. Seaman presents a classification that is orthogonal to Basili’s and simply distinguishes between two goals for empirical studies: generation of a theory or hypothesis and confirmation of a hypothesis [177].

In contrast to this focus on study goals, Shaw transfers a classification for study results from human computer interaction research in [181]. She differentiates between *findings*, *observations*, and *rules of thumb*: *findings* are “well-established scientific truths, judged by truthfulness and rigor”, *observations* are “reports on actual phenomena, judged by interestingness”, and *rules of thumb* are “generalizations, signed by their author but perhaps incompletely supported by data, judged by usefulness” [181].

Within this thesis the classification from Shaw is used to categorize results.

Data analysis Techniques to create hypothesis or systematically collect observations on the basis of empirical data are similar. Two methods that can be applied to gain hypothesis or observations are the *constant comparison method* [177] and *grounded theory* [46]. The constant comparison method is applied by a systematic annotation of interview records with codes. These codes might stem from an initial list of codes (preformed codes) or are created during the examination of the records [177]. After the annotation, parts of records with the same codes can be compared. Codes might be further refined. Finally, the comparison of the annotated code parts can lead to the identification of phenomena that

are partly described by multiple interviewees. Grounded theory is similar to the constant comparison method, but includes an additional round of feedback with interviewees or experts. These interviewees or experts are asked to confirm or refuse the made observations. Both techniques are used in this thesis.

In order to increase the probability that interesting observations can be made or new hypothesis can be formulated, studies with exploratory character can be applied. Such exploratory studies allow learning about a problem domain that is not well known. Mostly, exploratory studies are descriptive case studies or field studies. The researcher does not only focus on capturing a predefined set of data, only. For example, within interviews open questions are used to create an opportunity for interviewees to report on phenomena and experiences of which the interviewer is not aware. For this reason most studies that aim on hypothesis finding or collecting of observations collect and work with qualitative data. Within this thesis the first study was executed as exploratory study, which led to the motivation for the further research on evolution of MDE settings.

In contrast to studies that aim at formulation of hypothesis or observation making, quantitative data is used to confirm hypotheses or establish and evaluate findings. It is possible to prepare aspects of qualitative data for quantitative analysis. Seaman calls this process *coding* [177]. For example, for a textual list of tools that are used within a project coding can be used to extract the number of tools, which is quantitative data. Within this thesis this quantification shall be applied to extract information about the frequency of structural evolution steps on the basis of literature reports.

Validity discussion When performing studies or experiments, threats to validity need to be taken into account. Depending on the categorization, two or four types of threats to validity are distinguished. The simple categorization is a distinction between internal and external validity (as in [132]). Internal validity deals with whether the research design can be used to exclude alternative explanations or at least make them implausible [132]. In contrast, external validity concerns whether and to which degree the results can be generalized to other conditions [132].

An alternative categorization is presented by Wohlin et al. [214]. Internal validity is split into conclusion validity and internal validity, while external validity is split into construct validity and external validity. In this case, conclusion validity deals with the statistical significance of the relationship between a treatment and the outcome, while internal validity deals with the causal relationship between treatment and outcome (e.g. unobserved factors might influence the outcome) [214]. Further, construct validity deals with the question whether the results can be generalized to the concepts of the experiment [214]. For example, this concerns whether the participants in a study behave as they would normally (e.g. whether the behavior that is intended to be observed can actually be observed). Another question is whether sufficient characteristics of a concept are measured or implemented to represent it appropriately. Finally, external validity deals with whether and how far the results can be generalized to industrial practice [214]. However, typically, not all threats to the different types of validity are applicable to each study. For example, threats to internal validity are specific for experiments, but not for the studies presented in Chapter 4.

Within this thesis the forms of validity that are introduced by Wohlin et al. [214] are used to discuss and design validity of the study and the results. Since observational studies are performed within this thesis, only the three forms construction validity, external validity, and conclusion validity shall be discussed in this thesis.

Perry et al. [161] list different techniques to increase the validity of study results. One example is the *meta-analysis* [161]. The basic idea of this approach is to combine data from different studies. The more general principle of combining different evidences is also called *triangulation*.

Within this thesis a meta study is performed to gain information about the existence of structural evolution. To gain further information about the commonness of this phenomenon, the results of the meta study and records from case studies shall be triangulated.

Summary Subsuming there are diverse forms of empirical studies that can be used for different reasons in research. In Chapter 4 the studies that were performed in context of this thesis are introduced in detail.

2.6. Example MDE setting

Within this thesis mainly examples from practice are used to demonstrate the introduced classifications, modeling language, and analysis techniques. For example the analysis techniques in Chapter 7 will partially be explained on the example of one of the collected case studies from SAP.

The example of the generation of java code with EMF is used in multiple chapters within this thesis as a running example (e.g. in Chapters 3, 6, 8, and 9). This **EMF example** is chosen, since it represents a relatively simple MDE setting and is, as an open source project, a commonly accessible case study⁶.

In this section, the EMF example is introduced. As summarized in Figure 2.8, there is the activity “create.ecore model”, which is a manual activity to create an “ecore model”. This “ecore model” is consumed by the automated activity “create.EMF generator model”, which generates an “EMF generator model”. This artifact can be manipulated in the manual activity “manipulate.genmodel”. Together with the “ecore model” the “EMF generator model” is consumed by the automated activity “generate.java code”. This activity produces “interface”s, “adapter factories”, and “interface implementations”. Activity “generate.java code” takes old versions from “interface implementations” into account, when the activity is executed a second time. Each generated “interface implementation” can be further manipulated in the manual activity “manipulate.implementation”.

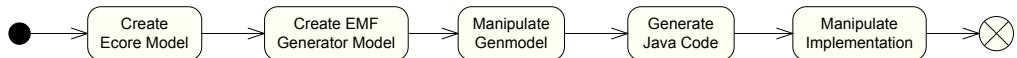


Figure 2.8.: Activities within the *EMF example*, illustrated as simple UML activity diagram [156].

The EMF example is mainly used in this original form and partly in a simplified version. However, in order to explain the automated derivation of activity orders in Section 8.3 the example is enriched with three further activities.

As illustrated in Figure 2.9, these activities are “create.genmodel”, as an alternative activity to create the artifact “EMF generator model”, “create.documentation”, is a manual activity for the creation of a documentation, and “merge.into” is a semi-automated activity that can be used to insert the “ecore model” into the documentation.

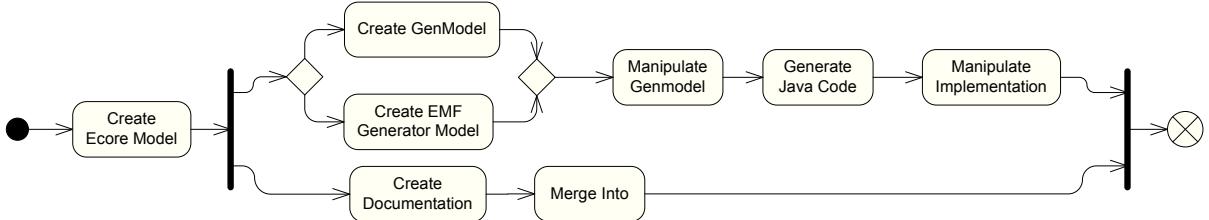


Figure 2.9.: Activities of the *extended EMF example*, illustrated as simple UML activity diagram [156].

⁶For a description of the EMF generation, see the Tutorial: <http://www.vogella.de/articles/EclipseEMF/article.html> (last access at November 9th, 2013)

Part II.

MDE in Practice

3. Analytical Insights

As summarized in Figure 3.1 this research focus of this thesis concerns three aspects: the question how MDE settings influence changeability, the question how MDE settings are interrelated with software development processes, and the question whether and how MDE settings evolve in practice.

In this chapter, these three questions are theoretically approached by analyzing the respective problem domains. As a first step, the concept “MDE setting” is introduced in Section 3.1. Afterwards the three aspects are examined successively. It is analyzed how MDE settings can impact changeability. Then, the interrelation between MDE settings and software development processes is investigated. Subsequently, it is theoretically examined how MDE settings might evolve and how this evolution can influence an MDE setting’s characteristics (i.e. the impact on changeability and the interrelation to software development processes). Based on this theoretical analysis, hypotheses on the actual manifestation of MDE settings in practice are formulated in Section 3.5.

This chapter is partially based on [P2] and [P4].

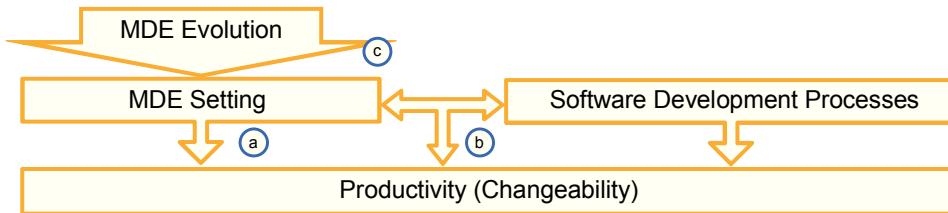


Figure 3.1.: Overview of research focus on MDE settings

3.1. MDE Settings

Brown’s modeling spectrum (introduced in Section 2.1) provides a useful taxonomy to understand motivations for the use of models. However, within a project multiple or all of the different uses of models can occur. For example, one model might be used to support communication with a stakeholder, while the code is created manually and an additional model is used for code visualization (e.g. to support code reviews). Further, different models might be used on different levels of abstraction (e.g. as specified in the MDA standard [154]). Thus, MDE can manifest in a multitude of the most diverse combinations of languages, tools, and automations.

To differentiate more clearly between the concrete technical aspects of manifestations of MDE, the term **MDE setting** is introduced for this thesis as

the set of manual, semi-automated, and automated activities that are employed during development, the set of artifacts that are consumed or produced by these activities, the set of languages used to describe the artifacts, as well as the set of tools that allow editing used languages or implement automated activities.

Thus, an MDE setting can include artifacts that are expressed in different languages. Since models are often used in combination with other languages, the set of languages in an MDE setting can include all kinds of formats.

Similarly, an MDE setting can include multiple automated activities. Examples for automated activities within an MDE setting are code generation, model transformation, model synchronization, compilation and interpretation. Compilation and interpretation are the last steps to gain an executable program or execute the program. Examples for manual activities are modeling and coding, but also quality assurance activities, like code reviews. A special form of modeling and coding are manual transformations, where a major part of the created content was already part of an input artifact. Like automated transformations, also manual transformations, might be endogenous or exogenous, horizontal

3. Analytical Insights

or vertical, and can either lead to the creation of a new artifact or adaptation of an existing artifact. Tools build the framework for the execution of the automated activities (e.g. model checkers like PRISM [122] or the TGG interpreter [113]) and provide editors (e.g. Visio¹ is used for the DSM in [50]) and support for manipulation of artifacts during manual activities.

Each activity consumes, manipulates, and produces a set of artifacts, which can be specified using different languages. Different artifacts that are used as input of an activity might specify different aspects (e.g. behavioral and structural aspects) or parts (e.g. different components) of a system. Artifacts can be interrelated in different ways as summarized in [P5]. First, an artifact can explicitly reference another artifact. Second artifacts can overlap in the described content. On the one hand, this happens, when artifacts describe complementary aspects of a system. In this case an overlap reflects how the content of both artifacts needs to be assembled. For example, a UML class diagram describes what classes are available in a system. A UML sequence diagram reflects parts of this information, since modeled objects and their communication need to be conforming to classes and their associations within the class diagram. On the other hand, an overlap can be the result of a model transformation that created one artifact on the basis of the other one.

The definition of the term MDE setting introduced above in Section 3.1, allows to capture all forms of technical settings from code-centric development to DSLs to MDE. For example, an MDE setting for code-centric development with Java includes artifacts formulated in Java and artifacts formulated as Java-Bytecode. Manual activities are coding of artifacts formulated in Java [90], while the two automated activities are compilation (with Java artifacts as input and Java-Bytecode artifacts as output) and interpretation (with Java-Bytecode artifacts (consisting of the translated program and libraries from the Java API) as input). Tools in such an MDE setting can be Eclipse as integrated development environment (IDE) with an editor and compiler for Java, as well as a Java Virtual Machine (e.g. the Mac OS Runtime for Java) for the desired platform [128].

Another simple example is the MDE setting for the DSM for inventory tracking systems presented in [50]. Here the used artifacts are formulated in the presented DSM and interpreted afterwards. Thus there is the manual activity to model the inventory tracking systems and the automated activity to interpret the model. The tool Visio is used as editor, while the interpreter is implemented bases on Visual Basic.

3.2. Analytical Insights: Changeability

Changeability is a software quality attribute that describes the amount of effort necessary to change a system [1]. There are two reasons why this internal quality attribute is especially important in software development. First, as stated by Lehman et al. [124], most successful software systems are subject to permanent change. For example, systems need to be adapted to new business needs. Second, most software development processes rely on incremental or agile development, where the software system is changed and revised several times during development.

In this section, it is discussed how changeability can be influenced by MDE (i.e. what changeability concerns are affected by MDE settings). Subsequently, a hypothesis on the relevance of these changeability concerns in practice is formulated. To better discuss the effects of changeability the term changeability concern is introduced here to refer to a specific dimension of changeability. MDE settings or other techniques can affect changeability concerns to different extents.

3.2.1. Basic Influences on Changeability

As in Fricke et al. [81] the system's architecture is often in focus when changeability is discussed. Also most work on changeability analysis in literature focuses on design patterns. For example, Garzas et al. measure changeability in terms of the number of indirections introduced through design patterns into code [88]. Other approaches focus on a direct analysis of the source code changes (e.g. [85]).

A mechanism that is used to improve changeability on level of system's architecture is *separation of concerns* [194]. The main *changeability concern* is the *number of system parts that have to be touched to apply a change*. This includes the fraction of the system the needs to be understood, as well as the fraction of the system that needs to be modified. The goal is to reduce this fraction.

¹<http://msdn.microsoft.com/de-de/office/aa905478> (last access at November 9th, 2013)

To reach this goal several rules and techniques are applied in system's architecture. For example in [191] the concepts *cohesion* (i.e. the degree to which elements of a module belong together [218]) and *coupling* (i.e. the degree to which modules rely on each other) are introduced. Until today, it is a widely used rule of thumb to reach a loose coupling and a strong cohesion, when splitting up software into modules or components. The concept of polymorphism that is used in object oriented programming allows that objects of different subtypes can be accessed via the interfaces of a common super type. This allows exchanging these objects. The use of well defined and standardized interfaces improves changeability, too. This concept is for example extensively used in *service oriented architectures*, where systems are built as composition of services that interact via specified interfaces. Finally, there is much work done in supporting identification and removing of code clones, such that changes do not need to be applied to all clones individually. Like these examples, all techniques that are applied to improve changeability at level of the system's architecture can be seen as ways to separate concerns.

Besides architectural aspects also the choice of the language has influence on changeability. The main mechanism that works here is *abstraction*. This abstraction influences intuitiveness and effort to describe certain software parts. The influence of the language is often unattended in literature on changeability, which might be a result of three reasons. First, the abstraction is not only relevant for changeability, but also for the performance of the initially implementing a version of the system. Second, the *changeability concern “intuitiveness and description effort”* it is even harder to measure or rate than the *changeability concern “number of system parts to be touched”*. Third, the choice of a language is often constraint by many traditions and factors, like available skills. Thus, an explanation that the focus of changeability research is often on architecture might be that – in contrast to the language choice – the architecture provides developers with the concrete option to improve changeability within a running project. However, there is a second indirect mechanism how the language influences changeability, which shall be mentioned here. This mechanism is the influence of the *language’s structure* (i.e. modularization concepts within the language) on the options to design and manipulate the system's architecture. For example, object oriented programming and the use of polymorphism depends on the choice of an object oriented language (e.g. instead of an imperative language).

Subsuming, changeability in “classical” software development (i.e. use of a single main language and no code generation) can be influenced by the choice of the language and the design of the architecture. The changeability concerns that are affected are “*intuitiveness and description effort*” as well as the “*number of system parts to be touched*”.

3.2.2. MDE's Influence on Changeability

In the following it is analyzed how an MDE setting can influence changeability of the software that is built with that MDE setting. It will be shown that MDE settings can affect the two identified changeability concerns via additional mechanisms. More importantly, MDE settings can affect further changeability concerns. As described in Section 2.1 MDE settings can include a combination of multiple artifacts and languages as well as a combination of potentially intermixed manual and automated activities. Remind that the term MDE setting is used in this context also when no modeling language is used.

Here, the different characteristics of MDE are systematically analyzed in detail to identify how they can influence changeability. For each characteristic of MDE settings, it is discussed which already identified changeability concern can be affected. Further, it is examined whether additional changeability concerns are affected by a characteristic. As a result this section comes up with a list of changeability concerns that can be affected by an MDE setting.

Combination of Languages

First, the fact that MDE settings can include a combination of different languages amplifies the effect that the choice of a language already can imply. While a single language can improve intuitiveness and description effort for specific types of changes (e.g. changes of structural aspects), other types of changes (e.g. changes of behavioral aspects) might be less well supported. Consequently, a meaningful combination of different languages can help to improve intuitiveness and description effort for different types of changes at once. For example, the language SQL is used in combination with other programming languages. In consequence, changes in the way a database is accessed can be implemented with less effort than with the combined programming language. Another example is the use of “*Status & Action*

3. Analytical Insights

“Models” (S&AM) in combination with ABAP code in the development of Business Objects for the Feature Package 2.0 at SAP [P3]. The S&AMs are specialized for describing behavior.

Similar, to a single language also the way how languages are combined can define conditions for the design of the system’s architecture (and the resulting separation of concerns). For example, the combination of Hypertext Markup Language (HTML) and Cascading Style Sheets (CSS) allows separating the specifications of structure and layout, while the single use of HTML does not. A simple manipulation of the layout of a website that is specified by a combination of HTML and CSS might be applied by changing a single part of a single CSS file. In contrast, the implementation of the same change on a website that is specified by HTML, only, might require the manipulation of multiple elements throughout multiple HTML files.

Besides these additional influences to the already identified changeability concerns, the combination of different languages can also add to the complexity for the developer. Similar to the question “how many system parts have to be touched to apply a change?” that can be asked for the changeability concern “*number of system parts to be touched*”, the use of a combination of languages instead of a single language lead to the question “how many languages have to be understood to apply a change?”. Thus, the combination of different languages in MDE affects the additional changeability concern “*number of languages to be understood*”. For example, for the implementation of a change in the *EMF example* (Section 2.6) artifacts up to three languages need to be understood: Ecore, EMF Generator Model and Java.

Combination of Artifacts

MDE settings are characterized by a combination of different artifacts. On the one hand, the combination of different artifacts during development can be system specific, as in “classical” development (e.g. the number of classes in a Java program leads to the number of Java code files). Further, the combination of different languages implies the use of different artifacts. However, MDE settings might also define further expectations on the combination of artifacts. This is due to the different artifacts that are required as preconditions and postconditions of (automated) activities.

The combination of artifacts depends for example on the question whether automated activities do allow that multiple artifacts of one role and type are consumed (e.g. a Java interpreter usually can consume an arbitrary number of Java-Bytecode files). Further, several input artifacts of same type might have different predefined roles (e.g. artifacts that are reused from previous projects, artifacts that are provided by the customer, and artifacts that are created during the project might be formulated in the same language).

Thus, what artifacts are necessary to implement a concrete change depends also on the automated activities of the MDE setting. The other way around, the languages that are necessary for a concrete change, depends on the set of artifacts that are necessary for this concrete change. This combination of artifacts lifts the level of the changeability concern “*number of system parts to be touched*” to the changeability concern “*number of system parts or artifacts to be touched*”. For example, in the *EMF example* (Section 2.6) the generation of Java code from an “*ecore model*” and an “*emf generator model*” results in different artifacts such as “*interfaces*” and “*interface implementations*”. Thus, the automated activity “*generate java code*” predefines the separation of Java artifacts.

Combination of Automated and Manual Activities

Finally, an MDE setting defines a combination of automated and manual activities, which might form activity chains. Automated activities define what artifacts are consumed together to produce other artifacts, which might be consumed by manual activities. Via this propagation of changes to other artifacts, which might need to be adapted further, the concrete combination of manual and automated activities influences the changeability concern “*number of system parts or artifacts to be touched*”. For example, for the MDE setting of the *EMF example* (Section 2.6) different changes are possible. Changes that affect the “*metamodel*” need to be applied on the “*ecore model*” and lead to subsequent adaptation of the “*emf generator model*” and of one or more “*interface implementations*”. In contrast, changes that affect aspects of the system that are not specified in the “*ecore model*” or the “*emf generator model*” can be directly applied to the “*interface implementations*”.

Besides the additional influence on already identified changeability concerns the combination of manual and automated activities lead to additional changeability concerns. First, changes that are applied to an artifact might need to be propagated to other artifacts, through a chain of manual and automated transformations. Comparable to the question “how many system parts have to be touched to apply a change?” the additional question “how many activities have to be applied?” arises. Thus, the combination of manual and automated activities leads to the changeability concern “*number of activities that have to be applied*”. For example, changes that are applied to the Ecore model within the *EMF example* (Section 2.6) require the subsequent new application of the automated activities “*create EMF generator model*” and “*generate java code*” to propagate the changes into the system. Further, additional manual activities might be necessary to adapt the “*emf generator model*” and the “*interface implementations*” accordingly.

Second, it is well known that automated activities might lead to problems when they are used to implement changes. For example, the repeated application of a code generator to generate source code from a changed model can overwrite parts of the source code. This leads to loss of information when these code parts were added manually after the previous generation of the code file. Similarly also other automated activities might cause problems, when they are applied to implement a change. This leads to two additional change concerns that are affected by MDE settings. First, the application of automated activities might lead to “*unexpected loss or preservation of content*” within artifacts. Some generators are able to deal with protected regions. Content within such protected regions is preserved during new application of the generation. Second, not only the content of single artifacts might be affected, but also references between artifacts. Thus, there is the additional changeability concern “*unexpected loss or preservation of references*” between artifacts.

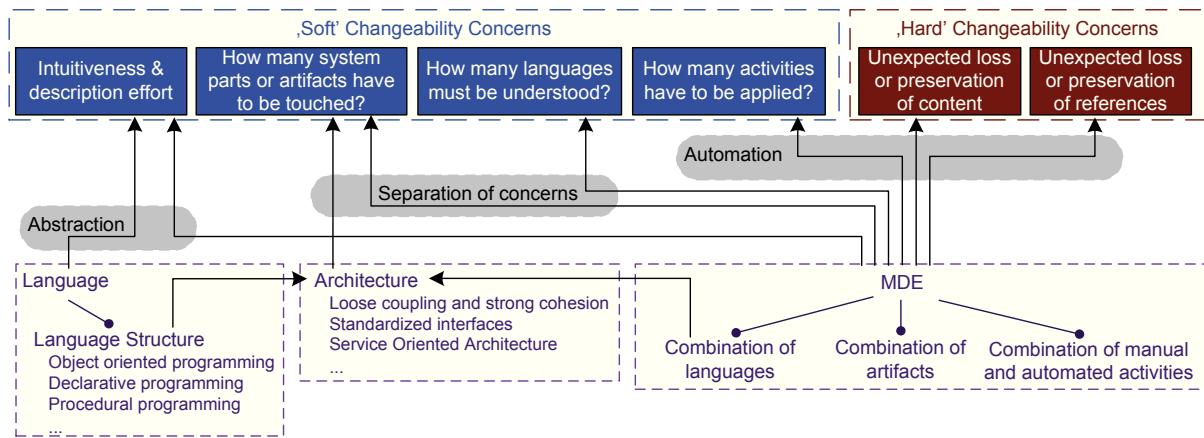


Figure 3.2.: Changeability concerns are affected by single languages, the architecture of a system, as well as the MDE setting via abstraction, separation of concerns, and automation.

Classification of Changeability Concerns

In Figure 3.2 the changeability concerns are summarized. The changeability concerns can be split into two groups: soft changeability concerns and hard changeability concerns.

Hard changeability concerns (i.e. “*unexpected loss or preservation of content*” of artifacts and “*unexpected loss or preservation of references*” between artifacts) can cause that the actual system implementation deviates from the developer’s expectation, when a change is implemented. Parts of the system implementation can get lost or remain in the system without being expected to. It is possible that an MDE setting does not affect a hard changeability concern at all. Apart from that, hard changeability concerns can be affected to different extents, e.g. depending on the number of artifacts of which content can get lost or is preserved unexpectedly, depending on the amount of content that can get lost or is preserved unexpectedly, or depending on the number of references in an MDE setting that can get lost or is preserved unexpectedly. If there is only one artifact of which content can get lost and the amount of content is small (e.g. only a configuration is affected), the hard changeability concerns “*unexpected*

3. Analytical Insights

loss or preservation of content" is affected weakly. In contrast, the same hard changeability concern is affected strongly, when the amount of content is not small or when multiple artifacts in an MDE setting can be subject of the loss of content.

In contrast, soft changeability concerns, describe the effort that is necessary to consistently include a change into a system. Thus, soft changeability concerns are the "*number of system parts or artifacts to be touched*", the "*number of activities that have to be applied*", the "*number of languages to be understood*", and the "*intuitiveness and description effort*" to describe a change within an artifact. In contrast to hard changeability concerns, soft changeability concerns are always affected to a certain extent. For example, if only a single artifact needs to be touched to implement a change the soft changeability concern "*number of system parts or artifacts to be touched*" is affected weakly. The same soft changeability concern is affected strongly if the number of artifacts that need to be touched exceeds 10.

Subsuming, both, soft changeability concerns and hard changeability concerns, can be affected strongly or weakly by an MDE setting.

3.2.3. Interrelation of Influences on Changeability

Looking upon the identified changeability concerns, it becomes clear that much of the mechanisms driving changeability on architectural level between different system parts, work also on MDE level between different artifacts.

For example, separation of concerns and the use of indirections enable on architectural level that certain parts of a system can be exchanged easily by applying only local changes. This separation is propagated to the artifacts and affects automated and manual activities. If parts of an artifact cannot be separated clearly, it will be difficult to apply techniques that aim at dealing with changeability concerns like "*unexpected loss of content*", such as protected regions. Also a missing clear separation between the artifacts influences changeability, as multiple artifacts might have to be touched for one change.

It is important to notice that decision how concerns are separated can be moved between architectural level and the design of an MDE setting. Automated activities might be designed to weave artifacts that specify different aspects of the system. Alternatively, reused parts that are separated on architectural level might become part of a transformation. For example, in OMG's MDA [154] parts of the implementation that are specific for the platform might be part of the transformation between platform independent model and platform specific model. This way, the platform specific aspects can be exchanged by substituting the transformation that is used.

Whether such a decision is better located at architectural level or at the level of the MDE setting is a trade-off. Separation of concerns that is created on architectural level can benefit from product specific characteristics. However, the decisions need to be done and justified for each single product. In contrast, separation of concerns that is created by the MDE setting can be reused for several projects. Thus, separation of concerns is a built-in part of the MDE setting and does not need to be discussed for every new project. However, on level of the MDE setting the decisions cannot benefit from product specific characteristics. The decisions have to fit the whole domain of products that is built with the setting. Naturally, separation of concerns that is created on both levels can be combined to reach an optimal effect for changeability.

Furthermore, MDE has influences on changeability that go beyond the separation of concerns. The design of automated activities and their combination with manual activities in MDE settings matter. Consequently, to understand how MDE settings in practice affect changeability, it is important to understand how MDE settings look like in practice.

3.2.4. Obstacles for Creating a Global Measure for Changeability

Developing a global metric for changeability that takes all identified changeability concerns into account is difficult for several reasons. In the following, these obstacles are discussed.

Measuring single changeability concerns: First, there are currently single changeability concerns for which it is hardly possible to develop a metric. For example, intuitiveness and description effort to specify a change in different languages cannot be measured today. The high number of poorly understood influences, such as culture and education, makes the development of metrics difficult.

Weighting changeability concerns: Second, it is difficult to decide how different changeability concerns have to be weighted. For a global metric of changeability it has to be decided, e.g. whether

hard changeability concerns are more important than soft changeability concerns.

Scope: Third, architecture and MDE setting have different scopes. An MDE setting might be reused by several projects for different products. Thus, a metric for changeability that measures effects of the MDE setting or the used languages has the potential to be independent of the concrete product. In contrast, the architecture is specific for a product (or a product line). Thus a metric for changeability that measures effects of the architecture holds for the specific product, only.

Need for change: Finally, the actual changeability depends on the need for change. All techniques that aim at improving changeability improve changeability for specific parts of the system or artifacts in the MDE setting. When there is a bigger need to change other parts of the system, there is not much benefit. Thus, the appropriateness of an MDE setting depends strongly on the question which artifacts will probably change. This need for changes can be domain specific, but might also change over time.

As a consequence a global metric for changeability is difficult to achieve. Nonetheless, focusing on single changeability concerns can support improvement of the overall changeability. This thesis focuses on the hard changeability concerns, as well as on the number of activities that have to be performed.

3.3. Analytical Insights: Process Interrelation

In literature, the combination of MDE approaches and software development processes is often realized by adapting one of them to the other (e.g. [120, 130, 221]). Consequently, the question arises why MDE settings can imply constraints on software development processes.

This section aims at identifying traits of MDE settings that can lead to constraints on software development processes. Therefore, first a reference model for the combination of MDE settings and software development processes is introduced. Afterwards, the need for this investigation of the process interrelation is motivated by a summary of assumptions on MDE settings and adaptations of processes within existing proposals for the combination of MDE and processes.

Subsequently, candidates for relevant traits of MDE settings are identified. It is discussed how these candidate traits can – depending on their manifestation – constrain software development processes.

3.3.1. Reference Model for Combinations of MDE Settings and Software Development Processes

Since the combination of MDE settings and software development processes happens quite differently in literature, a generic reference model of the interface between both is introduced in this section. The discussions on interrelation of MDE and software development processes in the remainder of this thesis, refers to this generic reference model.

Combinations in Literature

In literature several approaches for concrete combinations exist. In Table 3.1 some examples are summarized. For example, Pietrek et al. present ideas how RUP and the V-Modell XT can be combined with MDE, in order to improve automation [197]. The mapping to RUP is defined, such that different automated activities are spread over the process phases. Similarly, the mapping to an adapted version of the V-Modell XT (which is an extension of an approach by Fieber and Petrasch [70]) is specified by assigning different MDE activities to the phases of the process. Another approach to combine the V-Modell XT with MDE is proposed by Rausch et al. [166]. This short proposal focuses on the mapping of MDE artifacts to process documents, especially the documentation. Documentation should be automatically generated as a by-product of system modeling.

Sindico et al. present an industrial system engineering process (ELT from Elettronica SpA) that integrates Simulink, SysML, and EMF models into a process [184]. The MDE activities are spread over the different phases of the process. Also MDE artifacts are explicitly mapped to documents of the process. For example, a “*System Subsystem Design Description*” is semi-automatically created on the basis of the “*Solution Definition*”.

Approaches that aim at combining agile processes with MDE techniques most often assume that the MDE techniques are applied as substitution of the pure manual implementation activity within the iterations. An example for such a combination is Ambler’s Agile Model-Driven Development (AMDD) [6], which is also discussed by Pietrek et al. in [197]. AMDD is not a standard process, but specifically

3. Analytical Insights

created for MDE. Here iterations are implemented with a modeling activity and a subsequent implementation activity, each. In addition, an initial modeling cycle is implemented with two more extensive modeling activities. Mahé et al. [130] discuss the challenges for a combination of agile techniques and MDE more generally. Oriented on AMDD, the mapping idea from Mahé et al. focuses on filling implementation cycles of the desired agile process with modeling activities. Also Zhang et al. [221, 222] combine different agile processes with MDE techniques. Here the MDD iterations and activities are mapped to process sprints. Finally, Fernandez et al. present an MDD-based web development process in [68] that includes a distinct modeling as well as a coding phase.

Experience on applying agile techniques to a given MDE approach is reported by Kulkarni et al. [120]. Here the MDE activities are spread over two types of process sprints. In addition, some MDE artifacts are mapped to documents of the process. For example, meta-sprint deliverables can be design documents. Final an approach that should be mentioned here is presented by Fondement et al. [74]. In this approach no existing software development process is combined with MDE, but a framework for the creation of a process for a given MDE approach is presented. A concrete case study of an MDE approach and the correspondingly used process is documented by Mellegård et al. [135]. Here a mapping between MDE artifacts and process documents, such as “*function description*”, is described.

Table 3.1.: Summary of the proposed combinations of software development processes with MDE (*min.* = minimum mapping: the only MDE artifacts that are mapped to “documents” of the process are models and code files that are mapped to the software product; **MDE Process** = specific process for MDE approaches)

Approach	Process	Mapping to process documents	Mapping to process phases (single phase, iteration, multiple phases)
Pietrek et al. 2007 [197]	RUP	min.	spread over phases
Pietrek et al. 2007 [197] (based on Fieber and Petrasch [70])	V-Modell XT	min.	spread over phases
Rausch et al. 2005 [166]	V-Modell XT	✓	(implicitly) spread over phases
Sindico et al. 2012 [184]	ELT	✓	spread over phases
Ambler 2004 [6]	AMDD: agile MDE process	min.	mapped to (2 types of) cycles
Pietrek et al. 2007 [197]	AMDD agile MDE process	min.	mapped to (2 types of) cycles
Mahé et al. 2010 [130]	agile MDE process	min.	mapped to cycle
Zhang et al. 2004/2011 [221] and [222]	agile processes	min.	mapped to sprint
Kulkarni et al. 2011 [120]	SCRUM (agile)	✓	spread over (2) types of sprints
Fernandez et al. [68]	MDE process	min.	spread over phases

Reference Model

First of all it can be seen that a described MDE setting might not support the whole software development process. Parts of the process (e.g. concerning requirements engineering or testing) might not be considered in the MDE setting (as also experienced by Kuhrmann et al. [4]).

While most of the concrete approaches focus on the question how different phases, cycles, or sprints of a process are implemented with or supported by MDE activities, some of the approaches include or even base on a description of how MDE artifacts are mapped to documents of the process. Consequently, the reference model for the interface between a software development process and an MDE setting consists of two mappings as illustrated in Figure 3.3. On the one hand, artifacts (or groups of artifacts) in the MDE setting might be mapped to documents that are specified by the software development process. Not all documents from the software development process need to be mapped to artifacts in the MDE

setting (e.g. the creation of artifacts for documentation reasons might or might not be supported by the MDE setting). Similarly, not all artifacts from the MDE setting need to be mapped to documents in the software development process (e.g. many artifacts in the MDE setting are intermediate products that depend on the used MDE techniques, languages, and tools). The set of artifacts in the MDE setting that are actually mapped to documents in the software development process might even be limited to the resulting software, only.

More important is the second mapping, which is a mapping of activities in the MDE setting to activities, phases, or iterations in the software development process. An activity, phase, or iteration of a software development process can be implemented via one or more activities from the MDE setting (for example, multiple modeling, transformation, and coding activities might be part of a scrum sprint). Further, different activities in the MDE setting might be reused in multiple phases of the software development process or product life cycle (e.g. during implementation and maintenance). Note that activities in the MDE setting might remain unused and therefore are not mapped. Similarly, the MDE setting does not necessarily cover the whole process.

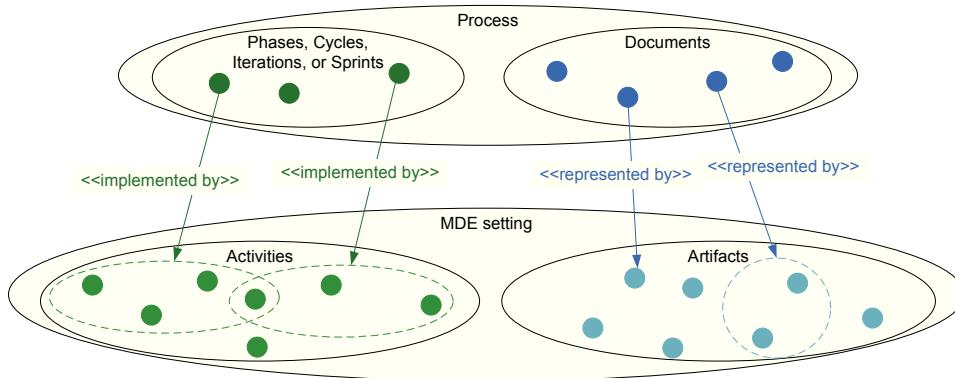


Figure 3.3.: Illustration of the general mapping idea of the reference model.

For example, in Figure 3.4 it is illustrated how an excerpt of activities from the MDE setting BO (that will be introduced in Chapter 4) might be embedded in an agile process, inspired by Ambler's AMDD [6]. An initial modeling activity is mapped to an initial modeling phase. The following cycles in the agile process are implemented by the remaining manual and automated MDE activities. Thus, in each iteration of the cycle these activities are performed.

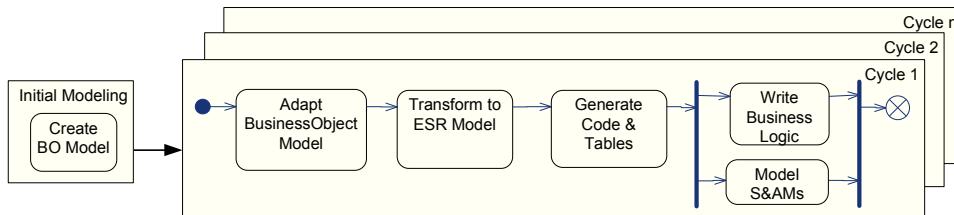


Figure 3.4.: Example how activities from the MDE setting in case study BO (see Chapter 4) might be used to implement iterations within a simple agile process (the used notation based on activity diagrams: the rectangles illustrate phases and repeating iteration cycles of a process).

These mappings of activities and phases is the interrelation, via which properties of an MDE setting might constraint the software development process, and the other way around. This, concerns on the one hand the order of activities. The order of activities or phases in the software development process must be compatible to the *possible orders of activities* in an MDE setting that is used to implement the process. On the other hand, non-functional properties like effort, risk, and complexity of activities in an MDE setting need to be compatible with the software development process that is implemented with these MDE activities.

3.3.2. Assumptions and Adaptation within Literature Combinations

Above several concrete proposals for the combination of MDE settings and software development proposals were mentioned. Before possible constraints and effects that arise from the interrelation of both technologies are studied in the next sections, this section motivates the need for this investigation. Therefore it is summarized whether the combination proposals rely on assumptions on the MDE settings or adaptations of the used processes. Table 3.2 summarizes information whether and how the processes are adapted in literature combinations. In addition, it is summarized whether and what assumptions are made on the MDE approach.

For the combination of RUP with MDE presented in [197], Pietrek et al. assume that the underlying MDE setting is an MDA approach. Some of the manual tasks in RUP should be substituted by automated activities. For the process, Pietrek et al. expect a shift of effort from the implementation phase to analysis and design phases. In addition only small changes on the process are proposed. For example, roles should be refined.

For the combination of the V-Modell XT with MDE, Pietrek et al. assume an underlying MDA approach, which comes along with a fixed set of required MDE artifacts and activities. The process is adapted, such that process phases are exchanged, additionally introduced, and partly merged. This is done in way that the different MDA abstraction levels (PIM and PSM) are reflected by the different phases. In contrast, the combination of the V-Modell XT with MDE that is proposed by Rausch et al. requires no adaption of the process. However, for the MDE setting this combination proposal assumes that code generation is part of the MDE setting and that an additional automated generation activity for the documentation is available [166].

Sindico's combination of the ELT process with MDE bases on concrete assumptions on the technologies, activities, and tools that are used for the MDE setting. In addition, they report on positive effects from automation: due to removed manual information propagation, expensive manual reviews could be removed from the process, too [184].

Ambler's Agile Model-Driven Development (AMDD) process is not introduced as a direct adaption of a standard process, but specifies an own MDE specific process. Nonetheless, constraints on the used MDE approach are formulated. First, AMDD bases on a fixed set of MDE activities (modeling and coding) and artifact roles (models and code files). This way more complex MDE settings, but also simpler MDE settings (e.g. with model interpreters) are excluded for AMDD. In addition, strict constraints on the time that can be used for these activities (e.g. minutes for the modeling activity and a few hours for the coding activity) are defined in [6]. In contrast Pietrek et al. discuss that OMG's MDA is "too heavy" for a combination with AMDD, only [197].

The proposal of Mahé et al. includes a process that is specifically created for MDE, too. Due to the similarity to AMDD, also this proposal makes concrete assumptions on the used MDE artifacts and activities. In addition, this proposal includes the option to use executable models [130]. As in [6], assumptions on the time required to run through a cycle of the process are made in [130]. Zhang et al. make the assumption that an MDE approach that is used within an agile process fulfills certain ideals like a high degree of automation and code generation [222]. The MDE specific process proposed by Fernandez et al. is specific to MDA [68]. Finally, Kulkarni et al. [120] start with a given MDE approach. The SCRUM process is adapted, such that additional meta sprints are introduced, in order to handle more time consuming tasks. In addition, the team structure is changed, such that module ownerships are substituted by feature ownerships.

The examination of the combination proposals supports the assumption that MDE and software development processes influence each other: in all approaches and reports either the software development process is adapted or there are assumptions on the design of the MDE setting (or both).

It is noticeable that assumptions on design of the employed MDE setting are actually defined in all proposals. This includes concrete assumptions on the set of artifacts, activities, and tools in the MDE setting, but also assumptions on the time required for activities (including effort that is put into modeling or coding) and the degree of automation. On the one hand, only one of the discussions (Pietrek et al. on AMDD) does without assumptions of the concrete structure of the MDE artifacts and activities. On the other hand, some assumptions are defined on the time required for different MDE activities.

In addition, the most approaches come along with changes to the process (or even define specific MDE processes). Proposed changes to processes affect the phases and sprints of the process, the number of manual quality assurance activities, the roles, and team structure (e.g. in Sindico's approach [184]).

Table 3.2.: Summary of the proposed changes and assumptions on software development processes and MDE (MDE Process = specific process for MDE approaches)

Approach	Process	Assumptions on the MDE Setting	Changes of the process
Pietrek et al. 2007 [197]	RUP	✓ (only limited structures of MDE setting)	(✓) (small changes, e.g. refined roles)
Pietrek et al. 2007 [197] (based on Fieber and Petrasch [70])	V-Modell XT	✓ (specified MDE artifacts and activities)	✓ (changed process phases)
Rausch et al. 2005 [166]	V-Modell XT	✓ (code generation; documentation generation)	-
Sindico et al. 2012 [184]	ELT	✓ (concrete technologies and tools)	✓ (removed manual QA activities)
Ambler 2004 [6]	AMDD: agile MDE process	✓ (specified MDE artifacts and activities; time constraints)	<i>no standard process</i>
Pietrek et al. 2007 [197]	AMDD agile MDE process	✓ (MDA is “too heavy”)	<i>no standard process</i>
Mahé et al. 2010 [130]	agile MDE process	✓ (specified MDE artifacts and activities; executable models; short cycles)	<i>no standard process</i>
Zhang et al. 2004/2011 [221] and [222]	agile processes	✓ (high degree of automation; code generation)	-
Kulkarni et al. 2011 [120]	SCRUM (agile)	✓ (given MDE approach)	✓ (changed team structure; added meta sprints)
Fernandez et al. [68]	MDE process	✓ (specified MDE artifacts and activities - oriented on MDA)	<i>no standard process</i>

3.3.3. MDE Traits

The discussion above in Section 3.3.2 supports the assumption that there seems to be an interrelation that can affect diverse aspects of software development processes. However, while these examples illustrate possible effects of the interrelation of MDE settings and software development processes, only some of them contain vague explanations for the interrelation (e.g. that an MDE setting should allow short cycles). General predictions what characteristics or traits of an MDE setting lead to the effects is rare. Similarly, there is only rare literature that explicitly investigates the interrelation between MDE settings and software development processes [8, 96, 106, 190].

Therefore, this section aims at identifying traits of MDE settings as candidates to explain constraints on software development processes. The examples examined above are not used as a starting point for the investigation for three reasons. First, this is the fact that the number of combination proposals that actually work with (adapted) standard processes is quite small (only 6 of the 10 proposals listed in Table 3.2). Second, these combination proposals mainly stem from research. Thus, they were not necessarily applied in practice (which holds for at least three of the 6 combinations with standard processes) and do not represent practitioners’ solutions. Third, while these proposals reveal how broad the spectrum of possible effects on processes can be, it remains unclear whether the chosen adaptations were necessary and good solutions.

Thus, instead of trying to extract and generalize cause-effect relationships from the few eligible combination proposals, the nature of the process interrelation will be approached from another direction here. It is theoretically examined, why and how MDE settings can lead to constraints on software development processes. So far it is clear, that some artifacts in MDE settings can fulfill the roles of artifacts or documents that are specified in software development processes. More important, activities from software development processes are implemented using a set of MDE activities. In the following, this later connection of MDE settings and software development processes is used as basis to identify relevant traits of MDE settings. Such *MDE traits* are structural characteristics of an MDE setting and can have diverse manifestations. This thesis focuses on *MDE traits*, where some of the theoret-

3. Analytical Insights

ically possible manifestations of this trait can imply constraints on software development processes. Such a manifestation is referred to as *process-relevant manifestation* of an MDE trait. Correspondingly, a manifestation that does not imply constraints on software development processes is referred to as *process-neutral manifestation* of an MDE trait.

To search for promising candidates for relevant MDE traits, the role of activities of within MDE settings is systematically examined. It is considered whether extreme manifestations of these roles might have impact on software development processes. When examining activities in an MDE setting, two aspects can be taken into account. First, MDE traits can concern the order of activities. Here, an extreme situation is an activity for which a relative order to each other activity is predefined by the MDE setting. Thus, no other activity might be performed in parallel to this activity, with the consequence that the development is split into phases. Such *phases* are the first trait that will be discussed further. Second, MDE traits can concern effort, risk, and complexity associated with activities. Considering single activities, manual activities are in most cases associated with more effort and risk than automated activities (assuming that these automated activities are well tested and mature). Further, from all types of manual activities, one is associated with high effort and risk to lead to inconsistencies between artifacts, on the one hand and simultaneously has a very low overall benefit concerning the content of the system specification, on the other hand. This is the trait *manual information propagation*, which is chosen to be discussed further. Further an MDE trait can concern a characteristic on the complexity of a set of activities. The focus here is on a typical characteristic of MDE settings. These are the chains of activities that need to be performed to propagate a newly introduced change to the artifacts that represent the system under construction (i.e. artifacts that are compiled or interpreted). Such chains will be discussed as the trait complexity of activity chains.

In the following, the three identified candidates for MDE traits are investigated more in detail.

Phases

The first MDE trait results from the occurrence of activities for which a relative order to each other activity is defined (i.e. an activity that is not executed in parallel to any other activity). Such activities are called synchronization point. A global synchronization point is a synchronization point, where all other activities in the MDE setting either succeed or precede, but are not executable in parallel to this activity. Note that the considered order of activities excludes cycles for error correction or the application of changes. For example, in case of an error correction a code generation is followed by the test and error detection, which might then be followed by the correction of the change in the model. However, for the order of activities, the code generation is not considered as an activity that precedes the adaptation of the model.

Depending on their location synchronization points can split the other activities in the MDE setting into *phases*. These *phases* are sets of activities (or extensive manual activities) that partly can be executed in parallel or without a fixed order.

Phases and synchronization points affect how developers interact. First, developers who provide different consumed artifacts of a synchronization point have to wait for each other until the synchronization point can be triggered and artifacts that are created by the synchronization point are available to continue with the work in the next phase. As a side effect, the synchronization can affect the apportionment of work between developers, since they will try to decrease waiting times. This can decrease the freedom given by process, e.g. a development team in Scrum [176] should be free of external constraints on the way of interaction. Second, the resulting set of *phases* can affect the whole work. Developers working in a team on the same release are enforced to work in the same phase (e.g. before or after a specific synchronization point). As a consequence, phases can drive the tailoring of the team structure and the software development process for implementing new releases. Subsuming, an MDE setting that includes multiple phases that result from synchronization points can enforce developers to synchronize their work and wait for each other.

Manual Information Propagation

In MDE approaches artifacts on different levels of abstraction are used to describe a system. Content is moved from more abstract to less abstract artifacts and enriched further. Further, artifacts on the same level of abstraction might share common content, which has to be held consistent. Such move-

ment of content (called transformation in the following) is not always automated. However, manual transformations are an additional source for errors.

The MDE trait manual information propagation leads to additional potential for human errors and thus the need for additional quality assurance. This additional quality assurance is theoretically necessary for each pair of artifacts between which content was propagated manually. Thus, an MDE setting defines constraints on necessary quality assurance activities. Since affected artifacts are in most cases used as input for further activities and it is beneficial to prevent that working effort is invested on the basis of faulty documents, there is the need to apply these quality assurance activities already within the implementation phase. Consequently, the manual information propagation can enforce a mixture of quality assurance activities and development activities. For example, Sindico et al. observe the effect that the addition of automation to decrease manual transformations in Sindico's approach ([184]) leads to saving of manual quality activities. Thus, a software development process needs to allow for time and human resources to perform such additional quality assurance activities (e.g. reviews).

Complexity of Activity Chains

Finally, the MDE setting determines how many activities have to be applied successively to define and propagate a change into the artifacts that are compiled or interpreted. These activity chains are of different complexity. This complexity consists of two parts: the effort and time that is required for the different activities and the number of context switches that a developer has to deal with.

While the required effort and time are difficult to measure and depend strongly on the actual system under construction, the number of context switches can be assessed with a measure that actually reflects a soft changeability concern, too: the number of activities that have to be applied.

Whether, a long activity chain means a long time until the activities are performed depends also on the activities themselves. For example, a short activity chain with complex manual tasks can lead to a long duration for the implementation of a change. However, activities in an MDE setting differ in the consumed and produced artifacts and with it in the languages that need to be understood. Also the use of different tools and automation technologies maps to the activities in the MDE setting. Consequently, the length of an activity chain can be used to describe the number of context switches that a developer has to deal with. Thus, the length of activity chains is used within this thesis as the measure for the complexity of activity chains.

An aspect that is affected by the complexity of an activity chain is an MDE setting's applicability to agile processes. For example, when ten activities need to be performed for the application of a specific change, the application of an agile process, which requires short iterations, most probably decreases in its effectiveness. Subsuming, a software development process defines assumptions on the frequency of iterations and indirectly also on the possible complexity of work within these iterations. If the complexity of activity chains does not fit to these assumptions the applicability of the process as defined is restricted.

3.4. Analytical Insights: Structural Evolution

Above it was discussed how an MDE setting can affect changeability and what traits in MDE settings can have an impact on the software development processes that are combined with that MDE setting. It is possible that MDE settings have negative or disadvantageous impacts.

Consequently, it is a question where process-relevant manifestations of MDE traits stem from.

An MDE setting might be designed from scratch by experts and introduced in one step into a company. In this case, the existence of negative impacts on changeability or process interrelation seems to be rather unlikely and easy to correct.

However, there is an alternative scenario. MDE settings might also result from changes of former MDE settings. Thus, MDE settings might be the result of an evolution process. This leads to the question whether MDE settings can in practice evolve in a way that negative impacts of changeability or process conformance can occur. To approach this question, possible types of changes of MDE settings will be identified in the following. Then it is systematically analyzed how each of these change types can affect the characteristics of an MDE setting.

3. Analytical Insights

3.4.1. Types of Changes in Evolution of MDE Settings

What types of changes are possible for MDE settings can relatively straight forward be derived from the definition of an MDE setting. The considered change types are summarized in Table 3.3.

Implementations of automated activities, the used (modeling) languages, and supporting tools are the technical assets of the MDE setting. All three can be exchanged without affecting the order of activities or the number of elements in an MDE setting. Therefore, these changes are called non-structural changes here. Consequently exchange or evolution of an automated activity (e.g. any model operation or code generation) is referred to as change type $C1$ in the following. Similarly exchange or evolution of a used language is referred to as change type $C2$ and exchange or evolution of a used tool is referred to as change type $C3$. Further, changes can affect the number of elements in an MDE setting. Possible changes concern the number of artifacts (referred to as $C4$), the number of languages (referred to as $C5$), the number of manual activities (referred to as $C6$), the number of automated activities (referred to as $C8$), as well as the number of tools (referred to as $C7$).

Each change in the number of automated ($C8$) or manual activities ($C6$) leads to a change in the order of activities. The relative positioning of automated activities within (and behind) the manual activities is called *order of manual and automated activities* here. Only some changes in the number of manual or automated activities also change this order of manual and automated activities (e.g. an automated activity might be introduced between two manual activities). Therefore, this special case is referred to as change type $C9$ in the following. Changes that affect the order of activities or the number of elements in an MDE setting ($C4-C9$) are called structural changes here.

For a clear terminology it is distinguished between the terms *change*, *evolution step* and MDE evolution. A *change* is a local modification of an MDE setting. The term evolution step is used to refer to the combination of all changes leading from one MDE setting that was in practical use to a next one. Finally, the term *MDE evolution* (also simply referred to as *evolution* within this thesis) describes how MDE settings evolve over time due to evolution steps. An evolution step is a structural evolution step, if the set of changes contains at least one structural change.

Table 3.3.: Summary of change types

Number	Change type
Non-structural changes	
$C1$	exchange or evolution of an automated activity
$C2$	exchange or evolution of a language
$C3$	exchange or evolution of a tool
Structural changes	
$C4$	change of the number of artifacts
$C5$	change of the number of languages
$C6$	change of the number of manual activities
$C7$	change of the number of tools
$C8$	change of the number of automated activities
$C9$	change of the order of manual and automated activities

3.4.2. Possible Impact of Evolution on Changeability

In the following, it is discussed how different types of evolution can change how an MDE setting affects changeability. To gain an impression how multiplex effects of evolution types are, their impact on how an MDE setting affects other productivity dimensions is discussed, too. Changing an MDE setting leads to changes for developers and a company. This can affect the *degree of automation* of development, the *complexity* of working with the MDE setting, and the *changeability and maintainability* of the software that is built with the MDE setting. Further, the effort for *consistency maintenance* and *integration* is affected when working with different tools. Finally, tools and automated activities that are used in an MDE setting need to be maintained and therefore affect the *cost of ownership* for a company. These

aspects imply potentials and risks for the productivity of developers or the overall productivity of a company, respectively. Consequently they are called *productivity dimensions* here.

In the following, it is discussed how these productivity dimensions can be impacted by different types of changes on MDE settings. Changes in an automated activity ($C1$) and changes in the number of automated activities ($C8$) can affect the *degree of automation*. Changing a used (modeling) language ($C2$) or the number of used languages ($C5$) can have benefits concerning the degree of abstraction, but yields the risk that the developers lack the know how to use that language (affecting the *complexity of the MDE setting*) [15]. Also an exchanged or evolved tool ($C3$) can affect the *complexity of work* for developers, since the intuitiveness tool handling might change. Similarly, a growing number of models ($C4$), necessary manual activities ($C6$), or tools ($C7$) increases *complexity* for developers. In addition, a change in the number of models affects the need to *maintain the consistency* of different models [99]. Further, new tools can lead to additional activities to move artifacts between them (increasing the *integration effort*). As tools and implementations of automated activities have to be maintained, changes in either number of tools $C7$ or automated activities $C8$ can affect the *cost of ownership*. Of course, exchanging a tool or automated activity ($C3$ and $C1$) can to a smaller extent also affect the *cost of ownership*.

Finally, risk results from the addition of automated or manual activities if this leads to a change of the order of manual and automated activities ($C6$, $C8$, and $C9$). This can introduce constellations, where automatically created artifacts are touched manually (which imply risks for *changeability and maintainability*). Figure 3.5 provides an overview about the affected productivity dimensions.

To sum up, the main risks and potentials for non-structural changes concern the *degree of automation* and the *complexity* of the MDE settings. In contrast, structural changes imply some important additional risk and potentials, like changes in the effort required to maintain consistency of all required artifacts or changes in the *cost of ownership*. Further, structural changes can have stronger effects on the different domains of productivity than non-structural changes. For example, increasing the number of used languages ($C5$) has a worse impact on the required know how than just applying changes to a used language ($C2$) in most cases. Finally, there is a group of structural changes ($C6$, $C8$, and $C9$) that can affect the *changeability and maintainability* of the software that is built with the MDE settings. This group of changes has the potential to introduce or eliminate risky constellations from MDE settings. For example, whether or to what extent a hard changeability concern is affected might change. Therefore, these structural changes are called substantial structural changes in the following.

Consequently, an evolution step is called substantial structural evolution step if the set of changes contains at least one substantial structural change. Similarly the evolution of an MDE setting that contains a least one (substantial) structural evolution step is called (*substantial*) *structural evolution*.

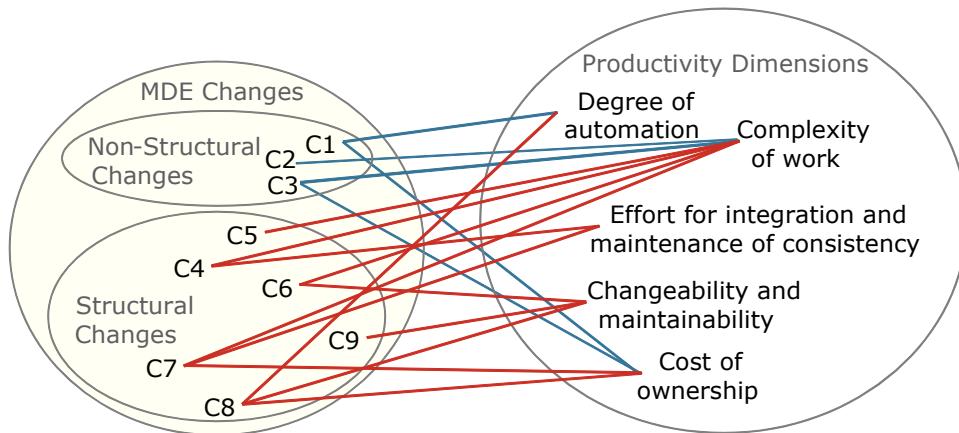


Figure 3.5.: Overview of change types and affected productivity dimensions.

3. Analytical Insights

3.4.3. Possible Impact of Evolution on the Manifestation of MDE Traits

Besides the possible effects on different productivity dimensions, the different change types can also affect the manifestations of the different MDE traits. In the following, these possible effects are discussed.

The number of synchronization points and with it the number of phases depends on the activities within an MDE setting and their order. Consequently, changes in the set of activities and changes in the order of manual and automated activities (change types *C6*, *C8*, and *C9*) can lead to new synchronization points (e.g. when the only activity that can be executed in parallel to the later synchronization point is removed) and to the removal of synchronization points (e.g. when an activity is added that can be executed parallel to a synchronization point). The manifestation of manual information propagation depends on the number of manual activities that lead to the propagation of content between artifacts. Consequently changes in the set of manual activities (change type *C6*) can affect the manifestation of manual information propagation. Similar to the number of synchronization points, the lengths of activity chains depend on the activities within an MDE setting and their order. Thus, the manifestation of the MDE trait complex activity chains can be affected by changes in the set of activities and changes in the order of manual and automated activities (change types *C6*, *C8*, and *C9*). Subsuming, the manifestations of the three MDE traits that are considered in this thesis can be affected by *substantial structural changes*.

3.5. Hypotheses

Above different theoretical characteristics of MDE setting in practice were discussed. This includes the possible impact of MDE settings on changeability (especially how hard changeability concerns are affected), the possible manifestations of the three MDE traits, and possible forms of evolution of MDE settings. However, it is a claim of this thesis that these characteristics do not only make a theoretically, but also an actual difference for MDE settings in practice. On the one hand, the actual manifestations of MDE traits and the effect on changeability can only be considered as possible reasons for the diverse degree of success of MDE in practice, if MDE settings actually differ in these characteristics. Further, in order to consider structural evolution as one reason for the characteristics on MDE settings in practice, it is necessary to show that structural evolution exists. To investigate the practical roles of changeability impact, MDE traits, and structural evolution, the following hypotheses are introduced in this section.

3.5.1. Hypotheses on Changeability

The analysis of the “problem domain” showed that MDE settings can affect various changeability concerns. However, it is still an open question whether these changeability concerns are relevant for MDE settings in practice. For example, all MDE settings in practice might only weakly or not at all affect the changeability concerns. In this case, there would be no need to develop changeability metrics or analysis methods to assess how changeability concerns are affected. Alternatively, all MDE settings that occur in practice might strongly affect changeability concerns. In this case, the questions arise whether MDE settings that do not affect changeability concerns are possible at all and whether the benefits of MDE can compensate the impact on changeability concerns. Finally, it is possible that MDE settings in practice vary strongly in the way they affect single changeability concerns. This case would motivate the introduction of changeability metrics and corresponding analysis methods. Further, only when existing MDE settings differ in their influence on changeability concerns, this influence can be a candidate for explanations for the different degree of success of MDE in practice.

To examine the diversity within MDE settings, this thesis mainly focuses on the hard changeability concern *unexpected loss of content*. The assumption is made that a phenomenon that exists in more than 5% of all MDE settings is no exception.

Since there are reports of both, successful and less successful applications of MDE in practice, the following hypothesis is formulated:

H_{changeability}: MDE settings in practice vary considerably in their influences on changeability. Thus, there are MDE settings that strongly affect the hard changeability concern unexpected loss of content (in more than 5% of all MDE settings) and also MDE settings that do not affect the hard changeability concern unexpected loss of content (in more than 5% of all MDE settings).

In case the hypothesis can be rejected, the additional influence of MDE settings on the hard changeability concern *unexpected loss of content* is most probably not the cause for the differences in the degree of success of MDE in practice. However, in case the hypothesis can be accepted, research on the analysis of the different changeability concerns is strongly motivated.

3.5.2. Hypotheses on Process Interrelation

The analysis above shows that MDE settings can theoretically include traits that can – depending on their concrete manifestation – lead to constraints on software development processes. As mentioned, the term *process-relevant manifestation* is used to refer to manifestations of an MDE trait that lead to constraints on software development processes, while the term *process-neutral manifestation* is used to refer to manifestations of an MDE trait that does not lead to constraints on software development processes.

The next question is whether process-relevant manifestations of the identified MDE traits occur in MDE settings from practice at all, and if so whether these process-relevant manifestations occur for all MDE settings from practice. In case a process-relevant manifestation of the MDE traits can be found in all MDE settings from practice in a similar way, the question arises whether certain software development processes (e.g. agile processes) are not compatible with MDE in general. However, MDE settings in practice might cover a broad spectrum concerning the manifestations of the MDE traits (i.e. MDE settings with a process-relevant manifestation of MDE traits as well as MDE settings with a process-neutral manifestation of MDE traits are not seldom). In this case, the interrelation of MDE settings and software development processes is a candidate to explain differences in the degree of success of MDE in practice. Further, a need for analysis techniques to assess the actual manifestations of the MDE traits arises. Such analysis techniques would for example support practitioners in choosing and tailoring a software development process, when an MDE setting is given.

Since this thesis aims at investigating whether the interrelation of MDE settings and software development processes can explain the varying degree of success of MDE in practice, following hypothesis is formulated:

H_{traits} : *In practice the identified MDE traits occur within a spectrum that includes process-relevant manifestations as well as process-neutral manifestations. Thus, there are MDE settings with process-relevant manifestations for some or all of the MDE traits, but there are also MDE settings with process-neutral manifestations for some or all MDE traits.*

From this abstract hypothesis a concrete hypothesis for each of the identified MDE traits can be formulated. The assumption is made that a phenomenon is not seldom if it exists in more than 5% of all MDE settings.

H_{phases} *In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait phases, in more than 5% of all MDE settings, each.*

$H_{manualInformationPropagation}$ *In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait manual information propagation, in more than 5% of all MDE settings, each.*

$H_{complexActivityChains}$ *In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait complexity of activity chains, in more than 5% of all MDE settings, each.*

In case the hypotheses can be rejected, the manifestations of MDE traits are most probably not the cause for the differences in the degree of success of MDE in practice. However, in case one or more of the hypotheses can be accepted, research on analysis techniques that allow assessing the manifestations of MDE traits is strongly motivated.

3.5.3. Hypotheses on Evolution

Above possible changes to MDE settings are categorized. The discussion of the different change types showed that some structural changes can indeed lead to a different changeability or change the manifestation of MDE traits. There is currently little knowledge whether the structural and substantial structural changes actually occur in practice.

3. Analytical Insights

However, only when structural evolution steps are common they are candidates to explain the characteristics of MDE settings in practice. It is assumed here that a phenomenon is common if it occurs in more than 25% of all cases. Therefore, the following hypotheses are formulated here:

$H_{existence}$: *Structural evolution and substantial structural evolution occur in practice.*

H_{common} : *Structural evolution and substantial structural evolution are common in practice (i.e. it happens for more than 25% of the MDE settings).*

In case the hypotheses can be rejected, structural evolution is most probably no cause for the diversity of MDE settings in practice. However, in case the hypotheses can be accepted, future research on reasons for structural evolution is strongly motivated.

4. Studies

In Chapter 3 the potential effects of MDE settings on changeability and software development processes were discussed. Further, it was analyzed how MDE settings might evolve over time. However, data about MDE settings in practice is required to evaluate the hypotheses that were formulated in Chapter 3.5. To gain appropriate data, three studies were performed in context of this thesis.

The first study took two years (2010 and 2011). The focus of this initial study was on capturing examples for MDE settings that are applied in practice. The study was performed in cooperation with SAP AG and six MDE settings were collected. There were two further outcomes of this initial study. First, it became clear that state of the art techniques for modeling development approaches (especially existing process modeling languages) are insufficient to express differences between MDE activities. Second, the interviews revealed several hints that the current MDE settings are a result of changes to MDE settings formerly used. Thus, it was this initial study that led to the suspicion that evolution might help to explain why MDE settings in practice are so complex and diverse. As a consequence, this first study motivates the investigation of possible changes in MDE settings and the formulation of the hypotheses $H_{existence}$ and H_{common} .

The second study was performed in early 2012. Motivated by the outcomes of the first study, the aim was to collect data on the existence of evolution of MDE settings. A goal was to investigate what types of changes occur, and in particular whether structural evolution or even substantial structural evolution happens in practice. An investigation of the question whether a specific phenomenon exists in practice, naturally runs the risk that the phenomenon may not be detected. Given this risk, this second study was performed as a meta study. The approach was to systematically search the literature in order to identify reports on the application of MDE in practice. This avoided the expensive process of collecting data within companies. A further advantage was that the meta study provided access to reports from a broad spectrum of domains. The meta study showed that structural evolution does indeed play a role in practice (as will be discussed in Chapter 5). However, although the reports which were identified include enough hints on the existence of structural evolution, they do not include enough details to reason about motivations and triggers that caused this evolution.

To fill this gap a third study was performed from early 2012 to the beginning of 2013. The focus of this study was to collect data about MDE settings in practice and their evolution history. As a result, five MDE settings in practice and their evolution history could be collected in cooperation with Capgemini, Carmeq, and VCat. In addition, the history of one of the SAP case studies captured during the first study could be collected. Further a single evolution step could be documented in cooperation with Ableton AG.

In the following, the three studies are introduced in detail. The study design is described and an overview about the collected data is provided. Finally, threats to validity of the studies are discussed.

This chapter is partially based on [P3, P6, P2], and [P4].

4.1. Initial Study of MDE Settings in Practice

When the first study started, only occasional reports existed about the actual use and combination of MDE techniques in practice. In this context, a goal was to explore the diversity of the subject and provide space for unexpected insights. Therefore this first study was designed as a descriptive and exploratory field study to gain insights into how MDE techniques are applied in practice. Two forms of data were collected using semi-structured interviews. The main aim was to collect data about the concrete structure of MDE settings in practice (i.e. how are technologies and languages combined). In addition, data were to be collected to explore motivations for and use of the MDE settings.

4. Studies

4.1.1. Study Design

This first study was performed in cooperation with SAP AG¹. In the following the design of the study is described. Therefore, it is first described how the concepts of MDE settings map to the context of SAP. Then it is described how the case studies were chosen. Finally, the applied research method and the method how the resulting data was analyzed are described.

Conceptual Context

When performing an exploratory study about the practical manifestation and variety of a concept from theory, side issues might turn out to provide relevant insights. Therefore, it is helpful to provide a definition of used terminology in a way broad enough to include such side issues.

For this study, the following definition of the term MDE setting was presented to the interview partners: “a setting of MDE techniques that bases on different languages and tools, implementing different transformations and code generation or supporting the developers”. A defined goal was to capture manual and automated activities, such as model transformation or code generation applied during development.

In addition, a simple role model (as illustrated in Figure 4.1) was used to understand an interviewee’s view on the MDE setting under study. In this role model, tools, transformations and languages of the MDE setting are provided by *tool developers* and are used by *developers* to create software, which is further used by *users*.

MDE settings were captured and scoped from one of two perspectives. The first is that of the creation of an important intermediate product or a part of the software (e.g. “creating a business object”). The second perspective is that of a development tool within the MDE setting that supports specific parts of an implementation process. An example for the latter case is the perspective of the tool Service Implementation Workbench (SIW), which supports developers in generating a web service on the basis of a WSDL file (Web Service Description Language) and existing components.

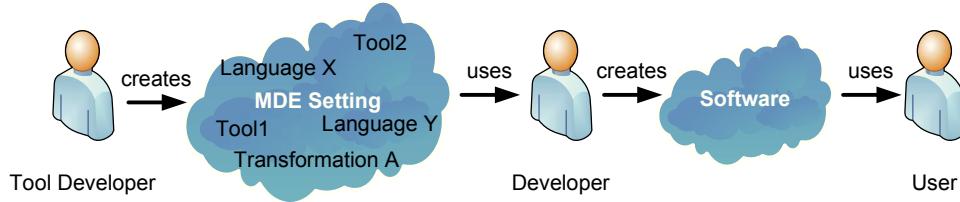


Figure 4.1.: Roles using and creating an MDE setting for the creation of software

Further, there are differences between the terminology used in practice and in theory. Two such differences played a role in most interviews. First, the term MDE is not necessarily used by interview partners to describe their MDE settings.

Second, the terminology of the roles changed. When the MDE setting was used internally, users were usually referred to as customers. However, when the MDE settings are products of SAP, the developers were often referred to as customers, while tool developers were simply called developers.

Choice of Cases

To identify appropriate MDE settings for the study, two contact persons within SAP provided a list of case studies. They chose MDE settings which are known within SAP for including the use of models and MDE techniques.

As a next step, the contact persons introduced the interviewer for each MDE setting to one or two interview partners. The interview partners chosen for their experience in development using the respective MDE setting, were SAP developers or consultants preparing SAP software for the customer. Otherwise, for some case studies from the development tool perspective (i.e. with a focus on the part of the MDE setting that is supported by a specific tool), the interview partners were tool developers (or

¹<http://www.sap.com/> (last access at November 9th, 2013)

software architects) who built the relevant tool. In case one or two interview partners agreed to support the study, the corresponding case study was included in the study. Initially, a set of seven case studies could be included in this study. Due to cancellation of one case study after the initial interview, six case studies were captured in the end.

Research Method

Interviews were used to capture case studies of MDE settings. This is relatively expensive in comparison with questionnaires, for example. In addition, a too low number of case studies does not allow empirical reasoning. However, this form of research has some important advantages when it comes to understanding complex phenomena and their drivers.

Focusing on case studies makes it possible to gain coherent pictures of complex MDE settings. This includes the different ways in which artifacts are used or reused in automated and manual activities, the order of activities, and the tools used for supporting the activities. This is even more important since MDE settings had not been captured explicitly within the cooperating company before.

Semi-structured telephone interviews were used to capture the case studies. Interviews have the advantage that misunderstandings can be resolved directly [214], which is beneficial when capturing complex phenomena. The structure of the interviews provided the basis to capture the MDE settings systematically. The semi-structured character of the interviews also supports the exploratory mode of the study.

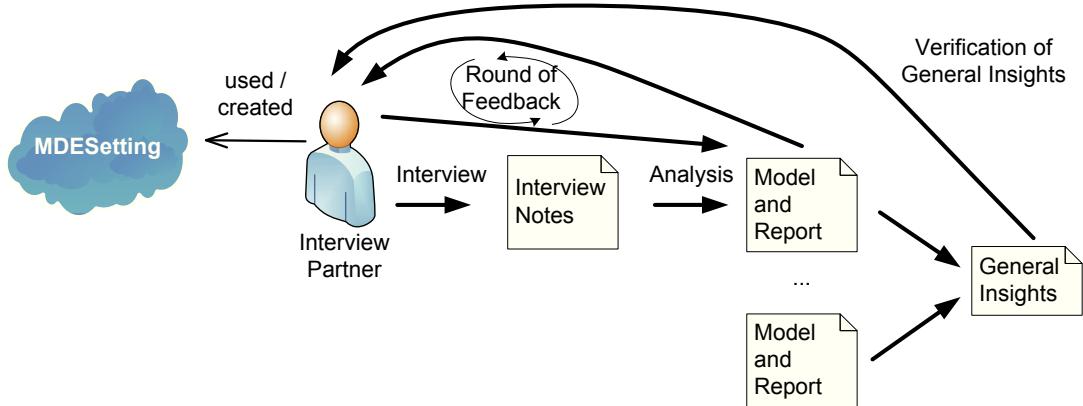


Figure 4.2.: Design of the study

The investigation of each case study started with an initial telephone interview with the main interview partner (as illustrated in Figure 4.2). All telephone interviews lasted between thirty and sixty minutes. They were performed by one interviewer who also took notes during the interview. To provide a frame, the initial interviews started with a short introduction of the goals of the study. As recommended for semi-structured interviews a set of question groups was prepared and used as capturing aids during the interviews. These question groups cover the following key topics (considering artifacts as models or source code):

- Used tools as well as modeling and programming languages
- Artifacts that are used and created during development
- Existing and created relations between artifacts
- Activities used to change, enrich, translate, generate, merge, compile, or interpret artifacts
- Degree of automation of individual activities
- Order of activities

4. Studies

- Performed (semi-) automated quality assurance activities on artifacts
- Responsible roles for different activities

In addition to the interviews the interview partners provided documentation material for four of the case study, such as application scenarios. In a next step, the information was structured and UML activity diagrams ([156]) were used to model a process of the captured activities. The activities were annotated with required input artifacts and produced output artifacts. In addition, a textual description of the results was created, to capture information about how relations between artifacts change, the degree of automation of the activities, responsible roles, and which tools and languages are used. Analyzing, structuring, capturing, and modeling the information took between one and one-and-a-half days work.

For various reasons, captured information always contains ambiguities and insufficiencies. One reason is that not for each case study the role of the developer (as shown in Figure 4.1) is fulfilled by an SAP member. In some cases the tool in the MDE setting is marketed through SAP. Consequently, terms like “developer” or “product” can be misleading. To handle such ambiguities and insufficiencies, the findings were provided to the interview partners for feedback. All in all, between four and ten rounds of feedback were performed per case study. Each round of feedback required formulating appropriate questions to tackle the ambiguities. Subsuming each round of feedback took approximately one day of work for formulating questions and adapting the models according to the feedback.

To evaluate the results for each project, a report was provided to the interview partners. In a further telephone interview, the findings (especially the captured models) were discussed. In three cases this led to some further small changes in the models. Preparing the final report together with the closing interview took another day of work. On average, a total of nine days of work were spent on the investigation of one case study.

Analysis of Data

After the elicitation, the data from the six captured MDE settings was analyzed. The quantitative goal of the study was to gain information about the structure of MDE settings in practice. This quantitative information was mainly captured within the models. After the study, the Software Manufacture Model language (which will be introduced in Chapter 6) was developed, in order to appropriately model the differences between the activities in MDE settings. After this language was introduced, models and transcripts from this first study were reconsidered. As a result, Software Manufacture Models could be reconstructed to model the whole MDE settings from the six SAP studies. From the models, it was possible to extract information like the number of automated activities or the number of languages that were used in combination. Further, it was possible to examine where different languages are combined to reach a separation of concerns (i.e. where different system aspects are specified using different languages). Based on the models it was also possible to identify where transformations are used to move content between artifacts of different levels of abstraction.

Inspired by grounded theory [46], transcripts and models of the different case studies were compared for the qualitative part of the study. This led to some general insights. Afterwards, the interview partners were asked to validate these general insights. In addition, a small questionnaire was used to ask for motivations of the introduction of the different case studies and for the software quality attributes that are aspired to be supported by the MDE settings.

Some time after the study was performed, the transcripts and the answers of the questionnaire were systematically revised to search for hints on evolution that happened to the captured MDE settings. Hints or concrete information on evolution were searched and categorized according to the change types that are introduced in Section 3.4.

4.1.2. Overview of Case Studies from SAP

The six captured objects of study are summarized in Tables 4.1 and 4.2.

First, there is the MDE setting used to develop business objects (an SAP specific type of components that captures functionality specific to a corresponding business need) for the feature package 2.0. This object of study (referred to as *BO* in the following) describes an old development approach, which was used some 100 times in nine teams from 2004 onwards. Today *BO* is substituted by a new development

Table 4.1.: Summary on captured case studies from first study

Case Study	Full Name
BO	Development of Business Objects for the feature package 2.0
BRF	Personalization of business processes using the tool business rule framework
BW	Definition and automated execution of reporting with the tool BW
Oberon	Development of a user interface and application based on business objects using the tool Oberon
SIW	Development of web services using the tool service implementation workbench
VC	Development of a user interface for SAP Net Weaver applications using the tool visual composer

approach, through substitution of tools. Motivation for the initial introduction of *BO* was to enhance quality, but also to reach transparency and a unified procedure for business object development. For this object of study 19 activities were captured.

The second object of study is the MDE setting for the development of web services with the tool service implementation workbench (referred to as *SIW* in the following). Since about 2003, the tool has allowed the development of a service on the basis of a web service description (formulated in web service description language (WSDL) [42]) supporting reuse of existing functionality. The introduction was intended to avoid redundant implementations and the resulting effort for maintenance. *SIW* is used throughout the SAP Business Suite. *SIW* has since been partially redesigned. For this object of study 16 activities were captured.

Further objects of study describe the development of user interfaces and their coupling to services or business objects. First, there is the development of a user interface for SAP Net Weaver applications using the tool visual composer [33]. This object of study is referred to as *VC* in the following. *VC* is used since 2006 ca. four times at the consulting company FIVE1 GmbH & Co. KG. Motivation for its introduction was a faster development as well as the ability to create prototypes. Between the different versions *VC* was often extended. For this object of study 30 activities were captured.

Then there is the development of a user interface and application based on business objects using the Oberon tool. *Oberon* has been used some 120 times since 2008. Motivation for the introduction of *Oberon* was the simplification of user interface technologies that are used in context of By-Design. In addition, the flexibility was enhanced through the use of similar models at design time and runtime. In this context, the number of tools was reduced and different areas of development were integrated into one tool. For this object of study 25 activities were captured.

The fifth object of study deals with configuration of business processes using the tool “business rule framework” (*BRF+*). This object of study is referred to as *BRF* in the following. The customer specific configuration can be seen as part of development of software systems, too. However, *BRF* can be applied by experts on the customer’ side. Since 2008, *BRF* has been used at least 70 times within SAP standard applications and adapted many times. Motivation for the introduction was cost reduction and increased transparency and agility of development. For this object of study 23 activities were captured.

The last object of study allows the definition of a mapping between data from different tools. The extraction of data, mapping, and generation of a HTML report is then performed automatically and periodically during runtime. The tool that supports this definition and automated execution is Business Warehouse, which is referred to as *BW* in the following. Since 1996, *BW* had some 19 000 customers (or users). *BW* was introduced to substitute the predecessor EIS (Execution Information System) to enhance performance and to enable consolidation of data from multiple systems. *BW* has since undergone various changes, due to changes of the underlying language (ABAP) or to integrate different transformations. For this object of study 19 activities were captured.

4. Studies

More details about the case studies can be found in [P3].

Table 4.2.: Key data on captured case studies from first study

Case Study	Approximate number of applications	Number of captured activities	Years in use
BO	ca. 100 times in nine teams	19	> 2
BRF	at least 70 times within SAP standard applications	23	> 3
BW	ca. 19 000 customers	19	> 14
Oberon	ca. 120 times	25	> 3
SIW	used throughout whole SAP Business Suite	16	> 8
VC	ca. 4 times at FIVE1 GmbH & Co. KG	30	> 5

4.2. Meta Study of Evolution

The meta study was performed to figure out whether structural evolution plays a role in MDE settings from practice. The literature was searched for reports on MDE in practice.

4.2.1. Review Process

In this section the process of the meta study is described.

Conceptual Context

The aim of the study was to identify reports on the application of MDE in practice. For this reason, several publication media were examined. Publication media that are specialized to MDE and DSL related topics were chosen, namely the proceedings of the MODELS (International Conference on Model Driven Engineering Languages and Systems) conferences from 2007 to 2011, ECMFA (European Conference on Modelling Foundations and Applications) and ECMDA-FA conferences from 2007 to 2012, the proceedings of the Workshop on Models and Evolution (ME), as well as its predecessors MCCM (Workshop on Model Co-Evolution and Consistency Management) and MoDSE (Workshop on Model-Driven Software Evolution) from 2007 to 2011, the proceedings of the OOPSLA Workshops on Domain-Specific Modeling from 2007 to 2011, as well as the Software and Systems Modeling journal (SoSyM) from 2007 to 2012, including papers published online until the end of July 2012. In addition, an online key word search was performed and references in reviewed papers were used to identify further reports. In addition, the ACM digital library was used for keyword search in the proceedings of the ICSE conference.

The search was for reports on the application of model-driven techniques or domain-specific modeling languages in practice. Note that there was a focus on no-purchase tool chains. As a result, it was possible to identify fourteen reports that describe MDE introduction or use ([10, 15, 50, 72, 108, 131, 146, 172, 182, 198, 205] and three case studies in [98] as summarized in Table 4.3). Cases where different aspects of the same application are described in multiple papers are counted as one report.

Choice of Cases

Evolution requires time. Therefore the reports were filtered to ensure that the reported period of time is long enough to allow the observation of evolution. Reports that focus on the initial introduction of MDE or on settings that were used for a single project only were not suitable. Therefore, six reports had to be excluded ([10, 50, 131, 172, 198, 205] as well as the telecom case study in [98], where the

Table 4.3.: Identified reports within the meta study

Publication medium	identified papers
ICSE (International Conference on Software Engineering)	[198](2007) [10](2010) [98](2011)
Journal on Software and Systems Modeling	[146](2011)
UML Modeling Languages and Applications	[131](2004)
MoDELS (International Conference on Model Driven Engineering Languages and Systems)	[15](2005), [72](2007)
ME (Joint MODELS'09 Workshop on Model-Driven Software Evolution (MoDSE) and Model Co-Evolution and Consistency Management (MCCM))	[205](2009)
Proceedings of the ACM OOPSLA 2003 Workshop on Domain-Specific Modeling Languages	[50](2003), [108](2009)
ECMDA-FA (European Conference on Model Driven Architecture-Foundations and Applications)	[182](2007), [172](2009)

described example was only used during one project). The remaining seven reports were chosen for the inclusion in the dataset ([146](CsTe), [72](CsBA), [182](CsFO), [108](CsFBL), [15](CsMo), as well as the case studies of the printer company (CsPC) and the car company (CsCC) [98]).

Analysis of Data

The reports taken from literature are written for a wide range of reasons, and therefore differ in focus from the meta study. Due to this diversity, the reports cannot be an appropriate source for qualitative insights. However, it is this diversity of underlying motivation that makes the reports suitable for quantitative evaluation of hypotheses on existence of structural evolution ($H_{existence}$ and H_{common}).

To extract the required quantitative information from the reports, they were systematically examined. Indicators of concrete information about evolution were extracted. Where possible, these collected hints were assigned to concrete change types and rated as structural or non-structural (according to Section 3.4). However, due to the character of the data source, the information about evolution is not necessarily complete. Further, it is not possible to determine whether all changes are part of a single evolution step or of multiple steps.

4.2.2. Overview of Cases Studies

The seven case studies from the chosen reports are summarized in Table 4.4. The reports stem from different domains, such as the telecommunication industry, financial organizations, and development of control systems.

The *case study of financial organization* (CsFO) [182], reports on the adoption of MDE in a financial organization. The report was published in 2007 and the described adoption of MDE started around 2005. During the pilot phase, the setting was used in different projects by 60 people. The adoption was envisioned by the QA department² of the financial organization to produce high quality software “efficiently and effectively” ([182]).

The *tool vendor case study of Function Block Language* (FBL) (CsFBL) is presented in [108]. Here a tool vendor reports how the language FBL changed together with its engineering environment. The report was published in 2009 and describes changes to the FBL setting over a period of 20 years. FBL supports “development of real-time control programs for distributed environments” [108].

²assumption of the author: “QA department” refers to a department that is responsible for quality assurance

4. Studies

The *case study of migration of a banking application* (CsBA) stems from [72]. Here Fleurey et al. present a process for the migration of systems to new platforms. To apply the process, it is proposed to substitute the transformations such that they fit the current use case. In addition, they describe how they actually adapted the process to apply it for the migration of a banking application. The report was published in 2007. The banking company decided to apply that approach because of quality assurance capabilities and because of the low proposed price.

The *Telefónica case study* (CsTe) is presented in [146]. The use of a DSML for the generation of configuration files is reported. The report was published in 2011.

The *Motorola case study* (CsMo) is described in [15]. Here Baker et al. describe the use of MDE within Motorola. The report was published in 2005 and describes experience over more than 15 years. Motivation for the introduction of MDE was the increasing complexity of systems.

The *case study of the printer company* (CsPC) is described in [98]. The report was published in 2011. The first pilot project that used model driven design started in 1998.

Finally, the *case study of car company* (CsCC) is also presented in [98]. The report was published in 2011. The motivation was to handle an increasing share of software during development.

Table 4.4.: Summary of case studies from meta study

Case Study	Full Name	Source
CsFO	Case study of financial organization	Shirtz et al. [182]
CsFBL	Tool vendor case study of Function Block Language (FBL)	Karaila et al. [108]
CsBA	Case study of migration of a banking application	Fleurey et al. [72]
CsTe	Telefónica case study	Mohagheghi et al. [146]
CsMo	Motorola case study	Baker et al. [15]
CsPC	Case study of printer company	Hutchinson et al. [98]
CsCC	Case study of car company	Hutchinson et al. [98]

4.3. Follow-up Study of MDE Settings in Practice and their Evolution

When the third study started, it was already known from the second study that structural evolution exists and is common in practice. However, there was still a lack of knowledge about structural evolution and motivations that drive structural evolution. To address these issues, the third study was started as an extension of the first study. It was the main goal to capture not only MDE settings, but also information about the evolution history of these MDE settings. While the first study was exclusively performed in cooperation with SAP, it was an additional goal of this third study to get also data from a more diverse set of companies.

As in the first study, quantitative data and qualitative data was captured. Additional MDE settings were modeled and the evolution history was documented. Also qualitative data about motivations and triggers for the evolution steps was collected.

4.3.1. Study Design

Since the third and the first study have a similar character and partly similar goals, the design of this first study was mainly adopted. A description follows of how the design of the third study was adapted to fit the new needs.

Conceptual Context

The conceptual context remained mostly the same as in the first study. At the start it seemed that the topic of the evolution of MDE settings might become an additional challenge for the communication. Fortunately, it turned out that it is relatively easy to inquire about the evolution history by asking question like: “Did you always use tool X?” or “How did you develop artifact Y before transformation Z was introduced?”.

Choice of Cases

Since a goal was to collect MDE settings from different companies, the process of choosing case studies had to be adapted. First, most companies are very cautious about providing detailed data to external researchers (as is necessary for case studies). This problem was approached in this study by contacting alumni students of the Hasso Plattner Institute and using personal contacts in different companies. The request was accompanied by a short description of the project and the data to be captured. Eventually the contacted persons passed the request on to colleagues. When the persons in charge of such a project were interested in the study they answered the request.

All in all, this led to responses from six projects (from five companies). Unfortunately, in two companies the management did not agree to the participation due to confidentiality reasons. However, three companies agreed to participate in the study: Capgemini (2 projects with 3 MDE settings), VCat, and Carmeq. In addition, one of the contact persons of the first study at SAP agreed to resume the participation in the study and helped to document the evolution history for one of the captured MDE settings. Finally, it was possible to document a single evolution step of an MDE setting from Ableton. However, in this case the corresponding MDE setting was not captured in detail.

Research Method

As in the first study, the goal was to collect detailed case studies. Again semi-structured interviews were used. To address the new issues, the method of eliciting the MDE settings was changed. The feedback process was augmented with a third interview (substituting the email contact). Further, the MDE settings were directly modeled using Software Manufacture Models. In addition, new questions were included about how the MDE settings evolved over time. This includes questions for motivations and triggers for the captured evolution steps. Finally, for the last two case studies a couple of students accompanied the interviewer during the interviews and were encouraged to ask questions, too. All evolution steps were planned and/or performed before they were captured for the study. As a result, models of the MDE settings and evolution steps were captured together with records from the interviews.

The more direct form of communication in this adapted research method, together with the direct use of Software Manufacture Models, and the experience the interviewer gained during the first study, reduced the effort for elicitation of an MDE setting to five work days per MDE setting.

Analysis of Data

The records from the interviews were systematically inspected and coded following the constant comparison method described in [177]. At the start, a set of preformed codes was used. These referred to the motivation for an evolution step, the institution or role that triggered the evolution step, and the institution or role that implemented the evolution step. During the inspection of the records, codes were added when necessary (e.g. for external influences on the evolution). Based on these codes it was possible to derive several observations.

The collected quantitative data (i.e. the models) were analyzed similar to the models from the first study (except that the models were already Software Manufacture Models). Further, the collected information about the evolution history was analyzed. The evolution steps were further categorized according to the change types presented in Section 3.4.

4.3.2. Overview of Cases Studies

In the context of this third study, five new MDE settings were captured together with their evolution history. In addition, the evolution history of the BO case study and an evolution step of a case study

4. Studies

from Ableton were documented. This third study has a focus on structural evolution which reflects in the questions that were used. In most cases, non-structural evolution steps (e.g. language evolution) were not captured. Consequently, this data cannot be used to make quantitative statements about distribution of structural compared to non-structural evolution steps. In the following, an overview in given about the captured case studies is given (summarized in Tables 4.5 and 4.6)³.

Table 4.5.: Summary on captured case studies from third study

Case Study	Company	Full Name
Cap1	Capgemini	Capgemini case study 1
Cap2a	Capgemini	Capgemini case study 2
Cap2b	Capgemini	Capgemini case study 3
VCat	VCat Consulting GmbH	Development of TYPO3 based websites
CarmeQ	CarmeQ GmbH	Development of AUTOSAR standard documents CarmeQ
Ableton	Ableton AG	Development of sound libraries for users of the software Live

Capgemini First Case Study (Cap1)

The first case study (Cap1) was captured in cooperation with Capgemini⁴ and is used in a project that has run for four years. In this project Capgemini builds software for a customer. There are two interacting MDE settings involved. The first MDE setting is used by the customer to collect requirements and create or prepare parts of the specification. The second MDE setting is applied within Capgemini to create prototypes, generate the specification, and to implement the software. This second MDE setting and its history were captured. The MDE setting is specific for the project, which holds especially for the generator that is used. Initially, a Capgemini internal standard generator was in use, which was soon substituted by the project specific generator. In consequence, the generator can be flexibly changed or extended. The case study was captured in summer 2012. Sixteen activities and eight historic versions of this MDE setting were documented (including the MDE setting that was in use at the time of the interviews. Seven structural evolution steps were identified.

Table 4.6.: Key information on captured case studies from third study

Case Study	Cap1	Cap2a	Cap2b	VCat	CarmeQ	Ableton	BO
Number of modeled activities	16	18	27	10	25	–	19
Years in use	4	3	5	7	8	2	>2
Number of captured evolution steps	7	5	6	2	5	1	7

Capgemini Second and Third Case Studies (Cap2a and Cap2b)

Also the second and third case studies (Cap2a and Cap2b) were captured in summer 2012 in cooperation with Capgemini. The two MDE settings are parts of the same project. This project aims at providing

³Due to a request, names of artifacts and activities from the Capgemini case studies are partially substituted within this thesis, in order to ensure confidentiality of the actually investigated projects.

⁴<http://www.capgemini.com/> (last access at November 9th, 2013)

two MDE settings that are used by a customer of Capgemini. Both settings can be applied in the same customer projects. They aim at reaching similar goals for different use cases.

MDE setting Cap2a has been in use for three years. The team that developed the MDE setting Cap2a consisted initially of one person and later three to four people. For Cap2a, 18 activities were documented as well as six historic versions of the MDE setting (including the MDE setting that was in use at the time of the interviews). Five evolution steps were identified. Three of these evolution steps are structural.

MDE setting Cap2b has been in use for five years. The team that developed the MDE setting Cap2b consisted initially of one person and grew for a short phase of about half a year to four to five members. For Cap2b, 27 activities were documented as well as seven historic versions of the MDE setting (including the MDE setting that was in use at the time of the interviews). Six evolution steps were identified. Five of these evolution steps are structural.

Development of TYPO3 based Websites (VCat)

The fourth case study was collected in cooperation with VCat Consulting GmbH⁵. The documented MDE setting supports development of websites that rely on TYPO3 as underlying content management systems (CMS). Motivation for that MDE setting was to improve productivity through automation and standardization. The case study was captured in winter 2012/2013. At VCat the MDE setting has been developed and used for seven years (one year in its current version). For this case study, 10 activities were documented as well as three historic versions of the MDE setting: the current version, a historic version, and a planned version. Two evolution steps were identified.

Development of AUTOSAR Standard Documents (CarmeQ)

The fifth case study was captured in cooperation with CarmeQ GmbH⁶. The captured MDE setting is used to create documents of the AUTOSAR standard⁷, including models and tables. The case study was captured at the start of 2013. Various versions of the MDE setting have been in use since 2004. For this case study, 25 activities were documented as well as six historic versions of the MDE setting (including the MDE setting that was in use at the time of the interviews). Five evolution steps were identified.

Development of Sound Libraries for Users of the Software Live (Ableton)

The major product of Ableton AG⁸ is a system called Live, which provides artists and musicians with an environment for musical compositions and productions. An important part of the business of Ableton is the development of libraries, which provide users with a collection of presets for instruments included in Live.

The case study that was captured covers the changes that are currently applied to this development of libraries. The actual MDE setting of this development could not be captured in detail. The captured evolution step was documented at the start of 2012.

Development of Business Objects for the Feature Package 2.0 (BO)

The MDE setting of the case study BO was already captured during the first study and is described in Section 4.1.2. In the context of this third study, the evolution history of BO was documented. Seven evolution steps were collected.

Summary

As summarized in Table 4.6, the captured MDE settings were in use for between two and eight years. Each captured model includes between 10 and 30 activities. For each of the different cases studies between one and seven evolution steps were captured (33 evolution steps overall).

⁵<http://www.vcat.de/> (last access at November 9th, 2013)

⁶<http://www.carmeQ.de/> (last access at November 9th, 2013)

⁷<http://www.autosar.org/> (last access at November 9th, 2013)

⁸<https://www.ableton.com/> (last access at November 9th, 2013)

4.4. Threats to Validity

As discussed in Section 2.5, the categorization of validity that was introduced by Wohlin et al. [214] is applied. To address the observational character of the three studies, threats to validity are discussed under the following viewpoints: Construction validity is discussed in terms of the question: “*Are data and effects captured appropriately?*”. External validity is discussed in terms of the question: “*To what extent can the results be generalized?*”. Finally, conclusion validity is discussed in terms of the question: “*Are the conclusions that are drawn correct?*”. The fourth category – internal validity – is not discussed, since it is specific to experiments.

In the following, the validity of the studies is discussed. Threats that concern the collection of data are discussed with respect to the three different studies. Correspondingly, threats that concern the use and analysis of the data are discussed with respect to the five empirical research aims of this thesis: first, to learn about the structure of MDE settings in practice; second, to evaluate the hypothesis H_{traits} ; and third, to evaluate the hypothesis $H_{changeability}$. For these three aims, data on the collected MDE settings from the first and third study are used (i.e. 11 case studies: six from SAP, three from Capgemini, one from VCat, and one from Carmeq). The fourth aim is to evaluate the hypotheses $H_{existence}$ and H_{common} . For this, records from the first study are used as well as reports collected in the second study (i.e. 2 datasets with 6 and 7 cases, respectively). The final aim is to learn more about motivations and triggers of structural evolution. For this, mainly data on documented evolution histories from the third study are used (i.e. 7 case studies: one from SAP, three from Capgemini, one from VCat, one from Carmeq, and one from Ableton).

4.4.1. Construction Validity

Construction validity (i.e. “Are data and effects captured appropriately?”) can be affected by design threats as well as social threats.

Design Threats

The main challenge in the design of the studies is to ensure that the data is captured correctly and completely. Due to restrictions, e.g. the time that was available for the interviews, it is always necessary to make a trade-off between correctness and completeness.

The information that is captured is the first and third study is rather complex, which concerns especially the details about how the relations between artifacts change. Therefore, it cannot be excluded that there are faults in the captured models. To minimize the numbers of faults, the design of the studies includes rounds of feedback. In addition, the approach was improved in the third study by an additional interview instead of email feedback. As a consequence it can be expected that the correctness is appropriate to use the models as a data basis for further reasoning.

Further, there are two possible sources for incorrectness or incompleteness in the evolution histories that were captured during the third study. The interviews had a focus on structural evolution, and as a consequence it cannot be estimated how often non-structural evolution steps occurred between the captured structural evolution steps. This drawback could be taken into account when designing the study, because there is no plan to use the data to reason about the relative share of structural evolution compared to non-structural evolution.

Secondly, two types of data were captured: quantitative data on the MDE setting as well as quantitative and qualitative data on the evolution history. Thus, it has to be considered whether capturing one type of data has influence on the quality that can be reached for the other type of data. Indeed there was the challenge to split the interview time such that an appropriate correctness and completeness of both data types can be reached. Apart from this trade-off in time, a good quality of the captured data on the MDE setting is a precondition for targeted questions on the evolution history. Due to this interconnection it was decided to focus first on a high correctness of the captured data on the MDE setting. This led to the benefit that questions on the evolution history could be asked quite intuitively. Further, it was ensured that there is still enough scope for capturing qualitative data about the motivations and triggers for the evolution. Based on the feedback from the last interviews of each case study, it can be assumed that the captured data is sufficiently correct and complete.

Due to the character of the data sources in the second study, the information about change types is most probably incomplete. It is thus probable that the amount of occurrences of structural evolution in practice is even underrepresented in the results of the second study. A similar tendency towards an underrepresentation of structural evolution holds for the hints on evolution that can be found in the records of the first study. This is due to the fact that evolution was not the focus of this first study. However, it is the aim to use these data to study the existence and commonness of structural evolution. Here it even adds to the validity that the underrepresentation cannot lead to a wrong acceptance of the hypotheses $H_{existence}$ and H_{common} .

Social Threats

In addition to the design of a study, social threats to the correctness and completeness of the captured data also have to be taken into account. Such social threats might play a role in the first and third study, but not in the second study, which was a meta study. One social threat is that the interviewee might filter information that he provides during the interview for various reasons.

The first reason is *hypothesis guessing* [214], which means that a participant in a study might base her behavior on her guesses about the hypothesis of the study. This can be a threat for the correctness and completeness of results. During an interview, the interviewee might be eager on presenting specific information he expects to be important for the study, or equally might omit information, threatening the completeness of the data. To compensate this effect, capturing aids in the form of question groups were used during the interviews. This way the interviewer was supported in following a systematic approach when asking questions.

The second reason is *evaluation apprehension* [214], which means that interviewees might be afraid of being evaluated. In the studies here, the interviewees might fear that their MDE settings, projects, or company could be evaluated negatively. As a consequence they might leave out information about failure. In the first and third study this can lead to an underrepresentation of reports on problems with changeability concerns or effects of disadvantageous combinations of MDE settings and processes. During the interviews, the questions were focused on the MDE setting. Thus, the quantitative data might reveal that a hard changeability concern is strongly affected or that the manifestation of an MDE trait is process-relevant. However, amongst other issues, this threat prevents the data from the studies being used to judge the actual extent of negative effects that might arise by hard changeability concerns or disadvantageous combinations of MDE settings and processes.

Within the third study, a further effect of the evaluation apprehension might concern the data on motivations and triggers for the different evolution steps. As motivations for changes are often disadvantages of or problems with the preceding MDE setting, it is possible that interviewees underplay corresponding information. For the same reason it is probable that existing plans for further evolution are omitted. This problem was approached by targeted and positively formulated questions about what improved after an evolution step and whether there were plans for future improvements.

In general it cannot be ruled out that information about evolution is incomplete. However, the quantitative data of the third study is not used for reasoning on distribution or frequency of evolution steps. Further, within the qualitative data possible incompleteness can be accepted. Nonetheless, the qualitative data that can be collected provides valuable insights into existing motivations for evolution.

A social threat to the construction validity are the *experimenter expectations* [214]. This means that questions formulated by an interviewer with strong expectations about the results might lead to a bias in the results. Since all interviews of the first and the third study were performed by the same main interviewer, this threat will be discussed here.

All research aims that are pursued on the basis of the captured data are potential reasons for such experimenter expectations. First, data on MDE settings of the first and third study were collected with the research aim of learning about the structure of MDE settings in practice. Possible expectations of the researcher might concern the complexity of the structure of the MDE settings. For example, a large number of languages might be expected. Alternatively, an academic interviewer, might expect an extensive use of transformation technologies, such as ATL. Thus there is a risk that the questions are too focused on established MDE technologies. This might lead to the omission of other tools within the interview. To tackle this risk the capturing aids (i.e. the question groups for the semi-structured interview) are formulated independently of technology. For example, the interviewer was reminded to ask open questions concerning existing automated steps.

4. Studies

Second, the same data on MDE settings is used to approach whether manifestations of MDE traits occur in a board spectrum and whether the hard changeability concern *unexpected loss of content* is affected. Only one of the MDE traits (length of activity chain) was already discussed at the time when the interviews of the third study were being conducted. Hard changeability concerns were already partly identified when the last interviews of the first study were performed. Consequently, it is possible that the interviewer inquires more explicitly about structures in the MDE setting that might affect MDE traits or hard changeability concerns. However, this only increases the probability that existing structures are identified and thus does not lead to a bias (i.e. no identification of phenomena that do not exist is to be expected).

Third, data from the first study was used to evaluate whether structural evolution exists. This led to no bias in the elicitation of the data, since structural evolution was not expected when the interviews were performed. The hypothesis $H_{existence}$ was rather a result of the exploratory part of the first study.

Finally, data from the third study was used to approach motivations for structural evolution. However, the interviewer had no expectations about such motivations and the corresponding part of the study was designed to be exploratory. Therefore, little bias is expected here.

4.4.2. External Validity

There are two main threats for the external validity of the studies (i.e. “To what extent can the results be generalized?”).

Interaction of History and Treatment

One threat to external validity is the *interaction of history and treatment* [214], which means that external events might influence the measured results. For the research aims of this thesis, possible effects of this threat on the extent to which the results can be generalized can be neglected. The reason is that the interviews for the various case studies were spread over a long period of time. Thus, the same external event would only have had influence on interviews of one or two case studies.

Interaction of Selection and Treatment

The other threat to external validity is the *interaction of selection and treatment* [214], which refers to whether the selection of cases in the study is representative such that the results can be generalized to other cases.

As mentioned above, for the investigation of an MDE setting’s influence on changeability and processes, data from the first and third study is used. This includes 11 MDE settings from four companies.

In general, four companies are not sufficient to generalize the results for all domains of software development. Despite these limitations, the eleven case studies can provide an adequate initial insight on the structure of MDE settings in practice as well as the variance of manifestations of MDE traits and the way hard changeability concerns are affected in MDE settings within and between different companies.

Further, the focus of the third study was extended to capture information about the evolution history of the MDE settings. Due to this changed focus it cannot be excluded that there is a selection bias (towards MDE settings with a longer evolution history) for the case studies that were collected after the focus changed (which are the Capgemini, Carmeq and VCat case studies). Thus, in the event that a longer evolution history promotes the occurrence of manifestations of MDE traits and hard changeability concerns, this selection bias increases the probability that process-relevant manifestations of MDE traits can be found in the case studies. Fortunately, this affects only five of eleven case studies. Nonetheless, the observed extent to which hard changeability concerns are affected cannot necessarily be generalized to MDE settings that rarely change.

The investigation of structural evolution will be approached in this thesis on the basis of data from the first and second study. This includes records from the 6 SAP case studies. These case studies have no selection bias concerning the topic of evolution. However, all six case studies stem from the same company. On the other hand there are 7 case studies collected during the meta study to learn more about the occurrence of evolution steps. Although a systematic selection method was applied, a small selection bias towards case studies with evolution cannot be excluded. In exchange, these seven case

studies stem from different countries, companies, and domains. However, the fact that data from two different sources is used helps to minimize selection bias as well as biases due to corporate culture.

It is questionable whether the results about existence of structural evolution can be generalized to all domains of software engineering. Without having data about multiple MDE settings for each domain, it is not possible to say, whether observed differences are specific to the setting or to the whole domain. However, thanks to the data from literature reports, which stem from different domains, it is possible to draw conclusions that are not specific to a single domain.

Finally, the aim to gain insights about motivations for structural evolution is approached on the basis of data from the third study. This includes seven evolution histories from five companies (Capgemini, SAP, Carmeq, Ableton, and VCat).

It is always difficult to draw general conclusions from a few case studies. Thus, a broader set of data that captures more domains of software engineering and different companies would be helpful to further substantiate the observations made in this study.

Despite the small number of case studies, it is fortunate that different companies are under study. Further, all observations that will be presented in Section 5.2.2 are based on at least two of the case studies. Thus, there are strong indications that the observations are not specific to a single domain.

As mentioned above a selection bias towards case studies with long evolution histories cannot be excluded for the third study. It is possible that motivations and triggers for evolution are different in projects where MDE settings often change compared to projects where MDE settings rarely change. Consequently, it is not necessarily possible to generalize the observations to MDE settings that change rarely.

4.4.3. Conclusion Validity

Finally, the conclusion validity (i.e. “Are the conclusions that are drawn correct?”) deals with whether the conclusions are statistically correct, i.e. whether the data is sufficient to accept or reject the hypotheses correctly. Therefore, this type of validity will be discussed separately for the different study insights in the Chapters 5 and 9.

5. Study Results: MDE in Practice

Based on the data from the three studies (see in Chapter 4), the nature of MDE in practice is examined in this chapter. First the collected data on MDE settings from the first and third study is used to explore characteristics of MDE settings. Afterwards, insights on the nature of structural evolution in practice are presented and discussed, as summarized in Figure 5.1.

This chapter is partially based on [P3] and [P4].



Figure 5.1.: Overview of this chapter: investigated aspects of MDE are the structure of MDE settings and their structural evolution.

5.1. MDE Settings in Practice

In this section data on the captured MDE settings is summarized. Subsequently, some observations on MDE settings that have been made on the basis of the first study are described.

5.1.1. General Results

In this section it is examined how different languages and automation activities are combined in MDE settings that were captured and modeled in the first and third study. Further, it is discussed how the data fits together with results from previously published studies and the validity conclusions that might be drawn from this data is discussed.

Languages

In the captured MDE settings different types of models and artifacts are used. In the following the languages that are used in the case studies are summarized. The languages are clustered in four categories: general purpose and programming languages, DSLs, modeling languages, and languages without a category. During the first study (i.e. for the SAP case studies) the interviewees have been asked to rate the categories for languages used in their projects. However, it turned out that the gained results are often ambiguous. Therefore, no time of the interviews of the third study was invested for asking the interviewees to rate the types of the used languages. Instead, a short rating was made during the analysis of the data, based on the records. Note that these ratings (as shown in Table A.1) shall help to orientate, only. For many of the languages and data formats other options for rating exist.

All in all 56 different languages were identified for the 11 captured MDE settings (as summarized in Table A.1). The captured MDE settings include five general purpose languages or programming languages, e.g. ABAP or Excel (which was in one case rated as GPL and in one case not). 21 domain specific languages (DSLs) can be identified, e.g. HTML or status and action models (S&A Models). In addition, the captured MDE settings include 14 modeling languages, e.g. BPMN. Finally, 15 further artifact types can be identified, e.g. WSDL artifacts (web service description language), or rulesets (Regelsprache und Modellierungssprache). Each MDE setting combines on average 6.36 languages. Of the eleven MDE settings, seven included between 3 and 6 languages, while four included between 7 and 18 languages. It is worth mentioning here that the case study VCat, which is not called “model-driven” by the corresponding developers, was one of the case studies with the most combined languages (11 languages). All languages are summarized in Appendix A.

Activities

Per case study between 10 and 30 activities were captured. During this elicitation, activities of different degrees of detail were captured. It can be differentiated between minimal changes or transitions and complex changes or transitions. Within a minimal change or transition only one or a hand full of connected model elements are changed or transcribed to another artifact, respectively. In contrast, within a complex change or transition a potentially big part of the model is changed or transcribed to another artifact, respectively. Activities that represent minimal changes and transitions are often described in manuals and tutorials, e.g. for learning to handle a tool. However, for getting an overview and analyze an MDE setting they might be too fine granular. An example for such minimal activities in the study is the creation of a “*floorplan*” using Oberon. Here, a standard “*floorplan*” is automatically initialized in form of a template that can later be filled with information.

Degree of automation Considering the more complex activities 38 fully automatically supported generation and transformation activities are captured. This contains automated activities, where a complex change is performed on a model (or artifact), or automated activities, where on the basis of one or more input models (artifacts) at least one output model (artifact) is created or manipulated with a complex transition from the input models to the created/manipulated model. Variants of an automated transformation or generation are counted as one activity in this statistic. Further, 12 complex semi-automated generation or transformation activities are captured, e.g. the generation of a proxy using the Service Implementation Workbench (SIW), and one complex semiautomated template instantiation.

Many automated activities do not interfere with the work of developers, since they are not followed by further manual activities. However, there in only three of the captured MDE settings no automated activities have succeeding manual activities. For example, in VC the generation of a UI model on the basis of a BPMN model is followed by manual activities. The same holds for the generation of code and the generation of a WSDL file in SIW. Similarly, in BO the generation of code and the instantiation of templates are followed by manual activities. Also in VCat, Carmeq, Cap1, Cap2a, and Cap2b some automated activities are located before further manual activities, as summarized in Table 5.1.

Verification activities In addition, seven of the case studies contain – aside from tests on code level – fully automated and semi-automated verification activities. In SIW it is verified that a mapping between service descriptions in different languages is complete. In BO a set of checks is applied on the business object model. Further, in BRF certain analysis steps are applied of the rule sets. In addition, the BRF rule sets can be simulated. Carmeq includes 6 fully automated activities that support verification in combination with 3 manual verification activities. Finally, for each of the case studies Cap1, Cap2a, and Cap2b a fully automated verification activity could be identified.

Compilation vs. Interpretation Where applicable, the interviewees were asked whether the resulting system implementations are compiled or interpreted (results shown in Table 5.1). Interestingly, the answer was only in two cases clearly “compiled” (SIW) or “interpreted” (Oberon). Within the object of study BO the ABAP code is compiled, but the additionally used S&A models are interpreted.

For the user interface in VC it depends on the use case whether interpretation is sufficient or compilation has to be performed. For BW it was reported that a mix of both, compilation and interpretation, is used. The share of interpretation grows over time, as the dynamic parts of the underlying ABAP increased. Similarly, for BRF the answer was that a mix is used for the created ABAP code as well as for the created rules. In Cap1, the resulting Java code is compiled to Java-Bytecode, which is further interpreted. The TYPO3 system created in case study VCat is directly executed. Finally, case studies Carmeq, Cap2a, and Cap2b do not result in software, but in documents that are meant to be read by humans.

Separation of Concerns

The principle of separation of concerns [194] is inbuilt in some of the case studies. Five of the settings base this separation on the use of different languages. For example, in BO behavioral parts of the business object can be defined separately in S&A models, while the rest of the system is implemented in ABAP code. The separation is used to enable that certain parts of the business object can be implemented at a

Table 5.1.: Distribution of automated generation and transformation activities as well as distribution of compilation and interpretation

Object of Study	# modeled automated generation and transformation activities	# automated generation and transformation activities without succeeding manual activities	compilation / interpretation
BO	1	1	ABAP code is compiled ; S&A models are interpreted
SIW	3	2	compiled
VC	2	1	mixed (depends on the use case)
Oberon	0	0	interpreted
BRF	2	0	mixed
BW	1	0	mixed
VCat	5	5	executed
CarmeQ	4	4	-
Cap1	3	3	mixed
Cap2a	5	4	-
Cap2b	12	6	-

higher layer of abstraction (i.e. the parts that can be implemented using S&A models are separated from the parts that have to be implemented in ABAP). In the SIW the development of a service is separated from holding the corresponding web service description consistent in the enterprise service repository (ESR). Thus, service description is separated from the code. BRF separates the rule definition from the process definition. Further, in VCat the design specifications are formulated using a different language. Finally, behavioral and structural parts of the specification are separated in the “*construction models*” in case study Cap1.

Four of the case studies explicitly specify separation of artifacts. In Oberon the basic UI model is separated from change transactions that define changes on the UI model. In BW the info source is separated from the transformation description for the data, which is further separated from the layout definition. Here the separated artifacts reference each other. In VCat the configuration extension is a separated artifact that includes reused aspects. Finally, in case study Cap1 the “*UI-Specification*”, which specified user interface aspects is separated from the specification. Similarly, in VC and Oberon the user interface models are separated from the functionality which is during execution accessed using the service concept.

Discussion of Compliance with Results from Related Studies

In the last years several studies on MDE were performed. In this section it is discussed how the results of the study presented here fit into the findings of these other studies. In [99] Hutchinson et al. present insights into practically applied MDE collected in a large empirical assessment. Among other things they subsume that “a lot of MDE success is hidden”, since often automation and modeling is used in a pragmatic way (“much of MDE in industry is the kind of modeling and/or automation that represents pragmatism in the face of an otherwise tedious or intractable problem.”). The results of this study lead to a similar impression. Only one of the captured objects of study follows an approach that can be compared to MDA [154]: the case study Cap1. Even so, most approaches work with far more abstract views of the system than plain code or gain important benefits based on automation (e.g. as in case study VCat).

5. Study Results: MDE in Practice

In their review paper [143], Mohaghegi et al. subsume that “Combining MDE with domain-specific approaches and in-house developed tools has played a key role in successful adoption of the approach in several cases.” The results presented in this section support this impression. Most objects of this study, which are successfully used in practice, combine models and domain specific languages and work with in-house developed automated activities.

In addition, Mohaghegi et al. [143] reveal two lacks in the reviewed reports. First, they state that portability to multiple platforms has often not been feasible. Here the case studies include an example that clearly allows portability: The user interfaces models within Oberon can be interpreted using different interpreters, which allows porting the application to desktop PCs as well as mobile devices. The second lack identified in [143] is the lack of examples for executable models. Their development is in [143] discussed to be still a challenge. However, the tool Oberon in case study Oberon is also a positive example for executable models. A second example, that this study revealed, is the use of S&A models in case study BO, which are interpreted, too. Thus, it seems that portability and execution of models is possible for use cases with a very clear focus on a specific domain.

In [145], Mohagheghi et al. state that “There is no tool chain at the moment and companies must integrate several tools and perform adaptation themselves.” This fits to the impressions of this study, too. In none of the 11 case studies an external tool chain was adopted. BW might be seen as a tool chain that is provided by SAP and adopted by the customers. However, the interviewee reported, that often only a part of the BW setting is adopted.

In [135], a situation is reported, where model are used fragmented without a “controlled sequence of models and transformations in the process”. In contrast, the case studies in this study come along with a sequence of activities, which is applied similarly in several projects. This different result might be explained with the different domain focused by the study presented here.

Finally, Weigert et al. [211] report that “MDE encompasses all phases of the software-development life cycle”. The captured case studies support this result. For example, personalization of software after deployment is supported by the case study Oberon. Also the MDE setting in Carmeq is applied since several releases.

Threats to Validity

In the following, threats to conclusion validity are discussed for the presented data (for the discussion of other threats to validity see Section 4.4). The data set that was used in this section is with 11 MDE settings quite small. From the viewpoint of statistical evaluations, a greater number of case studies on MDE settings would be necessary to draw conclusions on the usual structure of MDE settings in practice. However, the data is sufficient to provide a first impression of the possible complexity of combinations of different languages or the used number of automation steps within an MDE setting.

5.1.2. Observations on MDE Settings

The qualitative analysis of the records of the first study (i.e. the six SAP case studies) revealed four observations that are worth mentioning here.

Observations on Reuse

Two of the observations concern the reuse of MDE settings and activities within an MDE setting.

OM1 An MDE setting is reused over multiple projects.

Although MDE settings change over time, they are often reused. This holds also for company specific settings. For example, the MDE setting *BO* was long time a standard procedure and used in around 100 projects, while the *SIW* was built to build services for approximately three hundred BOR/Bapi functionalities. None of the case studies was used only once or even only a hand full of times (note that the use numbers given for *VC* are only the uses attended by the consulting company FIVE1 GmbH & Co. KG).

OM2 Single MDE activities are partly used in multiple phases of the software development life cycle.

Going a step further than [211], the observation is made here that several MDE activities are used in multiple phases of the software development life cycle. For example, models of the user interface design in *Oberon* are partly already used during design phase, before the actual implementation starts. Another example is *BW*, where the interpretation of the reporting (i.e. the generation of HTML) is performed during runtime. Also change transactions in *Oberon* can be defined and added to the system at runtime.

To evaluate this result six of the interviewees were asked whether they agree with the statement “Several activities (such as changing a model or generating an artifact) are used in multiple phases of the software development life cycle.” Five of the interview partners agreed. The other one disagreed, whereas this answer was not given generally, but with the specific case study in mind.

Observations on Understanding of the Concept “Language”

During the interviews of the first study it turned out that it is difficult to count the number of used languages. The qualitative analysis of the records revealed two observations on the reasons for that difficulty.

OM3 *It is not in each case easy to determine whether something is a language.*

For example, there are languages that have no explicit meta model, such as the rule set definition language of BRF. One of the interview partners get to the heart of it by asking whether something that is only supported by a single tool is a language. Another example is a setting where specific artifacts are called “models” explicitly, but interview partner would not call the formal definition of the artifacts structure a language definition. In consequence, some languages have no explicit name (e.g. the user interface models in Oberon, the internal meta models in Cap2a, and the error messages in Cap2b have no official name).

OM4 *It is not easy to determine what category a language is of (i.e. general purpose language (GPL), domain specific language (DSL), textual or graphical modeling language).*

During the first study the interviewees were asked to rate the language categories for the languages or artifact formats that are used. For example, one of the interview partners rated Excel as GPL, while another did not. One answer just explicitly excluded DSL. Another language was described as a textual modeling language which is graphically modeled. WSDL (web service description language) was not rated as belonging to one of the above mentioned categories (“WSDL defines messages, not their content (objects)” (transcription translated from German)). ABAP was rated as programming language or GPL. In one case the answer was “neither of them [GPL, DSL, or modeling language], programming language; most likely GPL” (transcription translated from German). This illustrates that the usually used terms are not sufficient to express differences that developers experience. Often clear definitions seem to be missing. Finally, HTML was barely rated as DSL (“can be called a DSL (reluctantly)” (transcription translated from German)) by an interview partner, who communicated that he rather would call HTML a rendering technology.

Threats to Validity

In the following, threats to conclusion validity are discussed for the presented observations (for the discussion of other threats to validity see Section 4.4). The four observations are a result of a qualitative examination of 6 SAP case studies. To generalize these observations for more than one company, it would be desirable to substantiate the data with observations from other companies with different sizes. Nonetheless, the observations are included here since they already provide interesting insights that can support researchers when preparing for future studies on MDE settings.

5.2. Nature of Structural Evolution

In this section, data from the first and second study is used to evaluate the hypotheses about structural evolution $H_{existence}$ and H_{common} . Subsequently, data about the evolution histories from the third study is used to gain further insights into motivations behind structural evolution.

5.2.1. Existence and Relevance

In Section 3.5.3 the hypotheses $H_{existence}$ and H_{common} are introduced. To evaluate these hypotheses (i.e. to evaluate whether structural evolution and substantial structural evolution exist and are common in practice) despite the fact that data on evolution is rare, the concept of triangulation (as described in [177]) is applied.

The data from two independent sources, each with its own advantages and disadvantages, are combined. The first data source are data records from the first study (see Section 4.1), which was performed with the focus on capturing the structure of MDE settings in practice. The observed cases were not chosen with the topic of evolution in mind, which reduces the selection bias. However, the disadvantage of this first data set is that all case studies stem from a single company and that all data was collected by the same interviewer. As second data source reports from literature were collected in the second study (see Section 4.2). Although a selection bias cannot be excluded for literature studies, the advantage of this data source is that it provides a broader spectrum of companies and domains and that the reports were captured by different research teams. Thus, the second data source does not suffer from the problems of the first data source and vice versa.

Data

Following, an overview about the changes identified for MDE settings from the SAP case studies and the literature reports is given.

SAP Case Studies For the case study VC, no hints about evolution could be found in the records. In contrast, for the case studies SIW, Oberon, and BRF, hints on evolution were identified in the records. Such hints are often part of descriptions of the improvements reached by the introduction of the current setting. For example, one record from case study Oberon included the statement that the development functionality that is now provided by one tool was split between several tools before. From this record it can be concluded that the number of tools changed ($C7$).

For the case studies BO and BW, more precise information is available. On the one hand, the evolution history of the case study BO was covered in detail in context of the third study (as described in Section 4.3) afterwards. On the other hand, the records for case study BW included a more detailed description of a former version of this MDE setting. Therefore, the difference to this former MDE setting is used to derive information about the evolution that happened. In these two cases, additional information about non-structural evolution could be captured.

The change types for which hints could be identified are summarized in Table 5.2. It is illustrated whether there is a more precise documentation of the changes or whether there are hints on a change type, only.

Literature Reports In the following, the change types that can be identified for the different case studies from literature reports are presented and summarized in Table 5.4.

The report of the case study CsFO ([182]) ends with a note that better integration of different tools ($C7$) and more automation of the construction phase ($C8$) are planned in future.

The report for case study CsFBL ([108]) describes several changes that happened to the tool support for the language FBL. The tool vendor started with providing the visual programming language FBL together with an editor and a code generator. The user interface of the editor was later on extended ($C3$). Over time they introduced the tool function test to enable developers to debug FBL ($C7$). The introduction of automated verification or debugging operations changes the order of manual and automated tasks. As a result manual programming is followed by automated debugging and further manual correction before the automated generation is applied ($C8, C9$). Further they report on the introduction of templates to allow programming on a higher level of abstraction. Developers have to choose and configure templates by specifying parameters. A chosen template with parameters is then automatically translated to FBL and from there code is generated. Thus, language (templates instead of FBL) and generation implementation (transformation plus generation) changed ($C1, C2$).

Fleurey et al. [72] present an approach to adapt a migration process towards the specific needs of the current application. A case study of the actual application of this migration process to migrate a banking application is also presented (CsBA). Interestingly, the changes that are actually applied

Table 5.2.: Identified change types for SAP cases studies (\circ = hints on changes; \bullet = documented changes)

	SAP Case Studies					
	SIW	Oberon	BO	BW	BRF	VC
<i>Changes in general</i>	\circ	\circ	\bullet	\bullet	\circ	
<i>Non-Structural Changes</i>			\bullet	\bullet		
<i>C1</i> exchange automated activity			\bullet	\bullet		
<i>C2</i> exchange language				\bullet		
<i>C3</i> exchange tool					\circ	
<i>Structural Changes</i>	\circ	\circ	\bullet	\bullet	\circ	
<i>C4</i> change number of artifacts			\bullet		\circ	
<i>C5</i> change number of languages			\bullet		\circ	
<i>C6</i> change number of manual activities				\bullet	\bullet	\circ
<i>C7</i> change number of tools	\circ	\circ	\bullet		\circ	
<i>C8</i> change number of automated activities	\circ	\circ	\bullet	\bullet	\circ	
<i>C9</i> change order of manual / automated activities	\circ			\bullet		

in CsBA differ strongly from the proposed changes. For example, it was necessary that the resulting system conforms to the development standards of the customer. Thus, it was not sufficient to produce code, but to provide corresponding models that were synchronized with the code, such that round-trip engineering on the migrated system was possible. Therefore, they replaced the code generation with an automated UML extraction. They integrated the Rational Rose code generator used by the customer to generate code skeletons out of the models (*C7*). Further, they added a generation to migrate the remaining code from the platform-independent model (extracted from the original code) into the code skeletons (*C1, C2, C4, C5*). Conforming to the round trip engineering, some manual migration tasks have to be applied to the models (*C6*). The corresponding reapplication of the Rational Rose code generation adds an additional automated step to the MDE settings (*C8*). Thus, instead of being only followed by manual migration, the automated migration is followed by manual migration activities on the Rational Rose model, a generation of code and further manual migration activities on the code. The order of manual and automated tasks change, as manual migration is intermixed with automated code generation (*C9*).

In [146] a DSML for the generation of configuration files in telecommunication industry is presented (case study CsTe). It is also reported that the DSML was changed later on. The verification language EVL to incrementally check the correctness of the models during development was integrated. This intermixes manual modeling activities with an added automated analysis for correctness (*C8, C9*). Further, the generation of the configuration files was exchanged, by an implementation that is based on a composition system conforming to the approach reported in [104]. Motivation for this change was the wish to be flexible to reach higher levels of abstraction. The number of input models and languages changed from one to a flexible number (*C4, C5*). Also the number of manual modeling activities changes for the developer, who has to create a number of different DSL models (*C6*).

In [15], Baker et al. described how the use of MDE within Motorola changed over time (case study CsMo). This includes reports about changing tools (*C7*) and a changing number of languages and used models (*C4, C5*) with the introduction of Message Sequence Charts (MSC) and SDL. Further, it is reported about changes in MSC (*C2*) that enabled the introduction of automated generation of test cases (*C8*).

5. Study Results: MDE in Practice

Although the report about the printer company (CsPC) in [98] includes hints that the MDE setting under study changed, the information is not sufficient to make assumption about the actual type of change. Similarly, the report about the car company (CsCC) in [98] includes not much detailed information about the actual change. At least this report explicitly includes the information that the used modeling language changed.

Table 5.3.: Identified change types in literature reports (\circ = hints on changes; \bullet = documented changes)

	Meta-Study						
	CsFO [182]	CsFBL [108]	CsBA [72]	CsTe [146]	CsMo [15]	CsPC [98]	CsCC [98]
<i>Changes in general</i>	\circ	\bullet	\bullet	\bullet	\bullet	\circ	\bullet
<i>Non-Structural Changes</i>		\bullet	\bullet		\bullet		\bullet
<i>C1</i> exchange automated activity		\bullet	\bullet	\bullet			
<i>C2</i> exchange language		\bullet	\bullet		\bullet		\bullet
<i>C3</i> exchange tool		\bullet					
<i>Structural Changes</i>	\circ	\bullet	\bullet	\bullet	\bullet		
<i>C4</i> change number of artifacts			\bullet	\bullet	\bullet		
<i>C5</i> change number of languages			\bullet	\bullet	\bullet		
<i>C6</i> change number of manual activities			\bullet	\bullet			
<i>C7</i> change number of tools	\circ	\bullet	\bullet		\bullet		
<i>C8</i> change number of automated activities	\circ	\bullet	\bullet	\bullet	\bullet		
<i>C9</i> change order of manual / automated activities		\bullet	\bullet	\bullet			

Summary on Hypotheses

As summarized in Tables 5.2 and 5.4, all types of structural evolution that were identified in Section 3.4 actually occur in practice. This validates the first hypothesis $H_{existence}$.

The evaluation of the second hypothesis H_{common} requires a statistical test. In each of the two data sets 5 cases with structural evolution have been identified and in each of these cases also hints for substantial structural have been found. As defined in hypothesis H_{common} , the phenomenon of structural evolution is considered to be common if it exists in more than 25% of all MDE settings.

The null hypothesis h_{0_all} is that the percentage of MDE settings that are subject to structural evolution in practice is below or equal to 40%. Correspondingly, the alternative hypothesis h_{1_all} is that the percentage of MDE settings that are subject to structural evolution in practice exceeds 40%. The tested percentage of 40% substitutes the required 25% here, in the interest to approach the actual percentage. When h_{0_all} can be rejected this is an even stronger confirmation of H_{common} .

To enable a comparison, the probable percentage is also tested for the two single data sets. Since both data sets are very small, only the minimal percentage of 25% is tested. The corresponding null hypotheses are as follows: $h_{0_smallSet}$ is that the percentage of MDE settings that are subject to structural evolution in practice is below or equal to 25%. The corresponding alternative hypothesis $h_{1_smallSet}$ is that the percentage of MDE settings that are subject to structural evolution in practice exceeds 25%. The hypothesis $h_{0_smallSet}$ will be tested separately for both data sets.

For the test of the hypotheses the binomial test is used as a test that can be applied to small samples, in order to test for two categories whether data points that conform to one of the two categories occur to a certain percentage within the data set. The two studies categories are MDE setting that underlie structural evolution vs. MDE setting that do not. For the test a significance level of 5% is chosen.

In Table 5.4 the results of the test results are summarized. The hypothesis h_{0_all} can be rejected. The probability that 10 MDE setting that underlie structural evolution are identified for a sample size of 13, while the percentage of such MDE settings is below or equal to 40%, is less than 5% (p-value = 0.007793). Thus, h_{0_all} is not plausible.

For both single case studies the hypothesis $h_{0_smallSet}$ can be rejected, too. Here, the probability that 5 MDE setting that underlie structural evolution are identified for a sample size of 6 or 7, respectively, while the percentage of such MDE settings is below or equal to 25%, is in both cases less than 5% (p-values = 0.004639 and 0.01288). Thus, $h_{0_smallSet}$ is not plausible. This adds to the validity, since it shows that the commonness of structural evolution is not specific for one of the data sets.

The same test was applied to identify the probable percentage of the substantial structural evolution step $C9$, which occurred in 5 of the 13 MDE settings. Therefore, the used null hypotheses are: h_0 that the percentage of MDE settings that are subject to structural evolution in practice is below or equal to 15%. The corresponding alternative hypothesis h_1 is that the percentage of MDE settings that are subject to structural evolution in practice exceeds 15%.

Here the test leads to the rejection of the null hypothesis, too. Thus, change type $C9$ is with more than 15% percentage of the MDE settings in practice not seldom. Again the application of the test to the single data sets, supports the assumption that the effect is not specific to one of them.

Table 5.4.: Results of binomial test to check whether probabilities for structural changes (and change $C9$) exceed 40% (or 15% respectively)

Date	# occurrences	sample size	h_0	h_1	p-value	95% confidence interval	Result
SAP	5	6	$p \leq 25\%$	$p > 25\%$	0.004639	0.4182 - 1	$h_{0_smallSet}$ is rejected
Meta study	5	7	$p \leq 25\%$	$p > 25\%$	0.01288	0.3413 - 1	$h_{0_smallSet}$ is rejected
All	10	13	$p \leq 40\%$	$p > 40\%$	0.007793	0.5053 - 1	h_{0_all} is rejected
SAP $C9$	2	6	$p \leq 5\%$	$p > 5\%$	0.03277	0.0628 - 1	h_0 is rejected
Meta study $C9$	3	7	$p \leq 5\%$	$p > 5\%$	0.003757	0.1288 - 1	h_0 is rejected
All $C9$	5	13	$p \leq 15\%$	$p > 15\%$	0.03416	0.1657 - 1	h_0 is rejected

Subsuming, the results of examination of both data sources supports the hypothesis that structural evolution is common in practice (H_{common}). Table 5.5 summarizes the hypotheses.

Threats to Validity

In the following, threats to conclusion validity are discussed for the presented evaluation of hypotheses $H_{existence}$ and H_{common} (for the discussion of other threats to validity see Section 4.4). The two data sources that are used in this section to evaluate the hypotheses $H_{existence}$ and H_{common} provide us with information about 13 MDE settings, only. Of course a bigger number of cases would allow making more accurate statements how often structural changes occur in practice. However, as described above, the use of two different data sources minimizes selection bias as well as biases due to corporate culture. In fact, the observed frequencies of structural evolution are comparable for both data sets. Thus, although

Table 5.5.: Summary of hypotheses on structural evolution

Hypotheses	SAP records	Meta study	All
$H_{existence}$ Structural evolution and substantial structural evolution occur in practice.	✓	✓	✓
H_{common} Structural evolution and substantial structural evolution are common in practice (i.e. it happens for more than 25% of the MDE settings).	> 25%	> 25%	> 40% ✓

larger scaled empirical studies may help to get more accurate information in future, the data used here is sufficient to determine whether structural changes are sufficiently common in practice to be a relevant object for further research.

5.2.2. Observations on Structural Evolution

The investigation described in Section 5.2.1 revealed that structural and even substantial structural evolution occurs and is common in practice. However, there is still not much knowledge about motivations and triggers behind this structural evolution.

Data on Evolution

To address these issues, the third study (presented in Section 4.3) was performed.

In context of this third study seven evolution histories were captured in cooperation with SAP AG, Ableton AG, Capgemini, Carmeq GmbH, and VCat Consulting GmbH. All in all, the seven case studies span 33 evolution steps. The observed structural changes are summarized in the Tables 5.6, 5.7, and 5.8. Details to all evolution histories can be found in Appendix B. In the following an overview of the collected evolution steps is given.

The case study BO was already subject to seven substantial structural evolution steps in a period of around six years. The MDE setting started as almost classical code centric approach with some activities to ensure tracing between code and data model. Later on a modeling tool was introduced followed by further tools that supported partial generation of the code and eventually, one of these generation tools was adopted company wide as standard (evolution step $S1$). To improve quality of behavioral implementation an interpretable modeling language was introduced in addition to the code (evolution step $S2$). The introduction of an additional modeling tool was motivated by the aim to introduce further quality assurance (evolution step $S3$). A next improvement was reached by introducing a semi-automated support for data migration between the modeling tools (evolution step $S4$). In order to reduce the number of tools the three modeling tools were integrated into a single new modeling tool (evolution step $S5$). A variant of the MDE setting was created to enable simple use at the cost of a reduced set of supported features (evolution step $S6$). Finally, the generation functionality was moved to this new modeling tool (evolution step $S7$). Details on this evolution and the underlying decisions can be found in Appendix B.1.

For the case study Ableton, one substantial structural evolution step was captured. In this case study a fully automated generation process was subject to refactoring (evolution step $S1$). As a side effect of this refactoring parts of the automated quality assurance are separated from the generation process. Details on the trade-offs behind this decision can be found in Appendix B.2.

For the case study Cap1 seven substantial structural evolution steps from a period of around four years are captured. The project started with a standard code generator, which was soon substituted by a project specific generator. In addition, a semi-automated support for the export of the data between two modeling tools was added (evolution step $S1$). To improve merge of the old version of the model and the version that is result of the semi-automated export, a first version of a diff-tool was introduced (evolution step $S2$). Due to changes the MDE setting of the customer the support for the export as well as the diff-tool were taken out of operation (evolution step $S3$). Further the automation of the

Table 5.6.: Identified structural changes in evolution steps of the case studies BO, Ableton, and VCat
 (• = documented change)

	SAP (BO)							Ableton	VCat	
	S1	S2	S3	S4	S5	S6	S7	S1	S1	S2
<i>Structural Changes</i>										
C4 change number of artifacts	•	•			•	•	•	•	•	•
C5 change number of languages	•	•	•		•	•	•	•		
C6 change number of manual activities	•	•	•	•	•	•	•		•	
C7 change number of tools	•	•	•	•	•	•	•		•	•
C8 change number of automated activities	•	•	•	•	•	•	•	•	•	•
C9 change order of manual / automated activities	•		•		•	•		•	•	

export between the main modeling tool and the code generator was improved (evolution step *S4*). Later on automated support for the implementation of a user interface based on mock-ups was introduced (evolution step *S5*). A change in the development process led to a situation that modified versions of the model are created by different teams and need to be merged. To support this merge a second version of the diff-tool was reintroduced (evolution step *S6*). Finally, to address a new need on additional documentation an additional generator was introduced (evolution step *S7*).

For the case study Cap2a five evolution steps from a period of around three years are captured. Three of the five evolution steps are structural evolution steps (and two of them are substantial structural evolution steps). The project started with a first generator that was substituted later on by a more flexible version (evolution step *S1*). This substitution was planned from the beginning and was motivated by the need to rapidly provide a working MDE setting to the customer. The underlying meta model was permanently changed over the time. To create new output artifacts the generator implementation was extended (evolution step *S2*). Later on a part of the generator was reimplemented, such that independence of the formerly used implementation technology is reached (evolution step *S3*). Finally, checks have been optimized over time, such that they can be applied automatically and regular (evolution steps *S4* and *S5*).

For the case study Cap2a six evolution steps from a period of around five years are captured. Five of the six evolution steps are substantial structural evolution steps. The first version of the MDE setting included a generator for the creation of the documentation. To improve the usability, the generator was integrated to a modeling tool and adapted the meta model that was already used in case study Cap2a (evolution step *S1*). In order to support creation of additional resulting documents three further generation activities were embedded into the existing generator over the time (evolution steps *S2*, *S4*, and *S6*). Addressing quality assurance, basic consistency checks for the models were introduced (evolution step *S3*) and the introduction of further consistency checks (following the example of case study Cap2a) is planned. Finally, the output format had to be changed at least one time (evolution step *S5*).

For the case study VCat two substantial structural evolution steps from a period of around seven years are captured. The first version of the MDE setting included a manual task to copy and clean TYPO3 instances from old projects, such that configurations could be reused. To improve this process and decrease the probability that content from old projects is preserved in the copied TYPO3 instance without notice, an automated instantiation and configuration of new TYPO3 instances was introduced (evolution step *S1*). Further the use of open source TYPO3 extensions should be improved in future. For several reasons, extensions need to be adapted before they are used. To support reuse of these

Table 5.7.: Identified structural changes in evolution steps of the case studies Carmeq and Cap1 (• = documented change)

	CarmeQ					Cap1					
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5	S6
<i>Structural Changes</i>											
C4 change number of artifacts	•		•			•	•		•	•	
C5 change number of languages	•		•			•	•		•		
C6 change number of manual activities		•						•	•		
C7 change number of tools	•	•	•	•	•	•	•	•	•	•	•
C8 change number of automated activities	•	•	•			•	•	•	•	•	•
C9 change order of manual / automated activities	•	•	•			•	•	•	•	•	•

adaption among the different projects at VCat, there are concrete plans to introduce an internal extension repository within VCat (evolution step S2).

For the case study Carmeq five structural evolution steps from a period of around eight years are captured. Three of the five evolution steps are substantial structural evolution steps. Initially, the AUTOSAR documentation was created without explicit modeling. To deal with inconsistencies between documents, a central model of the standardized software as well as an UML profile for AUTOSAR were introduced (evolution step S1). An automated generation of figures and tables on the basis of the central model was introduced. Later on macros were implemented to support the integration of figures and tables into the standard documents (evolution step S2). To support quality assurance for the generated figures and tables the modelers started to use diff-tools for comparison between old and new versions of the artifacts (evolution step S3). Further, a CI server was introduced, such that the generator is executed centrally (evolution step S4). Finally, an alternative implementation for some parts of the generator (e.g. the automated import between two of the modeling tools) was introduced (evolution step S5).

Combination of Changes

Based on the data described above, some observations on the occurrence and combination of evolution steps can be made.

O01 Structural evolution steps are not necessarily exceptions, but can occur in sequence several times

In four of the seven case studies from the third study (Section 4.3) five or more evolution steps could be captured. For example, the case studies BO and Cap1 were subject to seven structural changes, each. Only for one case study (Ableton) a single structural evolution step was captured.

O02 Substantial structural changes occur in most of the observed structural evolution steps.

In each of the captured case studies substantial structural evolution steps (including change type C9) occurred. Substantial structural changes occurred in 27 of the 30 (90%) structural evolution steps that were captured in the third study. The change type C9 occurred in 18 of the 30 (60%) structural evolution steps (i.e. in two thirds of the substantial structural evolution steps).

Table 5.8.: Identified structural changes in evolution steps of the case studies Cap2a and Cap2b (• = documented change)

	Cap2a					Cap2b					
	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5	S6
<i>Non-Structural Changes</i>											
C1 exchange automated activity	•	•	•	•						•	
C2 exchange language										•	
C3 exchange tool											
<i>Structural Changes</i>											
C4 change number of artifacts	•				•	•	•	•	•	•	
C5 change number of languages	•	•			•	•	•	•	•	•	
C6 change number of manual activities	•					•				•	
C7 change number of tools	•					•					
C8 change number of automated activities					•		•	•	•	•	
C9 change order of manual / automated activities					•			•			

O03 Structural evolution steps are most often combinations of multiple different structural changes.

Only two of the 30 observed structural evolution steps contained only one type of structural change. In contrast, 26 of the structural evolution steps consisted of 3 or more types of structural changes. Amongst these 26 structural evolution steps 4 included all 6 types of structural changes. The different changes often do not only coexist but reflect the same change or even cause each other.

O04 A minor observation in that context is that only seldom a change in the order of manual and automated activities (C9) is caused by improving an existing automated activity such that a manual activity is no longer necessary (C6).

This happened in the fourth evolution step in case study Cap1. However, in most cases C9 was caused by the introduction of additional automated activities (C8).

Trade-Offs

Based on the records of the third study, following observation on evolution decisions could be made:

O05 Structural changes are often trade-offs, e.g. with respect to costs and manageability.

For example, implementing a smaller new generation step is easier to manage than applying a change to an existing automated activity. A further factor in such a trade-off is the weight that is given to the different productivity dimensions. An example is the evolution step S3 in case study BO. In favor of better consistency and quality assurance, it was taken into account that links, between model and implementation, are no longer maintained during development.

Another example is the case study Ableton, where the initial MDE setting was successfully in use. Nonetheless, organizational change led to the need to adapt the setting in a way that understandability

and flexibility was improved. For reaching this overall improvement a small reduction in the *degree of automation* was taken into account.

Part of such trade-offs are also decisions against changes of the MDE setting. For example, in case study Ableton the introduction of automated support for the migration of data during the regular changes of the DSL was not done, since it was not rated as profitable.

In other cases, such trade-offs lead to a delay of MDE evolution, only. For example, the third evolution step in case study Cap2a included a complete new development of a generator on the basis of a new technology. The corresponding idea existed for a while and was only implemented when a number of other big changes to the generator became necessary.

As observation O04 suggests, in many of the observed cases it was decided to increase the *degree of automation* or tool support by adding new automated activities instead of adapting existing automated activities like transformation steps. For example, in case study Carmeq additional importers were added, instead of adapting the existing importer. Thus, a substantial structural change that has the potential to cause drawbacks for the changeability is accepted in favor of costs and manageability of the structural evolution step.

Changing weights: The factors involved in such trade-offs change over time. Costs that can be invested in an evolution step can change strongly. For example, the evolution step *S4* in case study Cap1 was implemented by a developer in his leisure time. The weight that is given to different productivity dimensions can also change. For example, evolution step *S1* in case study BO was mainly driven by the desire to increase the *degree of automation*. For a long time explicit conceptual modeling had a priority. Later on the priorities change, such that efficiency and total cost of ownership became more important. As a consequence, evolution step *S5* led to reduction of the number of tools and inconsistencies at the cost of a loss of graphical modeling capabilities, the loss of functionality to simulate status models, as well as loss of the ability to model design alternatives.

Software Requirements

A couple of evolution steps are motivated by the need to address new requirements for the software under construction (i.e. for the software that is usually built with an MDE setting).

For example, several evolution steps are motivated by the need for better quality.

O06 Automated checks on models to improve quality were introduced at different points in time.

Only in the case studies Cap1 and Ableton checks were part of the MDE settings from begin on. In the other five case studies checks were introduced later on. For example, in case study BO three evolution steps are motivated by quality issues. These evolution steps are S2, where the introduction of S&AM enabled automated checks of behavioral constraints, S3, which enabled additional validation activities and eliminated some potential for inconsistencies, and S4, which enabled cross checking between models in ARIS ad the service repository. In the first evolution step in case study Carmeq the machine readable version of models was introduced to tackle inconsistencies, too. Similarly, the introduction of the installation script in the first evolution step in case study VCat was motivated by the need to reduce unexpected preservation of content from former projects. Finally, the evolution steps S4 and S5 of case study Cap2a as well as the evolution steps S3 and S7 of case study Cap2b introduced additional checks into the respective MDE settings.

O07 Sometimes requirements on additional results of the MDE setting are motivated by the possibility to reuse models that are already part of an MDE setting for generation of additional artifacts, such as code or documentations.

An example for such a change is the evolution step S5 in the case study Carmeq. Here an additional importer was implemented to extract more of the existing information from the EA models. This enabled the generation of BSW service interfaces. Similar examples can be found in the case studies Cap1 (evolution step S7 introduced the automated generation of specifications), Cap2a (evolution step S2 introduced the generation of an HTML catalog), and Cap2b, where even three such evolution steps can be found (S2, S4, and S6).

Simplification of options: Besides the two mentioned observations on new requirements on the software under construction, a single case could be identified, where the motivation for an evolution step

was a simplification of options for the implementation of the software under construction. In evolution step S6 of case study BO this reduction is used to create a simplified version of the MDE setting, such that a new target group of developers is enabled to create business objects. Thus, this evolution step leads to the introduction of a setting that can be compared to a DSL.

Environment

Sometimes structural evolution is caused by changes in the environment of the MDE setting.

O08 *A first observation in that context is that changes in an MDE setting can be driven by the need to take other MDE settings into account.*

Changes in an MDE setting that is used in a cooperating project or company can lead to new opportunities for the integration of both MDE settings. For example, in Cap1 the customer's MDE settings was changed, such that it based on the same framework (Enterprise Architect) as the MDE setting of Capgemini, afterwards. As a consequence the automation and support for merging and export could be removed in evolution step S3. A similar mechanism worked, when the MDE setting of the customer changed, such that mock-up were modeled in the Enterprise Architect, too. As a consequence evolution step S5 was enabled, where a partial generation of user interfaces on the basis of the mock-up model was introduced.

Another example can be found in case study Cap2b. Here, functionality to generate an HTML catalog was introduced in evolution step S2 of Cap2a. After that, an HTML generator was introduced in evolution step S4 of case study Cap2b. The resulting HTML catalog is an extension of the HTML catalog that can be generated with Cap2a for a same project. Thus, both MDE settings can be combined to create a common result.

Interestingly the case study CsBA [72] from the second study is an example for a similar interconnection of two MDE settings. Here the MDE setting for the platform migration needed to be adapted, such that the MDE setting that is used to develop the migrated system further can be applied later on.

O09 *An additional mechanism that can be observed is that changes of MDE settings are sometimes inspired by other MDE settings.*

For example, the first evolution step in Cap2b adopted the meta models and the use of the tool Power Designer as basis for generation from Cap2a. Further, the introduction of consistency checks in evolution step S4 from case study Cap2a is reproduced in evolution step S7 in case study Cap2b. Two further examples can be found in case study Carmeq. The introduction of macros for the integration of figures and tables in evolution step S2 was inspired by already available macros that were used for layout reasons. Further, the approach to create an additional importer (instead of adapting an existing one) during evolution step S5 was already applied in an MDE setting that is used for the generation of figures from the AUTOSAR meta model.

Organizational change: Not only other MDE settings, but also the structure on teams can motivate evolution of an MDE setting. For example, the introduction of a diff-tool to support merges of different versions of a model (evolution step S6) in case study Cap1 was motivated by a change in the way how the Capgemini team cooperated with the customer team. Before the organizational change a central model was manipulated by the Capgemini team, only. Afterwards both teams got access to the model and applied changes.

Interestingly, a similar situation (different teams manipulate a model and the versions cannot be merged automatically without conflict), was approached in case study Carmeq differently. Here not the MDE setting was changed, but the organizational structure. As a consequence only a restricted set of people (one team) is allowed to manipulate the model. Thus, the same mismatch between development process and MDE setting was once solved by adapting the MDE setting and once solve by adapting the development process.

Pragmatic Developer's Decision

Following observation on the motivation behind structural evolution steps concerns the persons who trigger the evolution:

O10 *Some evolution steps are not planned centrally, but are caused by developers who add automation steps to ease their daily work.*

Examples, where developers evolved the MDE setting that they used themselves, can be found in the four of the case studies. In the first evolution step in BO similar automations of the same implementation aspects were introduced independently by developer teams that worked on different projects. Eventually, one of the automations was chosen as standard. In case study Cap1 even four evolution steps (S2, S4, S6, and S7) are triggered this way. As already mentioned above, the introduction of the model distillation tool to automatically correct typical errors in an output format (S4) was even performed by a developer in his leisure time. Finally, in case study Carmeq the macros and diff-tools (S2 and S3) were introduced by developers and in case study VCat the planned introduction of the new repository for reuse was triggered by the developers (and finds supported by the management).

Apart from evolution that is triggered and implemented by developers, developers might also propose changes to the MDE setting that are then implemented by the tool vendor. For example, in Cap2b the use of HTML generation to gain a new possibility for navigating through the model was an idea of developers that use the MDE setting.

There are also introductions of automation steps that are not triggered by developers. For example, the first evolution step in case study VCat and the first and fifth evolution step in the case study Cap1 were requested by the management or customers, respectively. However, it seems that developers have relevant insights into potentials for further automation of development. More important, it seems that developers even have an own interest in automating parts of the development.

Manage the MDE Setting

Finally, structural evolution might be motivated by the need to manage the MDE setting:

O11 *The motivation of some evolution steps is to reduce the complexity of MDE settings, which can be considered as “refactoring”.*

An example of this is the introduction of the new repository (MDRS) in business object development (evolution step *S5* in case study BO). This refactoring was completed in evolution step *S6*, were also the status and action management was integrated into the MDRS.

In case study Cap1 a less obvious clean-up occurs. Here the evolution steps *S3* and *S5* succeed changes in the MDE setting of the customer, which can be rated as clean-up. In both cases development activities that are performed by the customer were moved to the development tool that is used at Capgemini. Consequently a better integration of both MDE settings was reached.

A Note on Preparation Evolution: While clean-up evolution reintegrates MDE settings that are already complex, it is possible to find rare examples for evolution steps that prepare an MDE setting, such that following evolution steps can be applied more easily or even without structural changes. For example, in case study Ableton the build process is split up as a preparation for future evolution (i.e. the goal to move process to central build server). Also in the Telefónica case study in [146] (from the second study) the fixed DSML was substituted by a composition system for DSLs to gain better flexibility for changing the combination of DSLs used during development.

Discussion

Subsuming, the data from the third study could be used to make eleven observations on how structural evolution steps occur, on trade-offs that are involved and on motivations behind the structural evolution steps. Combining these observations to a coherent picture of structural evolution, two rules of thumb can be formulated.

Rule of thumb on trade-offs In Section 3.4 it was discussed that structural changes can affect multiple productivity dimensions (e.g. the *degree of abstraction*, the *complexity*, or *changeability*). This makes it difficult to change an MDE setting by only causing positive effects, especially when the budget is restricted. Considering that structural evolution steps are often trade-offs (O05), it can be expected that structural evolution steps, although leading to an overall improvement, will often also lead to drawbacks for less prioritized productivity dimensions.

Karaila [108] already identified a connection between Lehman's laws [124] of software evolution and the evolution of the DSL sold by Metso Automation. In the context of continuously changing environments and requirements, structural evolution, which can be motivated by these factors (see O06 to O09), has to be expected, too. This fits to the observation that structural evolution steps occur frequently on single MDE settings (O01). Combining these observations, the following rule of thumb can be formulated:

Even if an MDE settings is already successfully used, structural evolution steps, in particular substantial evolution steps, may occur that lead to drawbacks for productivity dimensions that are rated with less weight.

Evolution does not equal a steady improvement towards a fixed point. There will always be reasons for changes, such as changing technologies or new demands on the products. Even if the development is based on a well chosen DSML with complete code generation, a (non-structural) language evolution to fit the new demand might be very costly and thus not always possible [61]. For that reason, engineers might decide to apply structural changes even if there are drawbacks. In practice, a first workaround might be the decision to manually adapt generated code.

An example for this rule of thumb is the case study Ableton, which is described in detail in Appendix B.2. There, the initial setting was adapted with small drawbacks for the degree of automation, although it was successfully in use, before.

Rule of thumb on evolution and introduction of MDE The rule of thumb on trade-offs was formulated on the bases of the observation that structural changes are trade-off solutions that take weights that are associated to different productivity dimensions into account (O05). Considering that structural changes, especially substantial structural changes, have bigger impacts than non-structural changes and therefore can be used for significant improvements, following second rule of thumb can be formulated:

In some cases the introduction of MDE can be characterized as a result of a sequence of structural evolution steps, in particular substantial structural evolution steps.

The idea that MDE is introduced in multiple stages is not new. However, proposals for a stepwise introduction of MDE or MDA have often the underlying assumption that all involved evolution steps directly contribute to the introduction of MDE. The data presented here raises the question whether this straight-forwardness is always present under changing weights and trade-offs.

There are arguments for an introduction of MDE by evolution. First, there is the potential for better developer acceptance, especially when developers propose the evolution steps or participate in their implementation. Further, a stepwise introduction gives developers the chance to grow with the MDE settings instead of being confronted with completely new requirements on their skills [15]. As Selic noticed in [179], the previous investments have to be taken into account, as well as the fact that major assets lie in legacy code, tools, and libraries. This is also the reason why the initially used DSML in the Telefónica case study in [146] was the Common Information Model, which laid the foundation for the more abstract DSMLs in later evolution steps.

An example for an MDE setting that evolved from a rather code centric approach to a model-driven approach is case study BO, which is described in detail in Appendix B.1.

Threats to Validity

In the following, threats to conclusion validity are discussed for the presented observations on structural evolution (for the discussion of other threats to validity see Section 4.4). This, section presents a mainly qualitative examination of 6 evolution histories (including 33 evolution steps). Since these case studies were selected with the aim to study structural evolution, a selection bias towards evolution histories with structural evolution is expected. Further, structural evolution was in focus during elicitation of the evolution steps. Consequently, conclusions on frequencies of structural evolution steps compared to non-structural evolution steps cannot be drawn. Similarly, statements on the number of structural evolution steps that can be expected per year should not be made on the basis of this data.

Fortunately, the data on the 33 evolution steps stems from four different companies. The fact that all observations are based on examples of at least two companies, allows concluding that they are not company specific.

Part III.

Modeling and Analysis

6. Modeling

To enable capturing and analyzing MDE settings, the Software Manufacture Model modeling language is introduced in this chapter.

The Software Manufacture Model language was designed, such that it allows identifying how an MDE setting influences changeability and how MDE traits are manifested. For this design, it was first necessary to identify required concepts. It has to be decided to which level of detail information will be specified within the model. On the one hand, nuances in modeling concepts can turn out to be essential to differentiate between diverse situations. On the other hand, details make the models more complex. This, however, complicates the tasks to create and analyze models and thus decreases the applicability. In Section 6.1 it is described how expressiveness and applicability are balanced in the language design.

The actual process of balancing included the creation of an initial design of the language, which was published in [P6] (referred to as “*initial Software Manufacture Model language*” in the following). For most concepts it was already possible to choose the appropriate level of detail. Basis for such decisions was the question what information is required for the analysis. However, at the time of the initial language design only rare knowledge about actual MDE settings was available. Consequently, it was for some concepts not possible to predict whether details turn out to be relevant to distinguish different approaches from practice. To tackle this uncertainty, the initial language design resulted in a rich expressiveness. This concerned especially language concepts to express the extent to which content is moved between artifacts. Another example is concepts that allow expressing in detail how sets of artifacts are interrelated.

In a next step, this *initial Software Manufacture Model language* was applied to model the six already captured MDE settings from the first study. Further the initial version of the language was used to capture the three case studies Cap1, Cap2a, and Cap2b in context of the third study. As a result 9 models were created that altogether included 193 activities. These 193 activities from practice were examined to identify what language concepts were used and to which degree details were modeled.

On the basis of these data the *Software Manufacture Model language* was designed as a simplified version of the *initial Software Manufacture Model language*. Besides a couple of simplifications that concern the syntax of the language, two main simplifications concern also the expressiveness. First, it turned out that it is not necessary to describe detailed relations between subsets of sets of artifacts. Here it is sufficient to express whether relations hold for all artifacts or only for a subset of artifacts of a set. Further, not all possibilities to model nuances of overlaps in content of two artifacts were used. Consequently, the expressiveness was reduced here, too. A detailed description of the reduction decisions can be found in Appendix C, where also the metamodels of the initial and final language design are compared.

The resulting *Software Manufacture Model language* was applied to again model the MDE settings of the 9 case studies mentioned above. Further, the *Software Manufacture Model language* was used during elicitation of the case studies Carmeq and VCat. The simplified language design proved to be sufficient to capture the different aspects of the case studies.

The design decisions are discussed in Section 6.1. Afterwards, the Software Manufacture Model language is introduced in Section 6.2. The use of patterns can help to capture experience that can be associated with certain structures in the model. As an additional prerequisite for the analysis, the Software Manufacture Model pattern language is introduced in Section 6.3.

This chapter is partially based on [P2] and [P6].

6.1. Language Design Concepts

In the following, design decisions for the Software Manufacture Model language are discussed.

First, it needs to be decided whether behavioral aspects or structural aspects should be captured. Other modeling approaches for the investigation of MDE (e.g. megamodels) most often address the structural aspects, i.e. the question how artifacts are interrelated [P5]. However, information about a specific state of artifact relations is not sufficient for an investigation of changeability support or the

6. Modeling

interrelation to software development processes. For these tasks, it is rather important to know how the artifacts and their relations can be changed when activities are performed. Therefore, the Software Manufacture Model language is designed as behavioral language, more precisely as process language.

In the following, the main concept of the language introduced and discussed.

6.1.1. Object Flow

For an investigation of the *process interrelations* and changeability the complete set of technically possible activity orders needs to be considered. Process languages focus either on control flow or on object flow. In seldom cases a language (e.g. UML activity diagrams [156] or BPMN [157]) can be used to express both – either simultaneously or alternatively. In the following, both possibilities are compared and it is explained, why the language for MDE settings was designed with a focus on object flow.

This decision shall be illustrated on the *EMF example* (Section 2.6).

Example 1

Figure 6.1 shows an object flow of the involved activities as they are described in Section 2.6.

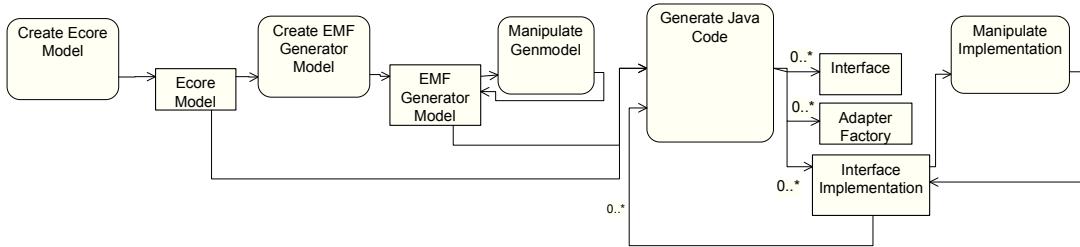


Figure 6.1.: Object flow of *EMF example* (Section 2.6). Notation based on UML activity diagrams [156].

A simple and intuitive control flow for this example is shown in Figure 6.2. Here activities are ordered and the last activity “manipulate implementation” can be performed multiple times. This loop in the control flow reflects the differences in the cardinality of artifacts that are handled in the different activities. While “generate java code” produces multiple “interface implementations”, “manipulate implementation” represents manipulation of a single “interface implementations”, only.

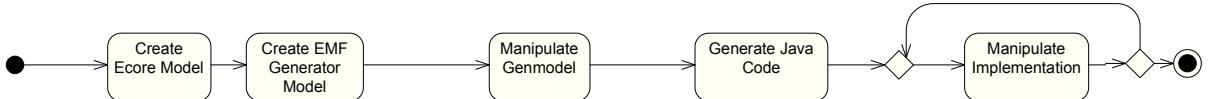


Figure 6.2.: Intuitive simple control flow of *EMF example* (Section 2.6). Notation based on UML activity diagrams [156].

The shown intuitive control flow is easy to perceive. However, it reflects a small subset of all possible orders of the activities, only. Besides differences in the cardinality other reasons for branches and loops in the control flow exist. Figure 6.3 illustrates different control flow edges that have to be added for different reasons. The edge that is cause by the difference in cardinality of the consumed and produced artifacts is illustrated in blue. Another reason is the fact that activities can be optional (referred to as optionality in the following). For example, the activities “manipulate genmodel” and “manipulate implementation” are optional, since they are not necessary to create a running system (e.g. for some use cases it is possible to build or manipulate a system without these activities). The edges that represent this optionality in the control flow are illustrated in brown. A third reason is a change of the system. Thus, an already completed system might be changed due to new requirements. The control flow might restart at every manual activity in the process. Edges that represent this change are illustrated in red. Finally, implementation of systems is always an error-prone activity. Therefore, many options in control flow result from error correction activities (illustrated in green). Such edges might originate from every point in development, where new or changed artifacts become available or are used for further activities.

Similar to changes of the system, correction activities might restart at every manual activity that was performed before the need for the correction was identified.

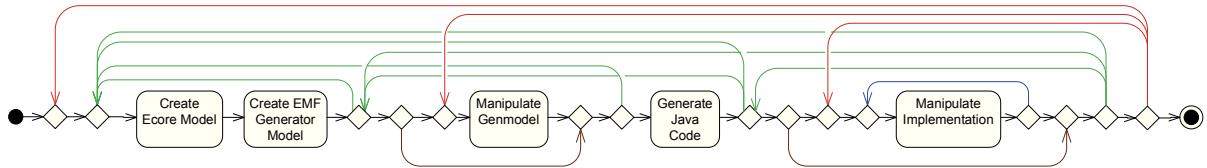


Figure 6.3.: Control flow edges that are introduced to represent orders of activities that are caused by different reasons (distinguished by color). Notation based on UML activity diagrams [156].

A closer look at the resulting control flow from Figure 6.3 reveals that some edges that are introduced for different reasons are duplicates. Therefore, in Figure 6.4 the control flow without these duplicates is shown. The edges that are illustrated in purple represent control flow that can be caused by different reasons. For example, the repetition of activity “manipulate implementation” can represents that the activity is performed on multiple artifacts, that the activity is performed again to correct an error, or that the activity is performed to implement a new requirement.

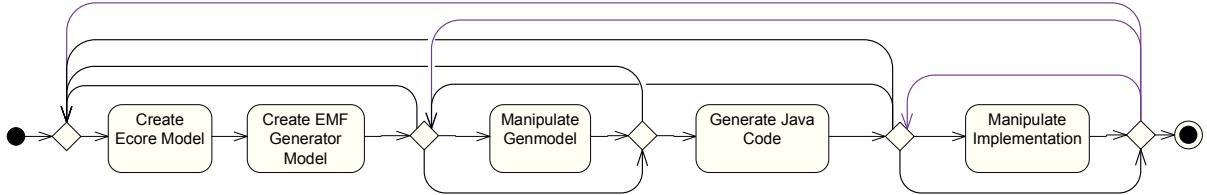


Figure 6.4.: Complete control flow for *EMF example* (Section 2.6). Notation based on UML activity diagrams [156].

This example shows that an illustration of the *complete* control flow (i.e. all technically possible orders of activities) can result in a complex model, even if only few activities are considered. In addition to *optionality*, *changes of the system* and *correction activities*, which can be observed in this example, also *alternative activity chains* can become relevant in other cases.

However, when approaching analysis goals like the identification of the lengths of activity chains or synchronization points, it is necessary to identify what orders of activities are possible and what orders are not. For example, based on the intuitive control flow shown in Figure 6.2, the minimum number of activities to introduce a change in activity “*create ecore model*” would be rated to high, since “*manipulate genmodel*” might not be required. In addition, it is relevant to differentiate between different causes for activity orders (e.g. whether an order represents one of the *alternative activity chains* or results from a correction loop). For example, in context of a change or correction after “*manipulate implementation*”, this activity can be followed by “*manipulate genmodel*” and “*Generate Java Code*”. This kind of order between “*manipulate implementation*” and “*Generate Java Code*”, however, needs to be differentiated from the order between “*manipulate genmodel*” and “*Generate Java Code*”, when reasoning about whether activity “*Generate Java Code*” is a *synchronization point* (see Section 3.3.3).

Subsuming, it is difficult to create control flow models that reflect all possible control flows (or all control flows excluding cycles for changes and error correction). In contrast, it is relatively easy to create object flow models (e.g. as shown in Figure 6.1), which include constraints on the order of activities. Since these object flow models can be used to derive the required information about control flow, the language design of Software Manufacture Models focuses on object flow.

To enable reasoning about the possible control flows, the Software Manufacture Model language will allow to model cardinalities for consumed and produced artifacts. Therefore, the concept *artifact role* is introduced. Instead of modeling artifacts, only artifact roles are modeled. During development one or more artifacts can have the same artifact role. An artifact role can be illustrated multiple times in a

6. Modeling

model. This can enhance readability, since an artifact role might be used in multiple activities. Figure 6.5 illustrates how this can look like for the *EMF generation* example.

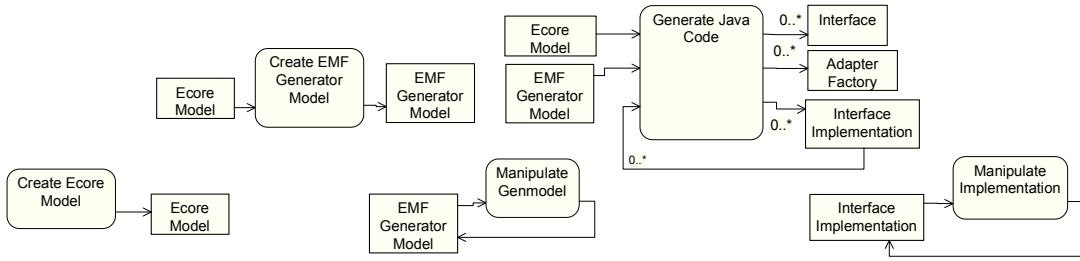


Figure 6.5.: Activities of the *EMF example* (Section 2.6). Notation based on UML activity diagrams [156].

6.1.2. Pre und Postconditions

The language should be a basis for the analysis of an MDE setting's influence on changeability concerns. The hard changeability concerns “*unexpected loss or preservation of content of artifacts*” and “*unexpected loss or preservation of references between artifacts*” deal with changes in the content of artifacts as well as references between artifacts (see Section 3.2.2). Consequently, the language requires concepts that allow expressing how content and references change when activities are applied.

Therefore, a concept is introduced to express preconditions and postconditions on relations between consumed and produced artifacts. Such relations can be information about content overlaps (i.e. how is the content of two artifacts related) or information on the existence of explicit references from one artifact to the other. Without information about changes in the content relations and references between artifacts, it is barely possible to reason about the actual character of an activity.

Example 2

For example, the activity shown in Figure 6.6 can have quite different consequences. Three examples are the following:

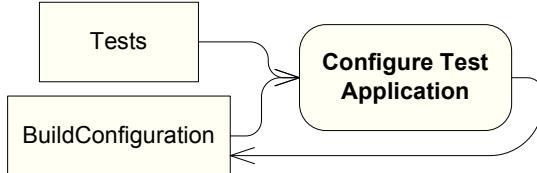


Figure 6.6.: Example activity “*configure test application*” with two input artifacts, where one of them is also output artifact of the activity. Notation based on UML activity diagrams [156].

- “Build configuration” is completely overwritten by a generation based on “tests”. This might happen, when the activity “configure test application” is an automated activity that was implemented to support developers in creating the “build configuration”. However, such an activity can lead to problems, when the generated “build configuration” is not complete and needs to be finished manually. Thus, when the activity is performed a second time to address changes in the “tests”, the hard changeability concern unexpected loss of content is affected.
- “Build configuration” and “tests” are merged and the result is written to “build configuration”. This version of the activity might be applied to prevent the problem of the previous version. However, in this case another hard changeability concern might be affected: unexpected preservation of content. When “tests” are changed and the activity is executed again, the full content of “build configuration” is preserved – including the content of the previous version of tests, which might still be part of “build configuration” (if it was not removed by another activity).

- “Build configuration” is changed, so that it has a reference to “tests”. This reference can be a precondition for the later execution of the “tests” or even the whole execution of the system (if the corresponding interpreter refused to execute “build configuration”, when a reference to “tests” is missing).

This example illustrates that the impact of an activity on changeability cannot be predicted only on the information about consumed and produced artifacts. However, this prediction can be improved when information is available about created references and moving content. Another reason for the introduction of these concepts is that they allow excluding impossible or useless orders of activities. This shall be illustrated on an example here.

Example 3

In Figure 6.7 two activities are shown: “adapt configuration” and “interpret”. Figure 6.7 provides only some information about the possible orders of these two activities.

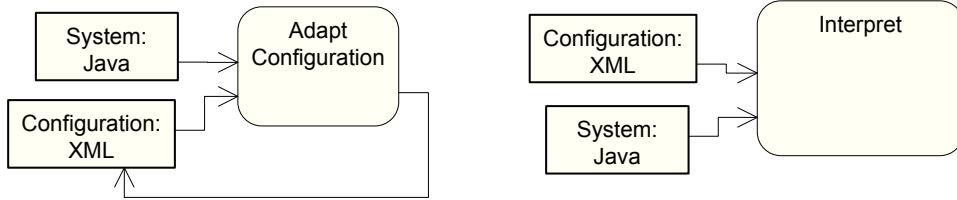


Figure 6.7.: Example of two activities with unclear order. Notation based on UML activity diagrams [156].

Both consume the same artifacts and none of them initially creates an artifact that is consumed by the other. The only hint for a possible order is the fact that “configuration” is adapted in “adapt configuration”. Thus, it might be desired that the results of this adaption are available when “interpret” is executed. However, this might not be necessary. The activity “adapt configuration” might be used for a readjustment, e.g. to enter a mode where logging is more detailed. In this case it is possible to perform activity “interpret” without a preceding execution of activity “adapt configuration”. Thus, given Figure 6.7, only, it is barely possible to exclude orders of these two activities.

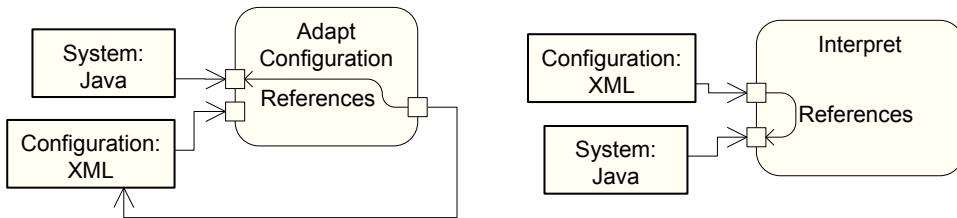


Figure 6.8.: Example of two activities with more clear order. Notation based on UML activity diagrams [156].

However as shown in Figure 6.8 information about preconditions and postconditions, concerning explicit references between artifacts, helps to exclude orders of activities. Here, the activity “interpret” has the precondition that the “system” is referenced by the “configuration”. Further, activity “adapt configuration” creates such a reference. Given a situation where activity “adapt configuration” is the only activity in the MDE setting that creates this reference, it can be excluded that activity “adapt configuration” is not executed before activity “interpret”.

The example illustrates how important information about the discussed pre- and postconditions can be for excluding impossible orders of activities. When it is not possible to exclude enough wrong orders of activities this can have impact on the evaluation of the length of activity chains or on the ability

to identify synchronization points. For example, when the necessity to execute some activities is not visible on the basis of input and output artifacts, only, the minimum length of an activity chain might be estimated too short. Further, if it is not possible to identify that certain activities can only be executed after an activity under study, the identification of this activity as synchronization point might be missed.

When introducing a concept to express how the content between two artifacts is related, the question arises, to which level of detail such content relations should be expressed. For example, it is possible to distinguish between an overlap of the contents of both artifacts and a situation where the content of one artifact is completely included in the content of the other artifact. Here, the application of the *initial Software Manufacture Model language* to MDE settings from practice helped to balance the level of detail according to what was actually used (the resulting supported level of detail is described in Section 6.2.1).

6.1.3. Degree of Automation

In order to evaluate what manual effort is required during development and to identify manual information propagation, information about the degree of automation of activities is required. Further, the information about the degree of automation is necessary to identify situations, where hard changeability concerns might be affected. Consequently, the Software Manufacture Model language should include a concept to explicitly express the degree of automation for activities.

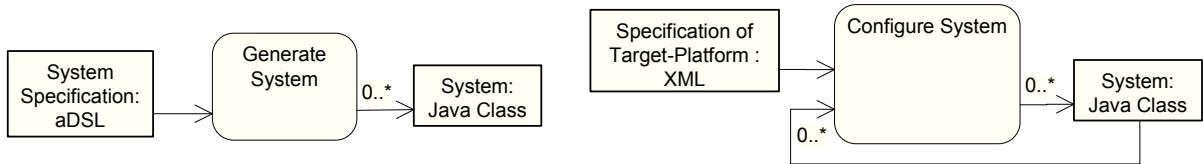


Figure 6.9.: Example for two activities where the hard changeability concern *unexpected loss of content* might be affected, depending on the degree of automation. Notation based on UML activity diagrams [156].

Example 4

Consider, for example, the two activities that are shown in Figure 6.9. The first activity “generate system” can be used to create a system implementation in Java on the basis of a “system specification” that is formulated in a DSL (“aDSL”). The second activity “configure system” manipulates a system implementation and consumes therefore a “specification of the target-platform”. While the name of the first activity allows guessing that this activity is automated, it is not clear whether the second activity is automated or not.

Consider the following situation: “system” was initially built (e.g. both activities were executed once) and, subsequently, the “system specification” is adapted due to the changing requirements. To propagate the changes into the “system”, both activities might be executed again. Activity “generate system” creates a new “system specification” and might even overwrite the old system specification. In this situation the cases that “configure system” is a manual activity or completely automated need to be distinguished. If “configure system” is a manual activity all manual effort that was put in the initial configuration of the system has to be invested again. Thus, the hard changeability concern unexpected loss of content might be affected here. However, the situation changes immensely, when “configure system” is completely automated. In this case no manual effort has to be invested at all. Instead the automated configuration is just executed again.

This example illustrates that absence of information about the degree of automation prevents an automated identification of some harmless situations (i.e. exclusion of risks). As a consequence, more situations with a potential to be problematic need to be investigated manually, when analyzing an MDE setting. Thus, it becomes clear that an explicit inclusion of information about the degree of automation is important. Theoretically, this information can be provided at different levels of detail. For example, the *initial Software Manufacture Model language* did allow a distinction which artifact relations in a semi-

automated activity are established manually, semi-automatically, or fully automated. However, this option was barely used. In most cases all artifact relations in a semi-automated activity are established semi-automatically, too. Therefore, the degree of automation will be expressed per activity.

6.1.4. Hierarchy

Many technologies for automated generation, transformation, or synchronization of software artifacts take advantage of separation of concerns within artifacts. On this basis the technologies can support changeability by preserving parts of artifacts (e.g. protected regions) or by working incrementally. To make such local access to artifacts visible, the language shall include a concept to illustrate hierarchy of artifacts. Similar to the degree of automation, the ability to model hierarchically encapsulated artifacts can help to exclude risks to changeability for some situations, without the need to examine them manually.



Figure 6.10.: Example for two activities, where it cannot be excluded that the hard changeability concern *unexpected loss of content* is affected. Notation based on UML activity diagrams [156].

Example 5

Let's consider another variant of the example from Section 6.1.3. For this example (shown in Figure 6.10) time it is known that activity “configure system” is a manual activity. In addition, activity “generate system parts” does not only produce the “system” artifacts, but also considers their former versions during execution. Based on the model in Figure 6.10 it is not possible to say whether both activities lead to a manipulation of the same parts of the artifact or not. As a consequence it cannot be excluded that the hard changeability concern unexpected loss of content is affected.

However, the introduction of hierarchy into the models can help to identify situations where different parts of an artifact are accessed. In Figure 6.11 one such example is shown. In this case it is shown that the automated activity “generate system parts” manipulates the “main-method” of the class, while the manual activity “manually configure system” is restricted to the “class attributes”. Given the knowledge that both parts of the artifact, “main-method” and “class attributes”, do not overlap, it is possible to exclude the risk that the hard changeability concern unexpected loss of content is affected in this example.

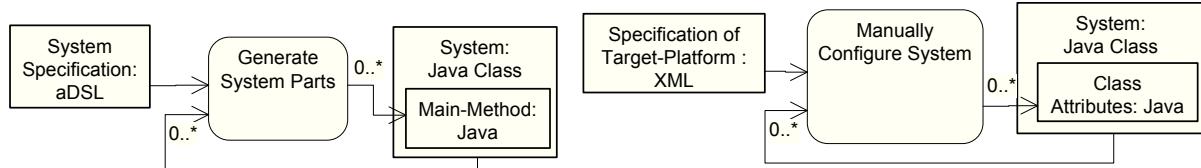


Figure 6.11.: Example for two activities, where the knowledge about the artifact hierarchy allows to exclude that the hard changeability concern *unexpected loss of content* is affected. Notation based on UML activity diagrams [156].

The language will not include a restriction concerning the depth of the nesting. Further, the nesting is for now not checked for conformance to the language specification of the exterior artifact. This is on the one hand a pragmatic decision, since such a check would require knowledge of language specifications, like grammars or meta models, of all languages and artifact types used in the modeled MDE setting. On the other hand, the separation within the artifact is not necessarily attached to elements of the

language, but is often marked in form of special comments. For example, protected regions within the EMF generation of models are marked with the comment “@generated NOT”.

6.2. Software Manufacture Models

As a modeling language for capturing MDE settings the Software Manufacture Model language is introduced in this section. A Software Manufacture Model is defined here as

a special process model, where the characterization of MDE activities captures how artifact relations change.

Thus, Software Manufacture Models combine software process models with information how process activities change or can change the megamodel of the system (i.e. the relations between models and other development artifacts).

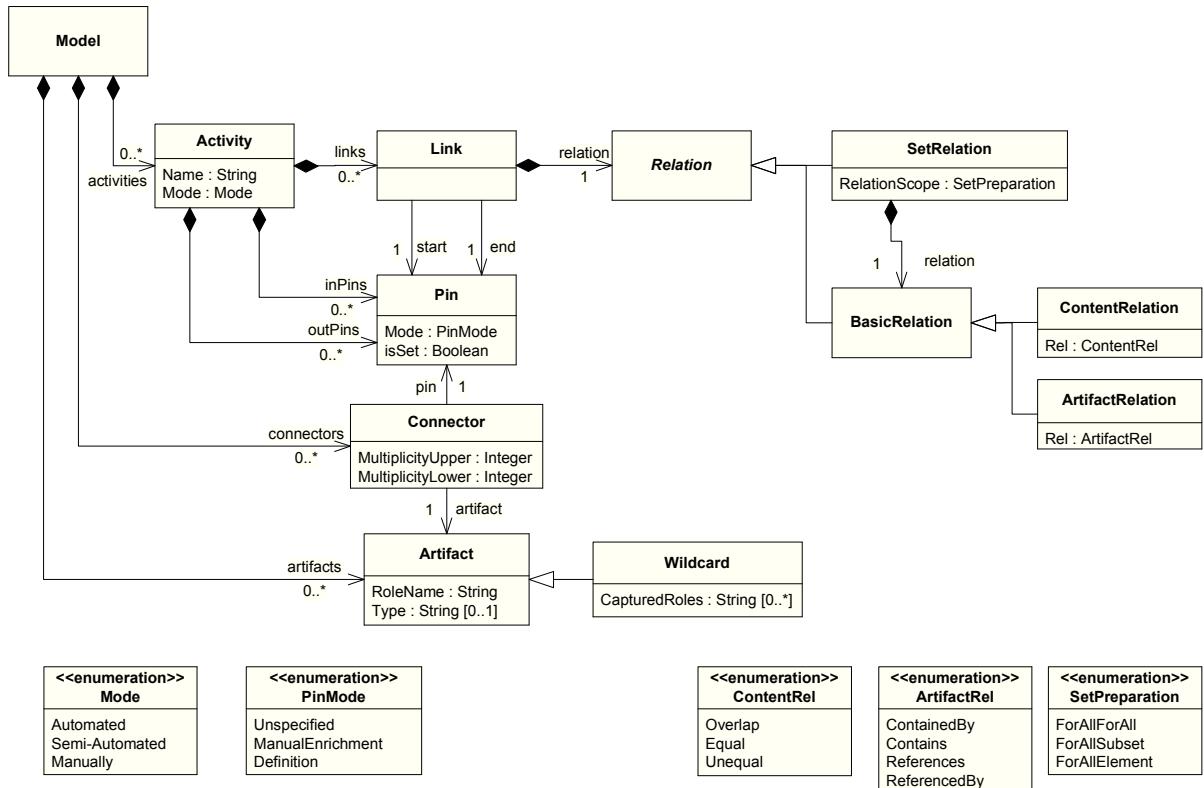


Figure 6.12.: Meta model of Software Manufacture Model language

In Figure 6.12 the meta model of the Software Manufacture Model language is shown. In the next Section the different aspects of the Software Manufacture Model language are introduced in detail, followed by a set of examples. In Section 6.2.2 a simplification mechanism is presented that enables a compact notation of complex activities. The semantic of activities with respect to changes in a megamodel is explained in Section 6.2.3 and following in Section 6.2.4 the terms *predecessor* and *successor* are introduced to allow reasoning about the order of activities in a Software Manufacture Model.

6.2.1. Language Description

A model consists of activities, artifacts, and connectors. A connector connects an artifact to a pin. Such, pins belong to activities and represent their input and output parameters (i.e. pins are the connection points between an activity and artifacts). Together, these four meta model elements can be used to express object flow in Software Manufacture Models.

As shown in the notation key in Figure 6.13, the notation of activities and artifacts is similar to the notation of UML activity diagrams [156]. Activities are shown as rectangles with rounded corners and artifacts are shown as rectangles. Pins are shown as small squares at the edging of an activity. Connectors are shown as arrow. In case the connector connects an artifact to an input pin, the arrowhead directs to the pin. Else, in case the connector connects an artifact to an output pin, the arrowhead directs to the artifact.

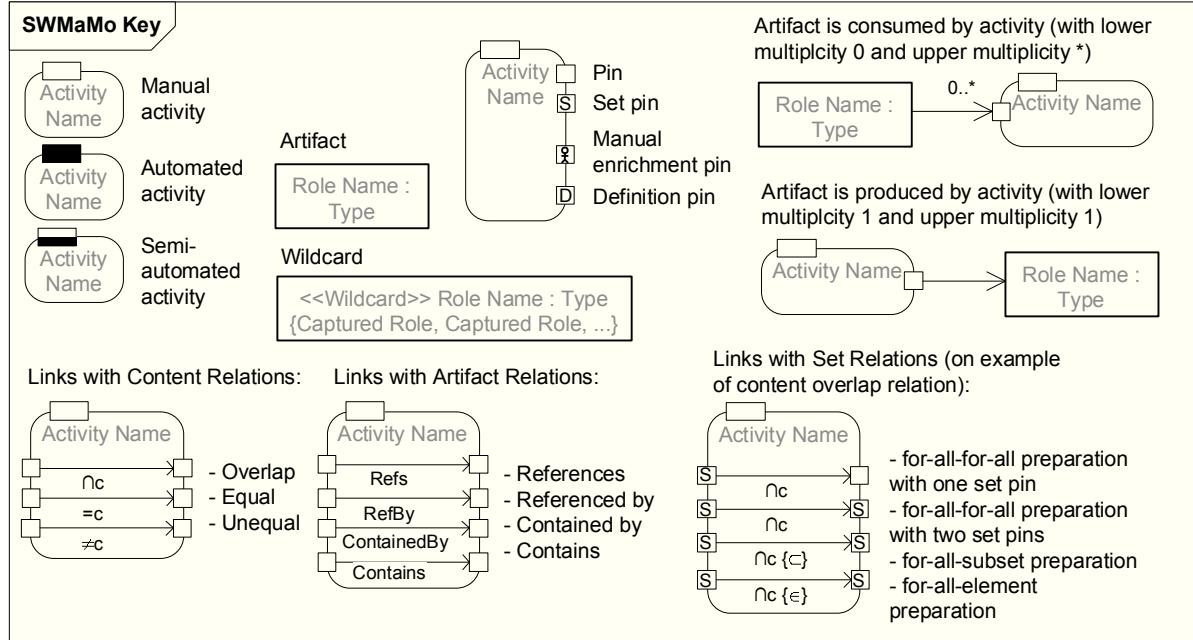


Figure 6.13.: Concrete syntax of Software Manufacture Model language

Figure 6.14 shows on a part of the EMF generation example how these four introduced model elements can be used to express object flow. It is possible that the same artifact is connected to two activities (e.g. “*ecore model*”). Alternatively, an artifact might be modeled twice to enhance readability of the model (e.g. “*emf generator model*”).

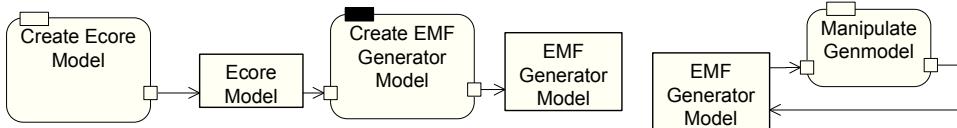


Figure 6.14.: Modeled object flow of a part of the *EMF example* (Section 2.6).

Activities

An activity is identified by a name, which can be chosen, such that it informally describes the activity. As already mentioned above each activity consists of two potentially empty sets of pins: input pins (*inPins*), and output pins (*outPins*). Artifacts that are connected to input pins belong to the precondition of the activity and are consumed by it. Artifacts that are connected to output pins belong to the postcondition of the activity and are produced by it (or modified, in case the artifact is also connected to an input pin).

Further, an activity has a potentially empty set of links, which are used to further describe precondition and postcondition of the activity concerning the relations of artifacts that are consumed, produced, or modified by the activity. Finally, an activity has a mode. This mode describes the degree of automation of that activity as *automated*, *semi-automated*, or *manually*.

6. Modeling

As summarized in Figure 6.13, the mode of an activity is illustrated by a rectangle that is placed at the upper left corner of the activity. The rectangle is white for mode *manually*, black for mode *automated*, and black and white for mode *semi-automated*. For example, in Figure 6.14 three activities from the emf generation example are shown (illustrated without links). Activity “*create.ecore model*” is a manual, while “*create.emf generator model*” is automated. In Figure 6.15 the semi-automated activity “*generate proxy*” from the case study SIW is shown.

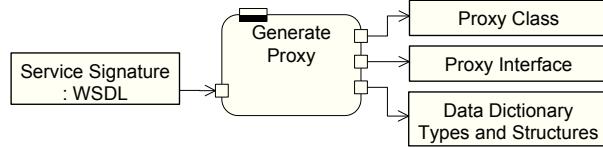


Figure 6.15.: Representation of activity “*generate proxy*” from case study SIW without specification of links

Artifacts

An artifact in the model represents a role, actually. During application of an MDE setting there can be multiple artifacts that fulfill this role. For example, there might be multiple Java source code files that are compiled during the same compilation activity.

A modeled artifact has a role name. In addition, a type can be annotated to indicate that different languages or file formats are used. As summarized in Figure 6.13, role name and type are separated by a colon. For example, the artifact that is consumed by the activity shown in Figure 6.15 has the role name “*service signature*” and is of type “*WSDL*”. In contrast the produced artifacts are only annotated with role names.

Connectors

A connector connects a modeled artifact to a pin. In addition a connector has an upper and a lower multiplicity. This multiplicity defines how many artifacts that fulfill the role of the modeled artifact can, during execution, be consumed or produced by the activity in the way that is described for the connected pins. The lower multiplicity describes the minimum number of artifacts, while the upper multiplicity describes the maximum number of artifacts. When the maximum number of artifacts is not constrained (i.e. can be arbitrarily high) the upper multiplicity is set to “****”. Per default lower and upper multiplicities are equal to one. In this case, they do not need to be annotated in the model.

As summarized in Figure 6.13, the multiplicity is annotated to the connector. The lower multiplicity precedes the upper multiplicity. For example, in Figure 6.16 activity “*generate.java code*” from the emf generator example is shown. One artifact of the role “*ecore model*” is consumed and arbitrary many “*interface implementations*” might be produced.

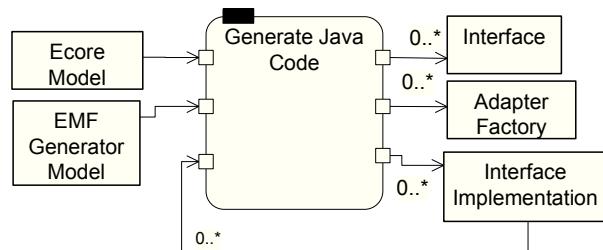


Figure 6.16.: Representation of activity “*generate.java code*” from EMF generation example without specification of links and pin types

Pins

A pin belongs to an activity and has a pin mode, which is either *unspecified*, *manual enrichment*, or *definition* (as specified in the metamodel in Figure 6.12). The mode *definition* describes that the artifacts connected to the pin define how the activity is automatically executed (such as a transformation specification). A pin of this mode is an input pin. The mode *manual enrichment* describes that the artifacts connected to the pin are enriched manually with content that was not there in any input artifact during the activity. A pin of this mode is an output pin. All pins that are neither manual enrichment pins nor definition pins are unspecified. As summarized in Figure 6.13, unspecified pins are annotated as white squares. Pins of mode *manual enrichment* are annotated as squares with a stick man inside. Finally, a pin of mode *definition* is shown as a square with a “D” inside.

As already indicated above, a connector between an artifact and a pin defines a multiplicity. Pins that allow the consumption of more than one artifact (i.e. where connectors with an upper multiplicity greater than one or multiple connectors are attached) are modeled as set pins. Each pin is either a set pin or not. As summarized in Figure 6.13, a set pin is shown as a square with an “S” inside.

Three example activities are shown in Figure 6.17. The artifact “*tests*” is connected to the manual activity “*adapt tests*” via an input pin and an output pin, which is a manual enrichment pin. This indicates that the developer, who performs the activity, does not only introduce content from the other input artifact “*implementing class*” to artifact “*tests*”. For example, the developer might extend “*tests*”, such that a special case is taken into account. The activity “*execute transformation*” has a definition pin, which is connected to artifact “*transformation specification*”. This indicates that this artifact specifies how the activity consumes “*model 1*” to produce “*model 2*”. Finally, the activity “*merge into*” consumes multiple “*models*”, which are merged into the “*documentation*”.

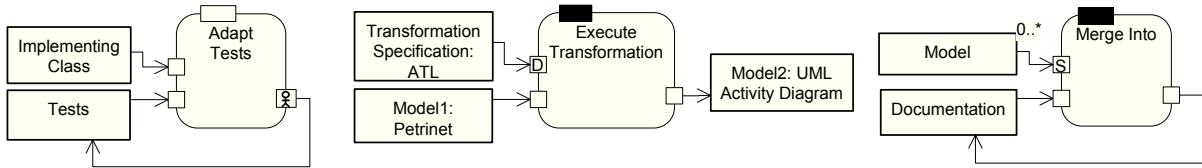


Figure 6.17.: Representation of three activities: “*adapt tests*”, “*execute transformation*”, and “*merge into*”, without specification of links

Links and Relations

Links are used to describe what relations between artifacts belong to the precondition and postcondition of an activity. Links are modeled between two pins (a start and an end). Both pins can be input pins or output pins of the activity that contains the link. Links that are modeled between two input pins belong to the precondition of the activity and thus describe what artifact relations are required before the activity can be executed. Links between two output pins, links from an input pin to an output pin, and links from an output pin to an input pin belong to the postcondition of the activity. Thus, these links describe what (new) relations exist between artifacts after the activity was executed.

A link has a relation, which specifies how artifacts are related to each other. The relation is formulated in the direction specified by the link, e.g. artifacts connected to the start pin are in this relation to artifacts connected to the end pin. When artifacts are connected to an input pin and an output pin as well, links that are connected to the input pin describe relations to the consumed version of the artifact (i.e. the version of the artifact that existed before the activity was executed) and links that are connected to the output pin describe relations to the resulting version of the artifact. Thus, a link between an input pin and an output pin that connected to the same artifact can be used to express how the artifact changed. As summarized in Figure 6.13, links are modeled as arrow between two pins within the activity. The arrowhead directs to the end pin of the link and the relation is annotated alongside the link.

There are three types of relations: two basic relation types as well as set relations, which are used to describe for which subset of artifacts that are connected to a set pin a basic relation holds. The two

6. Modeling

basic relation types are content relations and artifact relations. When one or both of the connected pins is a set pin, the relation needs a set relation.

Content relations describe how the contents of two artifacts are related to each other. There are three content relations that can be modeled within Software Manufacture Models: *overlap*, *equal*, and *unequal*. An *overlap* can be used to express that the contents of both artifacts overlap. For example, system aspects that are specified in a platform independent model are also included in a corresponding platform specific model within an MDA approach. This platform specific model includes additional content that concerns the platform specific aspects. An *overlap* between the content of artifacts can also occur when different views or aspects of the same system parts are expressed in two artifacts (e.g. of architecture and behavior). For example, objects that are modeled in a sequence diagram contain information about what classes are parts of the system, which is also content of a corresponding class diagram.

An *equal* relation is a more specialized form of an *overlap* relation. Here it is expressed that none of both artifacts contains content that is not contained by the other, too. This happens, when copies of artifacts are created (e.g. when a release is published) or in case of transformations, that just translate artifacts between two languages. Content relations of type *equal* or *overlap* that are modeled between input pins and output pins (or in the other direction) express that *content flows* from the artifacts that are connected to the input pin to the artifacts that are connected to the output pin.

An *unequal* relation can be used to explicitly express that the contents of the connected artifacts are not completely equal. For artifacts that are connected to an input pin and an output pin of the activity, *unequal* relations can be used to express that the resulting version of the artifact is created during the activity new. Content is not systematically preserved from the consumed version of the artifact and overlaps in content are rather random. In contrast, an *overlap* relation between the consumed and resulting version of an artifact, indicate that – although content might be changed – parts of the content are preserved.

As summarized in Figure 6.13, the link annotation for content relations is accompanied by a “ C ”. Thus, a link with an *overlap* relation is annotated with \cap_C , a link with an *equal* relation is annotated with $=_C$, and a link with an *unequal* relation is annotated with \neq_C .

Artifact relations are direct relations between two artifacts. This includes at the one hand the existence of explicit references between artifacts: *references* relations and *referenced by* relations. On the other hand there are containment relations: *contains* relations and *contained by* relations. Containment relations are used to express that one artifact is part of another artifact. For example, a method is a part of a class. Thus, a method is *contained by* a class and a class *contains* a method.

As summarized in Figure 6.13, a link with a *references* relation is annotated with *Refs*, a link with a *referenced by* relation is annotated with *RefBy*, a link with an *contains* relation is annotated with *Contains*, and a link with a *contained by* relation is annotated with *ContainedBy*.

Finally, set relations can be used to express content relations or artifact relations between artifact sets that are connected to set pins. Set relations consist of a relation scope and a basic relation, which is either a content relation or an artifact relation. The relation scope can be used to define for what parts of the artifact sets, which are connected to the two pins, the described basic relation holds. The default relation scope is *ForAllForAll*, which indicated that each artifact in the set of artifacts connected to the start pin is in the described basic relation to each artifact in the set of artifacts connected to the end pin. The relation scope *ForAllSubset* is similar to a *ForAllForAll* scope with the difference that only a subset of the set of artifacts connected to the end pin is affected. Thus, *ForAllSubset* indicates that each artifact in the set of artifacts connected to the start pin is in the described basic relation to each artifact in a specific subset of the set of artifacts connected to the end pin. Finally, the relation scope *ForAllElement* is a special case of the relation scope *ForAllSubset*, where the affected subset of artifacts connected to the end pin has the size one. Thus, *ForAllElement* indicates that each artifact in the set of artifacts connected to the start pin is in the described basic relation to a specific artifact in the set of artifacts connected to the end pin.

In practice it is sufficient to use relation scopes where one of both sets is restricted, only. The possible relation scopes defined for Software Manufacture Models are one-sided in a sense that this restriction is defined for the set of artifacts connected to the end pin. This implies no constraint, since the direction of the link is not bound to the question what artifacts are consumed and produced by the activity. Further, content relations are symmetric (i.e. the meaning does not change with the direction) and artifact relations can be expressed in both directions (e.g. *references* vs. *referenced by*).

In principle links and relations modeled for a set might be split up to modeled links and relations for the single artifacts within the set. However, this might lead on the one hand to redundant information, and is on the other hand only possible if the exact number of artifacts in the set is known.

As summarized in Figure 6.13, the link annotation for set relations with *ForAllForAll* prefix consists of the annotation of the basic relation, only. The link annotation for set relations with *ForAllSubset* prefix consists of the annotation of the basic relation followed by $\{\subset\}$. The link annotation for set relations with *ForAllElement* prefix consists of the annotation of the basic relation followed by $\{\in\}$.

Example 6

To get an impression how the above introduced language looks like, the models of four activities are presented in the following. All four activities have basically the same structure as the example from Figure 6.6 in Section 6.1.2: there are two consumed artifact roles and one of them is output of the activity as well.

The first example is the activity “adapt tests” that is shown in Figure 6.18. “Adapt tests” is a manual activity, where tests are adapted manually with respect to an implementing class. Content of the “implementing class” is moved to “tests”. Also the input version of artifact “tests” overlaps with the output version of this artifact, which indicates that the content is modified. Finally, the manual enrichment pin indicates that the new content of “tests” is not only propagated content from “implementing class”, e.g. like new covered test cases.

The second example in Figure 6.18 is the automated activity “generate new”. The activity models a generation step that takes a model and generates a new version of source code (i.e. content is moved from the “model” to the output version of the “code”). The content of the old input version of the code is not preserved (i.e. the artifact is overwritten – there is no systematic equivalence between input version and output version of the “code”).

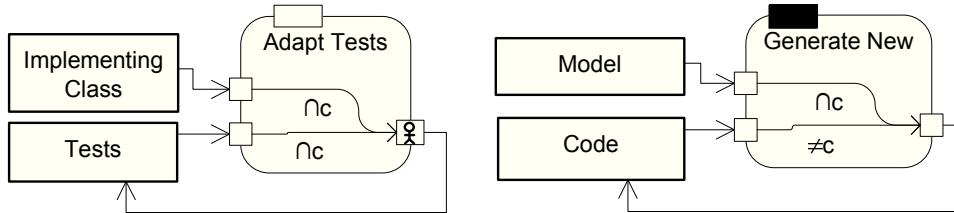


Figure 6.18.: Activities “adapt tests” and “generate new”

In Figure 6.19 the activity “configure test application” from Figure 6.6 in Section 6.1.2 is shown in Software Manufacture Model notation. Now it can be seen that the consequence of the activity is that the “build configuration” is changed, such that it gains references to “tests” (i.e. as a result of the activity the tests that shall be executed are references by the build configuration). The activity is semi-automated, i.e. there is tool support, but addition manual decisions are required.

A fourth activity is “merge into” which is shown in Figure 6.19, too. “Merge into” automatically merges multiple “models” into the “documentation”. This is modeled by a multiplicity at the connector from artifact “models”. The content of the “documentation” is modified, but mainly preserved (modeled by the overlap relation). Further, the set relation between the pin connected to “models” and the out pin connected to the “documentation” has a *ForAllForAll* scope, which is default. Thus, content from each “model” is moved to the “documentation”.

A more complex example for an activity is shown in Figure 6.20. The activity “generate java code” stems from the EMF example (Section 2.6) and consumes the artifacts “ecore model” and “emf generator model” in order to generate a set of “interfaces”, “adapter factories”, and “interface implementations”. It is a precondition of the activity that the “ecore model” is referenced by artifact “emf generator model”. Content of these two artifacts is used to create the produced artifacts. Finally, artifact “interface implementation” contains a part that is referred to as “generated NOT”. Sets of both artifacts can be consumed by the activity, too.

The links describe the relations between the sets of input versions of both artifacts and the set of output versions of “interface implementation”. The first link describes that each consumed “interface implementation” overlaps in content with one of the created “interface implementations”. Thus, the ac-

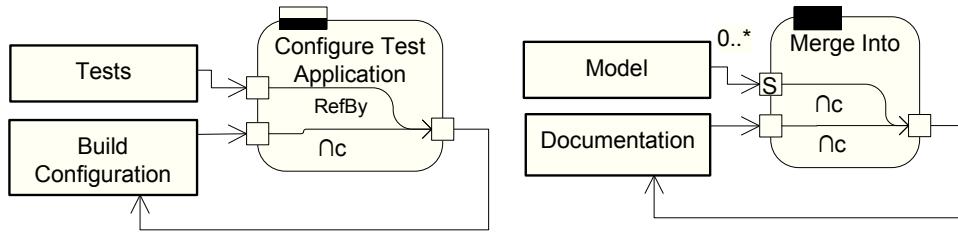


Figure 6.19.: Activities “configure test application” and “merge into”

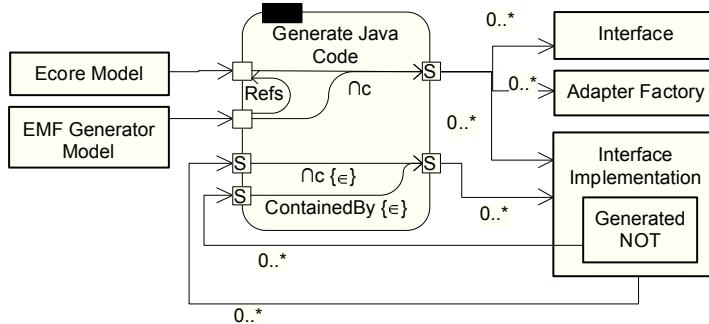


Figure 6.20.: Activity “generate java code” from *EMF example* (Section 2.6).

tivity does not only produce “interface implementations”, but can also adapt already existing “interface implementations”. The second link describes that each consumed version of artifact “generated NOT” is afterwards contained by one of the created or manipulated “interface implementations”. Thus, the activity “generate java code” preserves the artifact “generated NOT” when it is executed to manipulate “interface implementations”.

6.2.2. Notation of Complex Activities

On average each of the 228 activities that were captured for the 11 case studies of MDE settings from the first and third study, contains about 2 links. However, there are some (mostly automated) activities that consume and produce multiple artifacts and consequently contain more links. To tackle this complexity, the Software Manufacture Model language was initially introduced with some strategies to model activities more lightweight. Four steps of these strategies turned out to be useful during capturing and modeling the 11 case studies from practice.

To illustrate these steps one of the most complex activity from the case studies is chosen. The activity “*Generate Code*” from case study SIW consumes a configuration, context parameters, a set of templates, and a “*proxy class*” with its methods as input. On that basis, “*Generate Code*” produces an implementation for the methods of the “*proxy class*”, different artifacts for a “*data dictionary*”, different ABAP objects that include static parts and code slots for later completion, a “*mapping class*” that also includes static parts as well as a mapping slot for later completion, and finally different further objects, such as “*Table Entries*” and “*BAp Definitions*”. The consumed configuration defines how “*Generate Code*” is executed, comparable to a transformation specification. The mapping class is an ABAP object that is exposed here, since it is treated differently in further development steps. The proxy class already exists before the activity is executed, and has to be completed by “*Generate Code*”. In this special example, there are not only much different output artifacts, but that it is not clear before execution, which artifacts exactly are produced. The reason is that the set of produced artifacts depends strongly on the used templates, which are referenced in the configuration. In Figure 6.21 the compact characterization of the activity is shown.

Note that a main part of the complexity of this example activity stems from the fact that the results depend strongly on the actually used template. Thus, for an analysis on an MDE setting such an activity might be substituted by a more concrete activity that describes the effects of the activity for a specific template. However, the activity as is shown was captured during the studies.

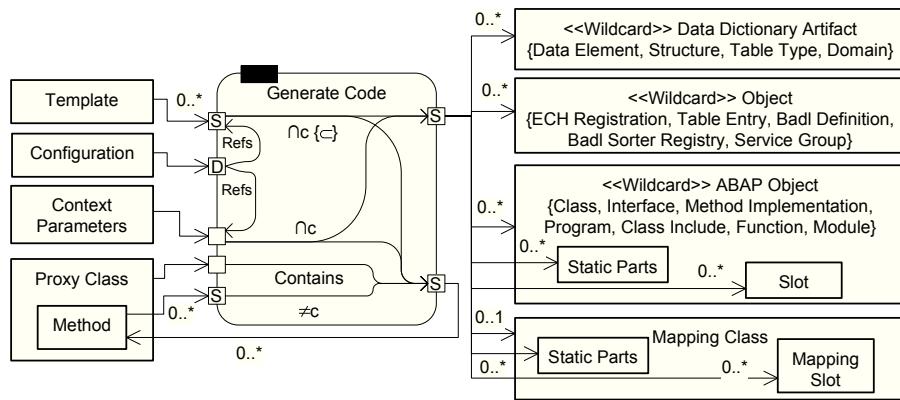


Figure 6.21.: A compact characterization of the activity “*GenerateCode*” from case study SIW

Especially in automated activities that allow an extensive configuration or base strongly on templates, it happens that a flexible number of diverse artifacts might be created or consumed. To tackle such uncertainties during modeling the modeler can in a first step introduce “wildcards”. A wildcard represents a set of artifacts with different roles and types that are treated within the activity as well as in the following development steps equally. In the example in Figure 6.21, three such wildcards were introduced: the Data Dictionary Artifacts, the ABAP Objects, and the Objects. Fortunately, the wildcard concept was required in three of the 228 activities, only. However, as in the shown example, the use of wildcards is for some activities essential and enables capturing the actual complexity while creating a model that is readable.

The next step for tackling complexity is the introduction of “internal sets”. This means that different input artifacts (respectively output artifacts) on an activity that have similar relations to other artifacts can be treated within the activity characterization as one set. Thus, the different artifacts are connected to the same set pin. For example, each “*template*” has a content overlap with a subset of all output artifacts. Further, each output artifact has a content overlap to the “*context parameters*”. Consequently, nearly all output artifacts can be summarized in one “internal set”. Exceptions are only the “*method*”s, which have further relations. Internal sets were used to model eight of the 228 activities.

The use of wildcards and internal sets, can lead to a loss of detail information when the relations of the combined artifact actually differ. For example, a combination of a *content equal* relation and a *content overlap* relation can result in a set relation that describes a *content overlap* for each artifact in the set. The detailed information that content between some of the artifacts does not only overlap, but is equal, gets lost.

In a third step, the activity can be modeled more clearly, when links describing the same relations are optically joined. For example, the two links between the “*templates*” and the above described internal set and the “*method*”s, respectively, are illustrated this way.

Finally, *containment* relations between two consumed or two produced artifacts might not be illustrated as links within the activity, but external. For example, the illustration of “*proxy class*” and “*method*” in Figure 6.21 shows that the “*methods*” are a subset of the “*proxy class*”. The same holds for the “*static parts*” and “*slots*” within the “*ABAP objects*” and “*mapping class*”.

6.2.3. Activity Semantic

The semantic of a Software Manufacture Model activity can be described as a *set* of possible changes in a megamodel that stores the actually available artifacts as well as their relations. The set of changes that are described by an activity results from the set of possible *application points* as well as the freedom given by unspecified side effects as well as freedom in specified results.

An *application point* is the set of artifacts (in the megamodel) that can be used to apply an activity on. The artifacts fulfill the artifacts roles that are specified as input of the activity, such that each specified artifact role is mapped to zero, one, or more artifacts in the *application point* according to the specified multiplicities and that all preconditions defined by the activity are fulfilled (e.g. specified relations between consumed artifacts exist). In each situation during development multiple *application*

6. Modeling

points might exist for an activity. For example, there may be multiple artifacts that might fulfill the same consumed artifact role specified by an activity. Further, the number of artifacts in an *application point* can vary, when the corresponding activity consumes a set of an artifact role with a variable multiplicity (e.g. 1..*). Whether, when on which application point an activity is actually executed is a decision that is made by the developers (or within a transformation chain, by the corresponding specification).

The application of an activity to an application point leads to two forms of changes. On the one hand there are possible changes that are not specified explicitly by the activity, but have to be expected as possible side effects. On the other hand there are the changes that are explicitly specified by the activity.

When artifacts are only produced by an activity it is easy to specify all created relations, to other artifacts. However, artifacts that are changed in an activity (i.e. artifacts that are input as well as output of the activity) might already have relations to other artifacts before the activity is executed. Thus, a part of the artifact might be referenced by other artifacts, the artifact itself might reference other artifacts, and the artifact might have content overlaps to other artifacts (or can be equal in content). These other artifacts are not necessarily considered in the activity. Consequently, such relations might get lost (but do not need to) as a side effect when the activity is executed.

Artifacts roles that are specified as output of an activity, only, lead to the creation of artifacts. The number of created artifacts for an artifact role can vary corresponding to the specified multiplicity. When old versions of the artifact existed, these are not taken into account, i.e. a completely new artifact is created (with own relations to other artifacts). In practice this new artifact might be stored by overwriting the old artifact.

Relations can be created for all artifacts that are consumed, manipulated, or produced in an activity. Again the actual number of relations depends on the actual number of artifacts mapped to the artifact roles, but also on the relation scopes of set relations. Between two produced or manipulated artifacts (i.e. relations specified between two output pins) or between a consumed and a produced or manipulated artifact (i.e. relations specified between an input pin and an output pin), content *overlap* and *equals* relations, *references*, and *referenced by* relations are created as specified in the activity. A specified content *unequal* relation leads to the explicit deletion of content *overlap* and *equals* relations between the corresponding artifacts. Finally, specified containment relations reflect that an artifact is or becomes part of another artifact.

Content *overlap* and *unequal* relations between an input pin and an output pin that is connected to the same artifact role indicates that the corresponding artifacts change. As a consequence the above described possible loss of relations to other artifacts can occur. However, when an *equal* relation is described between an input pin and an output pin that is connected to the same artifact role, this indicates that this loss of relations will not happen (since the content of the artifact is not changed). Note that if an artifact role is connected to an input pin and an output pin of an activity, a link that is connected to the input pin describes a relation to the *consumed* version of the artifact role. In case of a content relation this indicates that content of the *consumed* version of the artifact is moved to this other artifact.

6.2.4. Activity Orders: Predecessors and Successors

As explained above, one reason for basing the Software Manufacture Model language on the description of pre- and postconditions of activities is to enable an analysis of technically possible orders of activities. It is less important to derive a set of all possible order of activities for all concerns at once, than to answer questions what activity orders might be applied for specific concerns. Two central questions are:

- *Which activities need to be executed to gain a specific resulting artifact “X”?* Thus, it is a question which activities are predecessors for an artifact.
- *Which activities need to be executed to propagate changes from the specific artifact “Y” to the resulting artifacts?* Thus, it is a question which activities are potential successors of another activity.

Therefore, following terms are defined here: A predecessor set of a specific activity or artifact is a set of activities that are sufficient to create the precondition for the specific activity or to create the specific artifact. A predecessor of a specific activity or artifact is an activity that is part of a *predecessor set* (i.e. an activity that might be executed in order to create the precondition of a specific activity or to

create the artifact). For each activity or artifact multiple *predecessor sets* might exist (i.e. there might be alternative ways to enable the activity or create the artifact). A successor of a specific activity is an activity for which the specific activity is a *predecessor*. Note that – with this definition – none of both terms, predecessor and successor, refer to activity orders that result from correction or change of the system.

Further, a start activity is defined here as an activity without *predecessor*. Not only activities without precondition (i.e. without consumed artifacts) can be start activities. This is, because artifacts might be created external and are provided to the MDE setting. For example, a document might be created on the basis of a template that is reused. Similarly, a start artifact is defined here as an artifact that cannot be created within the MDE setting. Finally, the term resulting artifact is used to describe artifacts that are not used for the creation of other artifacts (i.e. artifacts from which no content flows to other artifacts). In Section 8.3.1 it is described how predecessor sets, predecessors, and successors are automatically identified within a prototypical implementation.

However, due to the possibility to express that artifacts are optionally consumed, it is possible that some ambiguities concerning the order of activities arise. This concerns mainly the differentiation of creation, manipulation, and recreation, but also ambiguities due to explicitly modeled correction cycles. To cope with these ambiguities, the following modeling conventions are defined.

Creation vs. manipulation: A first concern is the differentiation of an activity that manipulates an artifact role from an activity that creates an artifact role and additionally consumes other instances of that artifact role. The Software Manufacture Models language includes no concepts for explicitly differentiating this situation. To handle this following rule-of-thumb is introduced here: an activity is considered to create an artifact, if the artifact is only output of that activity or if the artifact is optional input (lower multiplicity is 0) as well as mandatory output (lower multiplicity is greater than 0) of that activity. Consequently, following modeling convention shall be applied to ensure a correct identification of predecessor sets:

MC1 *Models that are used for the analysis should be prepared, such that the multiplicities in the model allow a differentiation between manipulating and creating activities.*

Recreation vs. manipulation: Similarly, the Software Manufacture Model language includes no explicit constructs to differentiate between an activity that manipulates an artifact (and will be applied after an activity for the creation) and an activity that recreates the artifact in a later iteration of the development and takes the old version of the artifact into account. Currently a strict understanding of the definition of predecessors leads for both cases to the result that this (recreating or manipulating) activity is a successor of a creation activity for this artifact. This is basically correct, but does not reflect that – in context of a change – the recreating activity is used instead the creating activity. For a correct identification of predecessors and successors, the same modeling convention as for the differentiation of creating and manipulating activities might be applied (MC1). Thus, for a recreating activity the consumption of the old version of the artifact might be marked as optional, such that the activity is identified as creating instead of manipulating.

Correction cycles vs. implementation: Another characteristic of Software Manufacture Models that might lead to problems for the predecessor analysis is that it is not differentiated between an order of activities that represents “normal” development or enrichment and an order of activities that represents a correction cycle. This problem occurs when the result of a check or test is modeled in form of an error report artifact that is optionally consumed by e.g. one of the modeling or coding activities. With a strict understanding of the definition of predecessors, the check activity can be seen as predecessor of a modeling activity. This is not incorrect, but introduces ambiguities when the results of the predecessor analysis are used for further reasoning of development effort. To ensure comparable results, following modeling convention should be fulfilled:

MC2 *Within Software Manufacture Models that are used for the analysis, the modeled consumption of error reports should be removed.*

6.3. Software Manufacture Model Patterns

Goal of the Software Manufacture Model pattern language is to provide a simple approach to enable developers in capturing and summarizing structures in Software Manufacture Models, such that these

6. Modeling

structures can be associated with specific characteristics. A pattern can be used to support other developers in identifying parts in their Software Manufacture Models that should be investigated in more detail for the characteristics associated with the pattern.

The pattern match (i.e. the definition when a part of a Software Manufacture Model can be considered as occurrence of a pattern) is designed such that it is intuitively understandable. Thus, developers that document a pattern should be able to reason whether the associated characteristic holds for all Software Manufacture Model parts that can be matched to the pattern. The idea behind the pattern match that will be presented in this section is to identify activities in the Software Manufacture Model that have similar effects on how artifacts and their interrelations are created and change as the matched pattern activities. Matched activities can have additional effects on other artifacts and their relation, if these are not relevant for the pattern. In the following, the Software Manufacture Model pattern language and the pattern match are introduced and discussed.

6.3.1. Pattern Language

In Figure 6.22 the metamodel of the Software Manufacture Model pattern language is shown. The metamodel is mainly an extension of the metamodel of the Software Manufacture Model language (see Figure 6.12). In the following, the pattern specific extensions are presented.

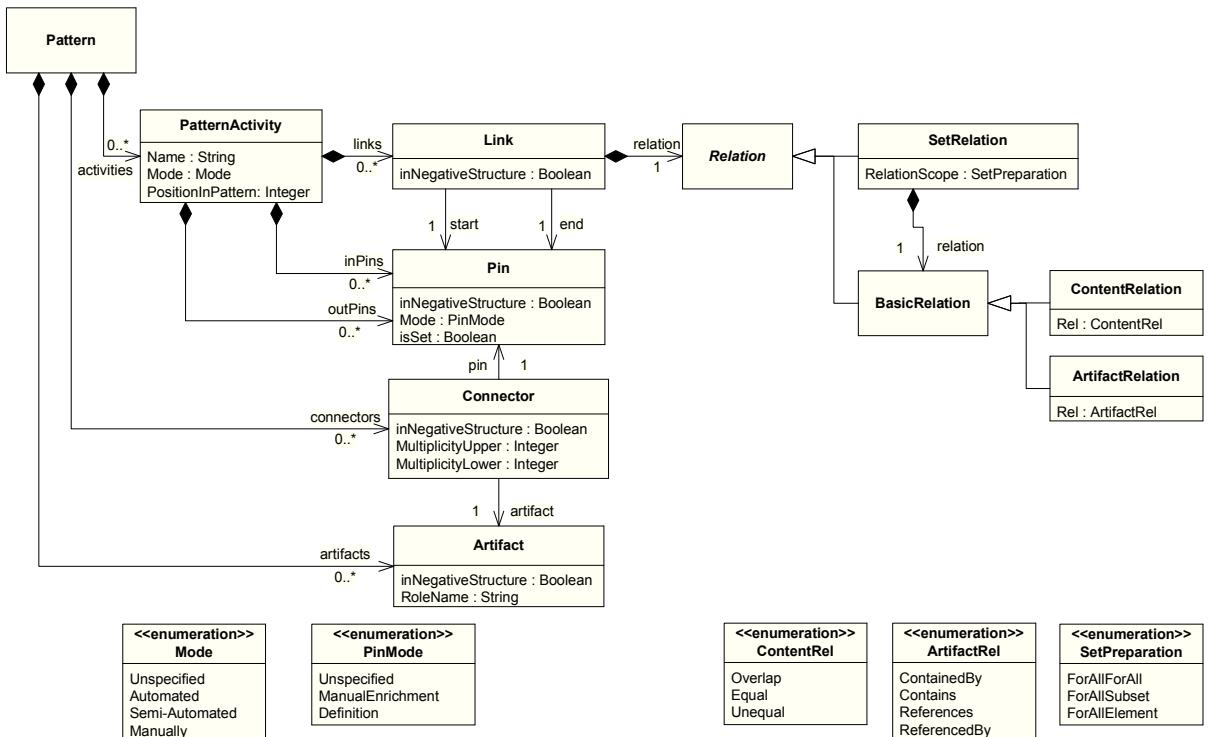


Figure 6.22.: Meta model of Software Manufacture Model Pattern language

A Software Manufacture Model pattern within the Software Manufacture Model pattern language contains one or more ordered pattern activities, which are similar to activities in Software Manufacture Models, but allow some degrees of freedom.

Artifacts within a Software Manufacture Model pattern represent roles for artifacts in a Software Manufacture Model. A pattern does not define a type of an artifact. The reason is that this type is not relevant for the analysis techniques that will be presented in this thesis. However, in future work an extension of the Software Manufacture Model pattern language might allow constraints on the type. For example, in Figure 6.24 the pattern *subsequent adjustment* is shown. Pattern activity *initial creation* has one input and one output. An artifact role that is connected to multiple pattern activities is shown as an own element for each pattern activity. This avoids the implication of a direct order between two pattern activities, as there can be multiple activities in a matched Software Manufacture Model that are

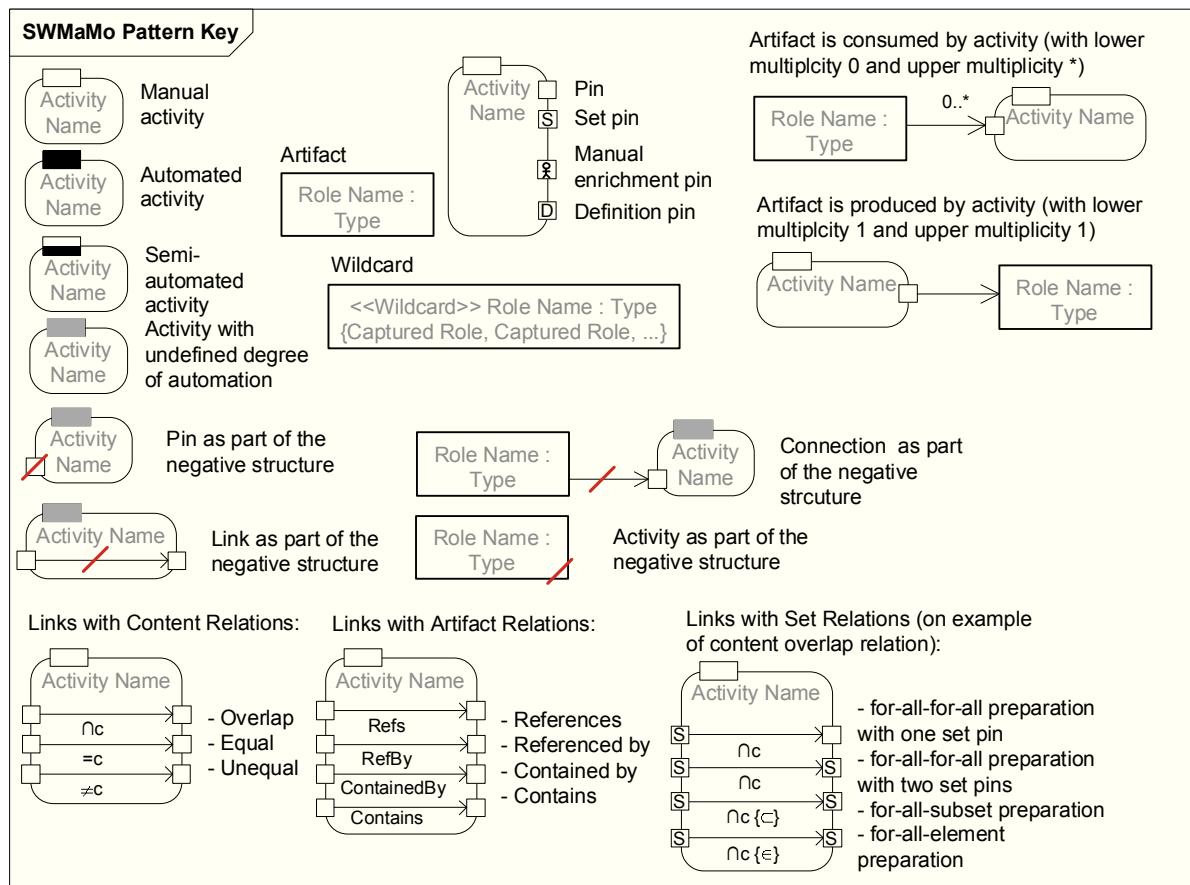
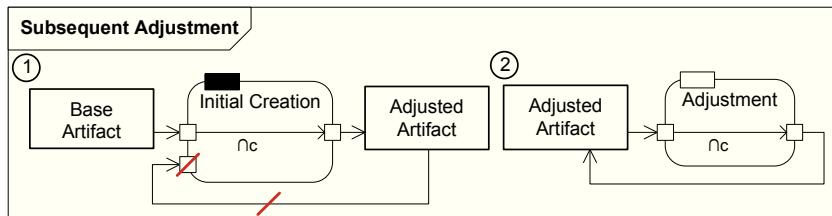


Figure 6.23.: Concrete syntax of Software Manufacture Model Pattern language

performed between two activities matched to pattern activities. As in Software Manufacture Models, an artifact can hierarchically contain another artifact, which can be modeled as a nesting or with the help of *containment relations*.

Figure 6.24.: Sample Software Manufacture Model pattern: *subsequent adjustment*

As described above, activities can have different modes in Software Manufacture Models. These modes allow indicating whether the activity is performed manually, semi-automatically, or automatically. To allow the definition of pattern activities without specifying this property, activities in Software Manufacture Model patterns can have the mode “undefined”. In the Software Manufacture Model activity *adjustment* the merge is performed manually, while activity *initial creation* is performed automatically.

A pattern activity describes only a part of an activity (i.e. an activity that is matched to a pattern activity can consume and produce additional artifacts). Therefore, “negative structures” can be specified in patterns. Such negative structures can be used to make explicit that a structure should not occur (comparable to negative application conditions in graph transformations [55]). The negative structure must not be matched when the corresponding pattern activity is matched to an activity in the Software Manufacture Model. As shown in the notation key in Figure 6.23, all elements (i.e. connectors, pins,

links, and artifacts) that are part of the negative structure are shown as crossed out elements (see Fig. 6.23). For example, in Fig. 6.24 the activity *initial creation* creates *adjusted artifact* without considering an input version of it.

6.3.2. Pattern Matching

To transform knowledge about a Software Manufacture Model pattern to knowledge about a concrete Software Manufacture Model it is necessary to identify a match between both. In the following, it is defined when a Software Manufacture Model pattern can be matched to a Software Manufacture Model.

For the illustration, a simplified version of the *EMF example* (Section 2.6) is used here. Figure 6.25 shows the corresponding activities.

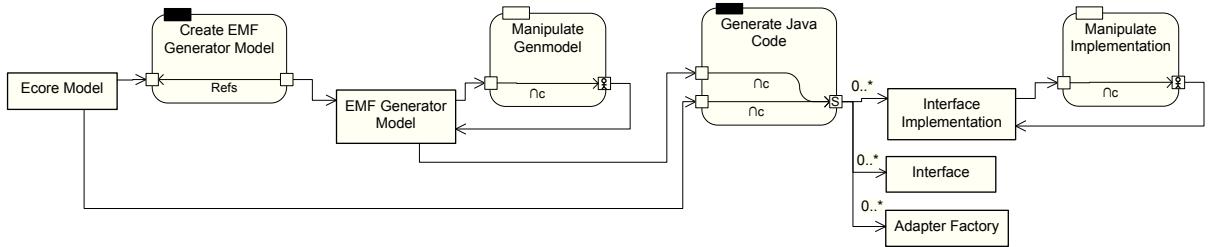


Figure 6.25.: Simplified version of the *EMF example* (Section 2.6) with Software Manufacture Model activities

A match of a Software Manufacture Model pattern to a concrete Software Manufacture Model consists of matches of each pattern activity to an activity of this Software Manufacture Model. The different pattern activities of the same pattern are matched to different activities. The order of the activities indicated in the Software Manufacture Model pattern has to conform to the possible orders of activities in the matched Software Manufacture Model. For example, pattern activity *initial creation* has to be matched to an activity that is a predecessor of the activity matched to *adjustment*.

A Software Manufacture Model pattern does not define how many other activities occur between two activities that are matched to two pattern activities. As shown in Figure 6.26, Software Manufacture Model pattern *subsequent adjustment* can be matched to the Software Manufacture Model of the EMF case study. A possible order of both activities conforms to the order indicated in the Software Manufacture Model pattern.

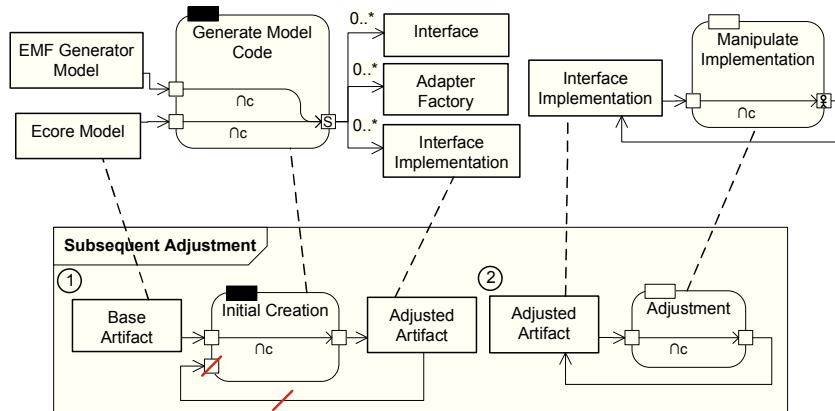


Figure 6.26.: Illustration of match of activities from the *EMF example* (Section 2.6) to Software Manufacture Model proto-anti-pattern *subsequent adjustment*

An activity in a Software Manufacture Model can be matched to pattern activities of different Software Manufacture Model patterns. The (non-negative) structure described in a pattern activity has to occur in the matched activity, too (isomorphic match). This means that two connected elements (activity,

connector, artifact, pin, or link) can only be matched to two similarly connected elements in the pattern. Two elements in the Software Manufacture Model pattern, describing the same artifact role, have to be matched to two artifacts of the same type and role or to the same artifact, respectively. Two distinct elements in the Software Manufacture Model pattern have to be matched to two distinct elements in the Software Manufacture Model.

Further, the mode of a matched activity needs to fit to the mode of the pattern activity (as summarized in Table 6.1). A pattern activity with the mode “undefined” can be matched to activities with all modes, a pattern activity with the mode “automated” can be matched to activities with the modes “automated” or “semi-automated”, and a pattern activity with the mode “manually” can be matched to activities with the modes “manually” or “semi-automated”. Finally, a pattern activity with the mode “semi-automated” can be matched to activities with the mode “semi-automated”. This enables specifying patterns for effects that base on the fact that a matched activity is partly automated as well as partly manual.

Table 6.1.: Summary of which Software Manufacture Model pattern activity modes can be matched to which Software Manufacture Model activity modes.

\downarrow can be matched to \rightarrow	semi – automated	automated	manually
<i>undefined</i>	✓	✓	✓
<i>automated</i>	✓	✓	
<i>manually</i>	✓		✓
<i>semi – automated</i>	✓		

The negative structure describes situations, where a pattern activity cannot be matched. When the whole negative structure of a pattern activity is matched to a Software Manufacture Model activity, the included match of the pattern parts that do not belong to the negative structure is invalid. If only a part of the negative structure is matched, while the rest of the pattern activity is matched, the whole pattern activity is matched validly. For example, an artifact matched to *adjusted artifact* in *subsequent adjustment* should not be input of the activity matched to *initial creation*.

A matched activity in a Software Manufacture Model can also contain unmatched links, pins, and artifacts.

A link in an activity that is matched makes the same or a more concrete statement about the relation of two artifacts than the corresponding link in the pattern activity. Only links with the same type of basic relation can be matched to each other.

Artifact links can be matched to similar artifact links in the pattern. Thus, *contains* can be matched to *contains* or *contained by* depending on the direction of the link (i.e. when the start pin of the link is matched to the end pin of the link in the pattern *contained* can only be matched to *contained by* and the other way around). Similarly, *references* can be matched to *references* or *referenced by* depending on the direction of the link.

Content links can be matched to each other following Table 6.2. Links specifying the same content relation can be matched. Further, a link with an *equal* relation can be matched to a pattern link with an *overlap* relation, since equal content is a special form of a content overlap. Consequently, characteristics that hold under the assumption that the content overlaps, hold also when the content is equal.

Table 6.2.: Content relations that can be matched to content relations in a pattern link

\downarrow can be matched to \rightarrow	\cap_c	$=_c$	\neq_c
\cap_c	✓		
$=_c$	✓	✓	
\neq_c			✓

Finally, a link with a set relation can be matched to a pattern link with a basic relation, when the

relation scope is *ForAllForAll* and the relation of the set relation can be matched to the basic relation in the pattern link. Further, a link with a set relation can be matched to a pattern link with a set relation, if the relation of the link can be matched to the relation of the pattern link and if the relation scope can be matched. A pattern link with relation scope *ForAllForAll* can only be matched to a link with the same relation scope. A pattern link with relation scope *ForAllSubset* can be matched to a link with relation scope *ForAllForAll* or *ForAllSubset*. A pattern link with relation scope *ForAllElement* can be matched to a link with each of the relation scopes. Thus, all relations created in the pattern activity will also be created in the activity. For example, pattern link might specify that from each output artifact a reference to one of the input artifacts is created (relation scope *ForAllElement*). A link might specify that from each output artifact a reference to each input artifact is created (relation scope *ForAllForAll*). All references that would be created due to the specification of the pattern link are also created when the specification of the link is fulfilled. Therefore this pattern link can be matched to the link. The other way around, no match would be created. Here the pattern link might specify that from each output artifact a reference to each input artifact is created (relation scope *ForAllForAll*), while the link specifies that from each output artifact a reference to one of the input artifacts is created (relation scope *ForAllElement*). In this situation the pattern link specifies the creation of more references than the link.

A pin has to be matched to a pin of the same mode. Further, a pin that is connected to a single artifact might be substituted by a set pin. In this case, for all artifacts in the set hold the same or more restricted statements about their relations to other artifacts compared to the pattern (i.e. the link refinements hold).

Example 7

Figure 6.26 shows that activity “generate model code” is matched to pattern activity “initial creation”, while “manipulate implementation” is matched to “adjustment”. The “ecore model” is matched to “base artifact” and “interface implementation” is matched to “adjusted artifact”. The pin connected to “adjusted artifact” in “initial creation” is matched to a set pin, which represents that “generate model code” produces a set of “interface implementations”. Corresponding to the link specified in initial creation, the matched link in “generate model code” expresses that all “interface implementation”s overlap in content with the “ecore model”. “Generate model code” contains additional input and output artifacts. “Manipulate implementation” contains in addition a manual enrichment pin, which indicates that “interface implementation” contains afterwards also content that was created manually.

6.3.3. Discussion of Pattern Matching

The above presented match bases on the syntax of the Software Manufacture Model language. This simplicity is motivated by the need to support practitioners who document a pattern. They need to be able to understand what activities will be identified as match. This understanding is a precondition for judging that a pattern can be associated with a specific property (e.g. concerning changeability as it will be used in Section 7.1) that holds for the matched activities, too.

In contrast, the idealized goal of the match is to identify activities that fulfill the semantic of the pattern activities. The matched activities need to describe the same changes to artifacts and their relations as the pattern activities (and can describe additional changes). From a technical viewpoint the exploration whether the semantic of one activity includes the other activity can be quite complex (taken into account that the described changes depend also on the current situation, where an activity is applied to, e.g. the actual number activities in a consumed set).

To tackle this problem the syntactic pattern match was introduced above in Section 6.3.2. This syntactic pattern match is quite simple, but allows the identification of most groups of activities that fulfill a pattern. However, this simplicity is also a compromise, since in some special cases activities remain that might be used to fulfill the semantic of the pattern activity, but will not be identified by the syntactic match. This is illustrated in the following using three examples:

Example 8

First, the syntactic approach does not allow considering transitivity of containment relations. In Figure 6.27 an example is shown, that would not be identified as a match with the presented approach. For

both activities, matches to the pattern activities can be identified. However, the first activity can only be matched when “method” is matched to the role “consumed artifact” or “part”. Artifact “method” will not fulfill one of these two roles in the match of the second pattern activity. Consequently, the activity matches cannot be combined to a match of the whole pattern. In contrast to this result, the activities might - from a semantical viewpoint - be matched to the pattern, since a containment relation is transitive.

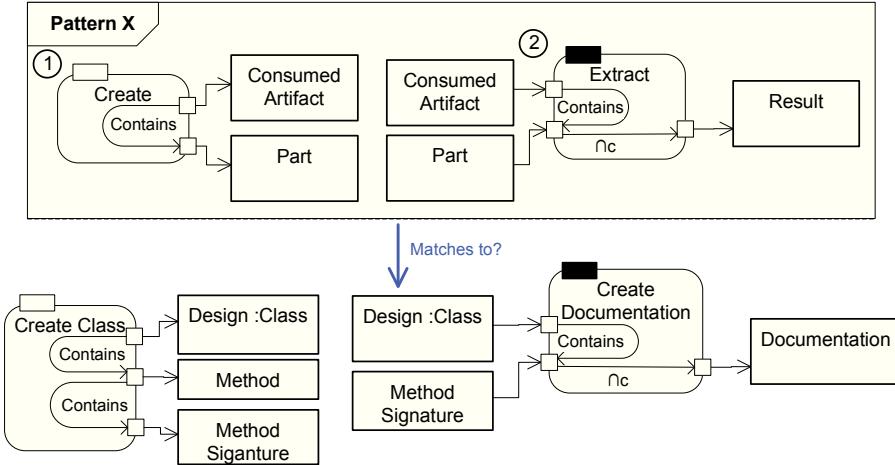


Figure 6.27.: Example of a match that will not be identified, since transitivity of containment relations is not taken into account.

The second example is illustrated in Figure 6.28 (left). Here a similar problem as in the first example is shown: The creation of the reference from the “start” to a certain point in the “target” happens semantically in the activity, too (the “method”, which is contained by of the “design” is referenced by the “documentation”). However, since the method is modeled explicitly, the syntactic approach would not lead to a match.

The third example is illustrated in Figure 6.28 (right). The problem illustrated here is, that two artifact roles in the pattern, “first merged artifact” and “second merged artifact”, need to be matched to the same artifact role “design part” in the Software Manufacture Model. Thus, the syntactic approach would not identify a match, while it is semantically clear that the activity “set merge” can lead to the same (and additional) changes for artifacts and their relations, as pattern activity “merge”¹.

The first two examples represent situations, where a match might be identified, when the *containment* relation would be taken into account. For example, before activities with such a *containment* relation are considered in the syntactic match, they could be enriched. Artifact relations of a contained artifact might be formulated in addition for the containing artifact. In the example in Figure 6.27 this would mean that “create class” would gain an additional link that describes that “design” contains “method signature”. In the example on the left side of Figure 6.28 this enrichment would lead to a link describing that “design” is references by the output version of “documentation”. After such a semantically neutral enrichment or *normalization* of the activities the above described syntactic matching would identify the pattern occurrence.

For the third example the situation is more complex. “Set merge” technically could be applied to reach the same changes as the pattern activity “merge”. However, there might be a reason that the two consumed artifacts in the pattern are modeled as distinct roles. Distinct further activities in the pattern might be formulated for the two roles. This, distinction is lost in the Software Manufacture Model of the “set merge”. Consequently, it is not clear whether the properties associated to the pattern do really hold for the activities in the Software Manufacture Model, too.

Thus, the presented syntactical change is designed in a way that it does capture the first two examples with preparation, only and that it does not capture the last example. Instead, the presented syntactical

¹Note that in the publication [P2] this special case was considered as a valid match.

6. Modeling

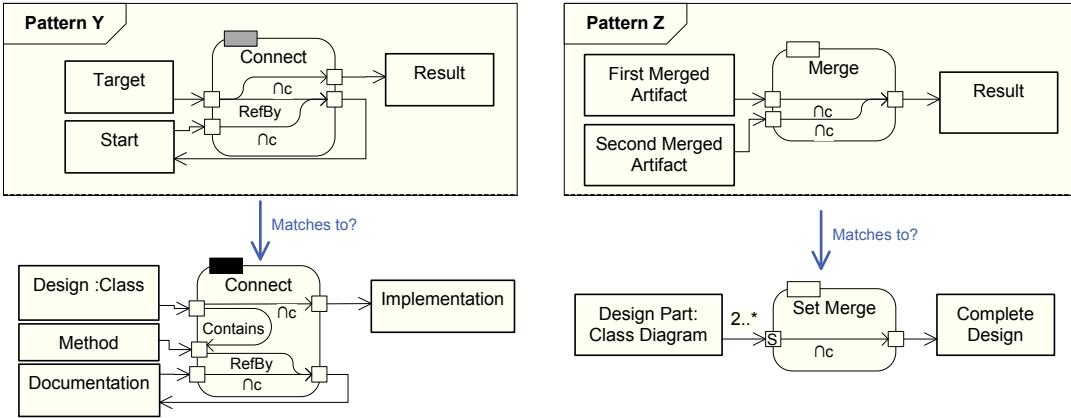


Figure 6.28.: Two example of matches that will not be identified. The identification of the match in the example on the left would require automated consideration of the fact that references that hold for an artifact a , do hold for an artifact b that contains a , too. The identification of the match in the example on the right would require to allow that multiple elements from the pattern are matched to the same element in the activity, in special cases, where non-set pins are matched to set pins.

pattern match enables a simple capturing and identification of structures in Software Manufacture Models. Subsuming, the pattern approach presented here is designed in a way that it can be intuitively used to capture and communicate experience with structures in Software Manufacture Models. A formal analysis is explicitly not the goal of the presented pattern approach. It is not designed for automated identification of all parts in a Software Manufacture Model that fulfill a certain property. Similarly the pattern approach is not designed in a way that the people who document a pattern should proof that the associated properties hold for each match. Both of these later use cases would require a full formal definition of the semantic of Software Manufacture Models as well as expensive proofs that properties of each pattern hold for all matchable Software Manufacture Models.

7. Analysis

Above it was discussed that MDE settings can affect hard changeability concerns and that certain manifestations of MDE traits can lead to constraints of the used software development processes (see Chapter 3). This Chapter introduces analysis techniques that can be used to identify parts of an MDE setting that affect hard changeability concerns as well as process-relevant manifestations of MDE traits.

These techniques conform to two approaches. On the one hand, some concerns and MDE traits are of local character, i.e. they manifest in a combination of a few activities, or even a single activity. Here patterns are used for the analysis. This is done for the analysis how hard changeability concerns are affected as well as for the assessment of the manifestation of the MDE trait *manual information propagation*. On the other hand, some MDE traits can only be identified on the basis of the information what activities in an MDE setting are predecessors or successors of each other. This concerns the MDE traits phases and length of activity chains. The hard changeability concerns can make an application of agile processes more difficult and thus influence the process interrelation as well. Similarly, the length of activity chain which is listed as MDE trait here is also a changeability concern (i.e. *How many activities have to be applied?*). All analysis techniques base on the Software Manufacture Model language (and Software Manufacture Model pattern language).

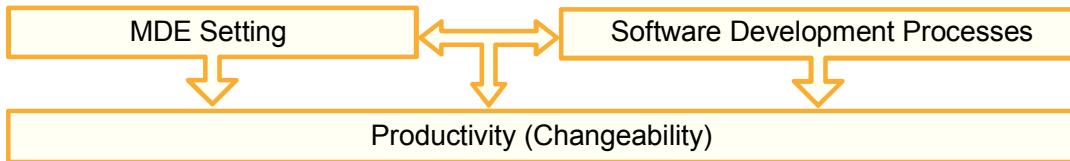


Figure 7.1.: Overview of this chapter: analysis supports concerns an MDE setting's impact on changeability and process interrelation.

In this Chapter, first patterns are presented, that allow analyzing how hard changeability concerns are affected, as summarized in Figure 7.1. Further, simple analysis techniques for the assessment of the manifestations of MDE traits are introduced. The Chapter closed with a discussion how and for what goals the analysis techniques can be applied.

This chapter is partially based on [P2].

7.1. Analysis of Changeability

In context of the first study (see Section 4), four proto-patterns for the analyses of changeability concerns have been identified. In this Section the chosen structure for the description of the patterns is introduced. Then it is described how the proto-patterns have been identified. An overview of the identified proto-patterns is given. Afterwards, the proto-patterns are introduced.

7.1.1. Description Structure of Patterns

In Section 6.3 the Software Manufacture Model pattern language was introduced. However, a pattern does not only consist of a structure in a model, but also of the documentation of further aspects, such as the consequences associated to the pattern (i.e. the knowledge that is associated with the specific structure). In the following the structure that shall be used for the systematic description of the changeability related properties of Software Manufacture Model patterns and Software Manufacture Model anti-patterns is introduced.

This structure for the description of Software Manufacture Model patterns and Software Manufacture Model anti-patterns (shown in Table 7.1) is oriented on Gamma et al. [86]. In both cases the pattern is first described structurally (supported by the presentation of the structure in Software Manufacture Model pattern notation). Then it is described how the Software Manufacture Model pattern affects

Table 7.1.: Structures of Software Manufacture Model anti-pattern and Software Manufacture Model pattern.

<i>Anti-Pattern Structure</i>	<i>Pattern Structure</i>
1. Description	1. Description
2. Benefits & Risks	2. Benefits & Risks
3. Resolution Strategy	3. Implementation
4. Origin	4. Applicability
5. Occurrences	5. Occurrences
6. Related Patterns	6. Related Patterns

changeability concerns (“benefits and risks”). When a Software Manufacture Model pattern is matched to a set of activities, the described influence holds for the activities and artifacts that are matched. For example, if a Software Manufacture Model pattern describes that an activity preserves certain parts of the content of an input artifact this does not necessarily hold for other input artifacts.

After the description of benefits and risks, implementation approaches are provided, followed by information when the pattern can be applied. Alternatively, for Software Manufacture Model anti-patterns resolution strategies are provided, followed by a description when associated risks are manifested (“origin”). Finally, known occurrences of a pattern are listed and relations to other patterns are described.

7.1.2. Identification of Proto-Pattern

Before the patterns are presented, it is described how the two Software Manufacture Model proto-anti-patterns and two Software Manufacture Model proto-patterns were identified. As introduced in Section 2.4, a pattern describes a structure that can be associated with positive properties and is often used to communicate an experienced solution to a common problem. In contrast an anti-pattern can be used to describe situations that should be prevented or that can be used as hints to identify risks. The term proto-pattern (or proto-anti-pattern, respectively) is used to describe that a pattern still has the status of a candidate. Thus, a proto-pattern already has the structure of a pattern and includes all associated information. However, the term pattern usually implies that at least three occurrences in practice are documented. Until this is the case a pattern has the status of a proto-pattern. This section introduces proto-pattern, since the identification of these pattern based on the SAP case studies, only. Whether the different proto-patterns have the potential to get the status of a pattern shall be discussed later on in Chapter 9.

The identification of the proto-patterns started with solutions that have been used in the SAP case studies (from the first study) to prevent that hard changeability concerns are affected. Based on these solutions, the concepts that underlie the Software Manufacture Model language were used for a theoretical consideration how parts of a Software Manufacture Model have to look like to affect the hard changeability concerns. This consideration was done by altering activity structures. For these activity structures change scenarios (i.e. a definition on which artifacts a change is introduced) were formulated. These scenarios were then investigated to identify whether one of the changeability concerns is affected. Although this approach was, only applied for a small set of variations, it was possible to identify some relevant candidates for Software Manufacture Model anti-patterns (Software Manufacture Model proto-anti-patterns).

Afterwards, the SAP case studies were revised with focus on the Software Manufacture Model proto-anti-patterns. Although it was not expected, it was possible to identify occurrences.

The processes captured in the SAP case studies are used in practice. Furthermore, it can be expected that the examples, which were chosen by practitioners of a company to be presented to external observers, are experienced as successful. Consequently, it was not surprising that the identified occurrences of the Software Manufacture Model proto-anti-patterns do not lead to failure of the projects. This is due to mitigating factors such as low probabilities that the critical change scenarios happen (i.e. where a good changeability is not required, the occurrence of a negative influence on changeability can be tolerated).

Finally, the identified Software Manufacture Model proto-patterns and Software Manufacture Model proto-anti-pattern were enriched with information about their influence on changeability concerns.

All in all, this process led to the identification of two Software Manufacture Model proto-patterns and two Software Manufacture Model proto-anti-patterns. In Figure 7.2 an overview about the identified proto-patterns is given. Software Manufacture Model proto-pattern *split manufacture* as well as Software Manufacture Model proto-anti-pattern *subsequent adjustment* affect the hard changeability concern *unexpected loss or preservation of content*. Artifact *split manufacture* can be used as solution for *subsequent adjustment*. Software Manufacture Model proto-pattern *anchor* affects hard changeability concern *unexpected loss or preservation of references*. Finally, Software Manufacture Model proto-anti-pattern *creation dependence* describes a situation, where different changes are coupled to each other, which can lead to the situation that more changes than necessary can cause the re-execution of activities. As a consequence, although it does not directly affect a hard changeability concern, *creation dependence* is an amplifier for negative effects on changeability, such as the effects caused by *subsequent adjustment*. *Anchor* can be used to reduce this amplifying effect of *creation dependence*.

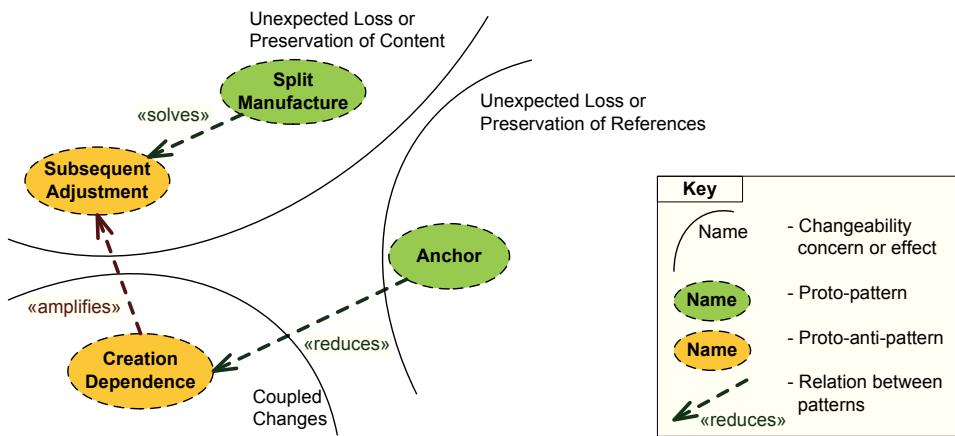


Figure 7.2.: Overview of identified proto-patterns and their interrelation

7.1.3. Software Manufacture Model Proto-Pattern Subsequent Adjustment

Subsequent adjustment is a Software Manufacture Model proto-anti-pattern associated to changeability concern *unexpected loss of content*.

Description The activity *initial creation* (Fig. 7.3) automatically creates *adjusted artifact* and fills it with content on the basis of *base artifact* (i.e. both artifacts overlap in content after the execution). No already existing version of *adjusted artifact* is considered. The activity *adjustment* adds further content to *adjusted artifact* manually.

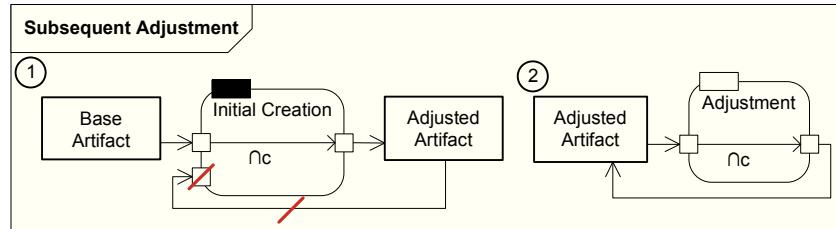


Figure 7.3.: Software Manufacture Model pattern *subsequent adjustment*

Benefits & Risks After the first execution (*initial creation* followed by *adjustment*) the *adjusted artifact* contains content that was added in both activities. Due to changed requirements, *base artifact* might be

7. Analysis

changed manually or by other automated activities. To propagate the change *initial creation* is executed again, which overwrites content of *adjusted artifact* completely. This leads to a loss of content that was added in the manual activity *adjustment*.

Developers might choose a workaround and apply the necessary change not to the *base artifact*, but directly to *adjusted artifact*. The resulting inconsistency of artifacts is a source for errors and increases the training hurdle for new developers that join the team. Further, benefits which are provided by the automation of the activity *initial creation* are nearly lost. For example, usually it is less error prone to change the more abstract *base artifact* than changing *adjusted artifact*. With the described workaround this benefit cannot be used. Alternatively, developers might copy-and-paste the parts that they have changed in *adjusted artifact* during the first execution of activity *adjustment* or developers might redo their work for each change. Both variants are annoying for developers and cost time and thus productivity.

Resolution Strategy First, the probability that *base artifact* will be changed – such that *initial creation* is executed again – should be weighted. When, no changes will be applied to *base artifact* it is not necessary to optimize the MDE setting, such that changes of this artifact are supported. For example, *base artifact* might be completely reused from another project or can be part of a library.

In a second step, it should be considered whether the parts of *adjusted artifact* that are touched by the two activities can be separated clearly. If this is possible, an implementation of Software Manufacture Model proto-pattern *split manufacture* can be applied to solve the problem. Alternatively, the separation on artifact level might be changed, e.g. by applying the generation gap pattern [78]. Then *adjusted artifact* can be split into two artifacts which reference each other.

If a clear separation is not possible, the structure of the artifact should be changed to separate the concerns affected by the different activities more strongly. Solutions may provide blueprints for the structure of the instances of *adjusted artifact*, change the language of *adjusted artifact*, or even apply a complete redesign of the way how information is split over different artifacts. A short term solution is changing activity *adjustment* in a way that the changes are not applied to *adjusted artifact* but to a copy of *adjusted artifact*. Thus, manually added content will not be overwritten in case of a new application of activity *initial creation* and might be copied in some other way to *adjusted artifact*.

Origin As mentioned above, in some cases Software Manufacture Model proto-anti-pattern *subsequent adjustment* can exist within a Software Manufacture Model without causing harm. For example, the probability that *base artifact* will change can be very low or there might be no need that product parts based on *adjusted artifact* survive changes (see the example in “occurrence” below). However, due to adaption of the product portfolio these conditions can change over time and enable the Software Manufacture Model proto-anti-pattern to cause harm.

A further possible origin of *subsequent adjustment* is a spread of development over different departments, where *adjusted artifact* is handed over between these departments. The first department might decide to generate *adjusted artifact* automatically, while the second department still assumes that *adjusted artifact* can be enriched with information.

Occurrence In the SAP case studies, one occurrence of Software Manufacture Model proto-anti-pattern *subsequent adjustment* was identified. In this example, the activity in the role of *initial creation* automatically initializes dummy data for the creation of a dummy service that is used during development for a prototype for a user interface. In this case, the presence of changeability concern causes no harm, as there is no need that the dummy service survives the whole live cycle of the software.

Related Patterns *Split manufacture* can be used to resolve the problems caused by *subsequent adjustment*.

Example 9

In Figure 7.4 an example extract of a Software Manufacture Model is shown. This example includes three activities: “Modify configuration” is a manual activity to modify a “configuration” file. “Generate implementation” consumes a “service description” and generates a “class diagram” and “java code”.

Further, a reference from the “java code” to the “configuration” is created. Finally, “implement method” is a manual activity, where the “java code” is enriched on the basis of the “specification”.

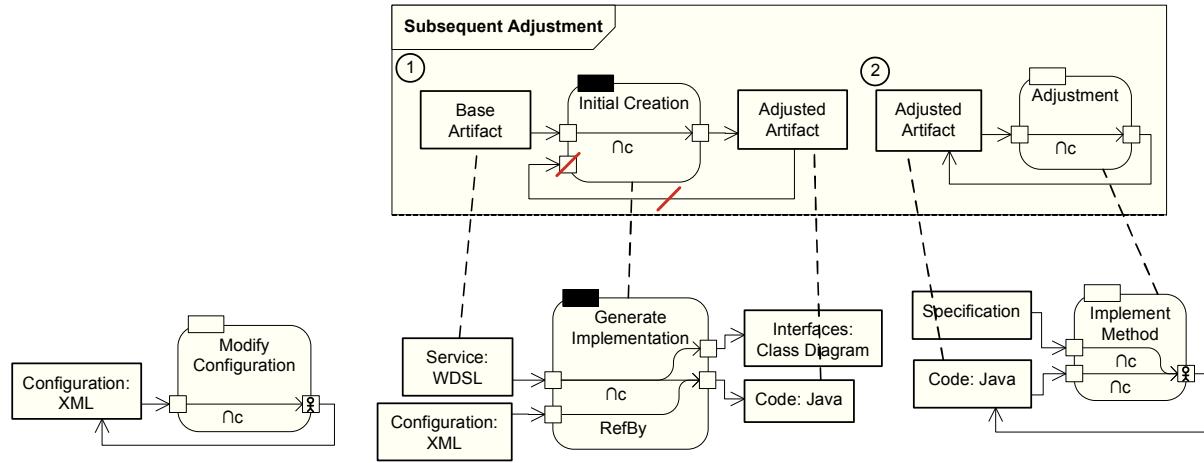


Figure 7.4.: Example Software Manufacture Model extract for creation of a web service implementation. Pattern *subsequent adjustment* is matched.

An example for the documents that might already be created within a project is shown in Figure 7.5. Here, a “service description” is available¹ as well as a textual specification and a configuration file.

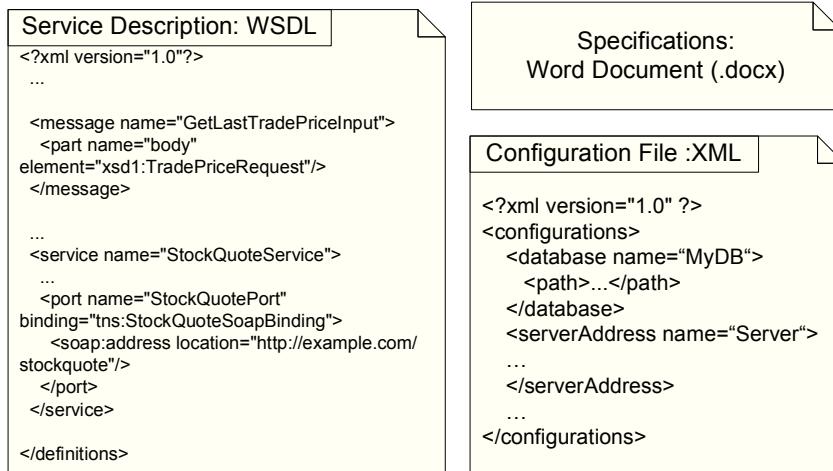


Figure 7.5.: Initial set of artifacts available: a web service description, a specification, and a configuration file.

An initial execution of activity “generate implementation” (which matches to pattern activity initial creation) might lead to the “class diagram” and the “java class” shown in Figure 7.6. Since the generated method is not yet implemented the manual activity “implement method” (which matches to pattern activity adjustment) is executed, which leads to the result illustrated in Figure 7.7(a).

Now a change scenario is entered, where artifact “service description” is extended such that other functionality is provided (illustrated in Figure 7.7(b)).

In order to propagate this change, such that the new functionality that is specified in the service description is also implemented, the activity “generate implementation” needs to be executed a second time. This result of this execution is shown in Figure 7.8. It can be seen that the whole artifact is generated completely new (since the model activity is not able to consider existing versions

¹The example is oriented on the example WSDL from <http://cs.au.dk/~amoeller/www/webservices/wsdlexample.html> (last access at November 9th, 2013), which is shown in Appendix E.

7. Analysis

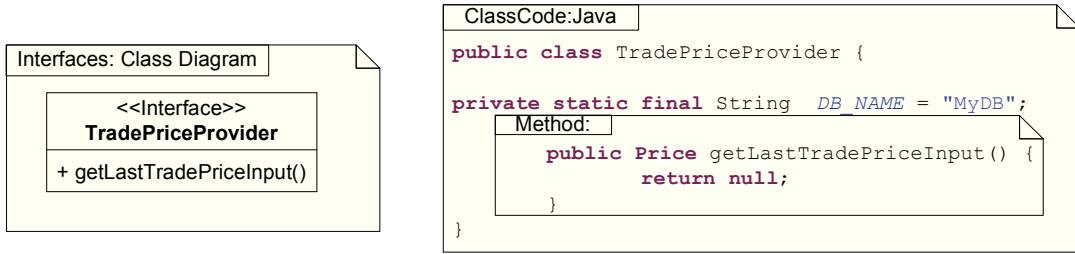


Figure 7.6.: Artifacts created by activity “*generate implementation*” from Figure 7.4 on the basis of the artifacts shown in Figure 7.5: a class diagram of the interfaces as well as a source code frame.

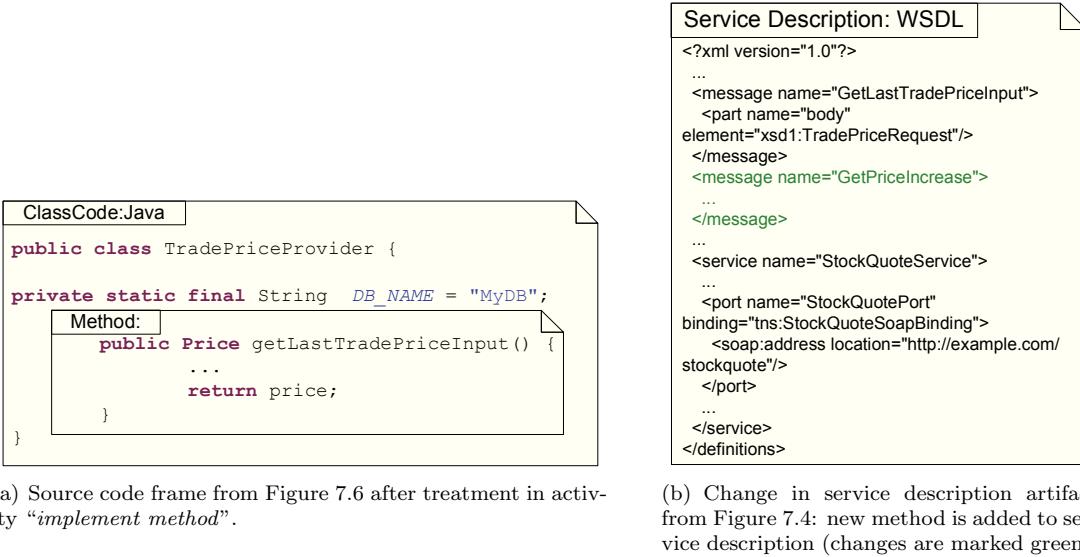


Figure 7.7.: Source code artifact after execution of activity “*implement method*” (left) and service description after change (right).

of the generated artifacts). As a consequence the (potentially extensive) implementation of method “*getLastTradePriceInput*” is lost. This example shows how pattern subsequent adjustment can lead to unexpected loss of content, when changes are applied.

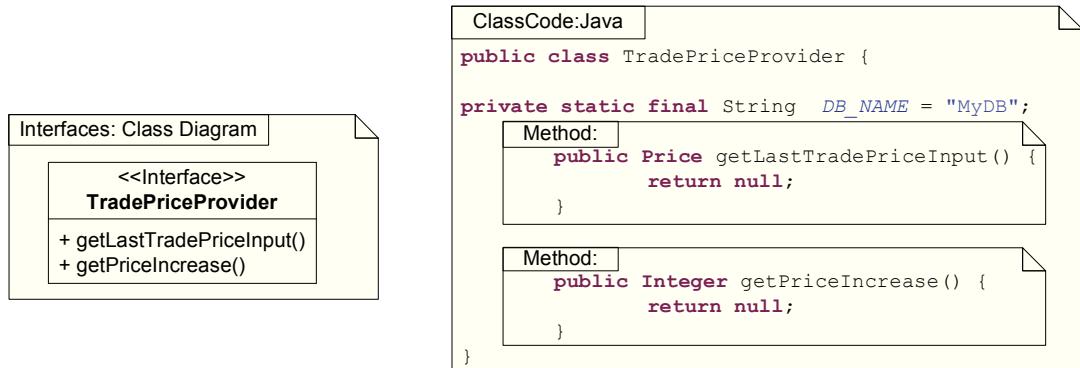


Figure 7.8.: Class diagram of interfaces and code frame after new execution of activity “*generate implementation*” on the basis of changed service description from Figure 7.7(b).

7.1.4. Software Manufacture Model Proto-Pattern Creation Dependence

Creation dependence is a Software Manufacture Model proto-anti-pattern that is an amplifier for existing changeability concerns.

Description As shown in Fig. 7.9 the automated activity *depending creation* creates *depending artifact* with a reference to the input *necessary artifact*. An already existing version of *depending artifact* is not considered as input of *depending creation*. In some cases, the process might contain alternative activities that allow restoring the reference without recreating the whole *depending artifact*. Such an activity might define more restricted changes to *depending artifact* and help to prevent the below described consequences.

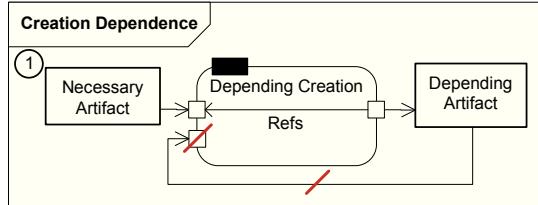


Figure 7.9.: Software Manufacture Model pattern *creation dependence*

Benefits & Risks The general risk in coupling of changes is that other problems with changeability, e.g. caused by occurrences of subsequent adjustment, are intensified. Coupling changes, which would not necessarily trigger further activities, to changes, which have successors that are associated to changeability problems, enhances the number of triggers for these changeability problems. Applying *depending creation* couples the creation of the artifact *depending artifact* (and thus creation of its content) to the creation of a reference to *necessary artifact*. Consequently, all activities that manipulate *necessary artifact* are additional predecessors for the creation of *depending artifact*.

Due to changing requirements, e.g. within agile development, *necessary artifact* might be changed. As a consequence, the reference has to be restored, when the reference's target is deleted during the change of *necessary artifact*. However, the reapplication of *depending creation* leads to new creation of *depending artifact*. Then, changeability problems that are associated to activities that are triggered by a change of *depending artifact* occur unnecessarily often. Developers might use manual workarounds for restoring the references.

Resolution Strategy First, it should be weighted whether there is a changeability problem that can be amplified by the *creation dependence* (i.e. whether successor activities of *depending creation* strongly affect a changeability concern). In addition it should be weighted whether it is probable that *necessary artifact* will be changed. For example, if an artifact is used as library or as an interface the probability that this artifact is changed might be very low. In case that either no changeability problem is amplified or that a change in *necessary artifact* is not probable, it can be decided to leave the situation as is.

Otherwise, there are two types of resolution approaches: limiting change propagation or splitting up the reference creation from content creation. The first approach is to limit the propagation of the change from the *necessary artifact* to the *depending artifact*, by preventing that the reference gets lost, when *necessary artifact* is changed. For example, the *anchor* Software Manufacture Model proto-pattern might be used for that reason. If it is not possible to specify an appropriate *anchor*, it is necessary to identify possible changes on *necessary artifact* and restrict them to the really necessary ones.

Alternatively, an alternative activity should be introduced, that allows updating the references in *depending artifact* without complete recreation of the content. Further, it should be localized which parts of the *depending artifact* have to be changed to restore a reference. Another solution is to adapt the automation of *creation dependence* such that it changes the parts of *depending artifact* that reference *necessary artifact* only, instead of completely recreating the *depending artifact*. This might be used to design the successor activities such that they do not need to be reapplied in case of these local changes in *depending artifact*.

7. Analysis

However, it is possible that there is a strong coupling between the content of *depending artifact* and the references to *necessary artifact*. For example, the definition of a layout (as *depending artifact*) might depend strongly on the question, which parts of a complex data structure (as *necessary artifact*) are referenced. The fact that the creation of the whole *depending artifact* is coupled to the reference creation can be a hint for a strong coupling of reference change and content change.

For a resolution of such a strong coupling of reference change and content change it has to be considered whether it is possible to split both changes. This might not be a simple task as underlying paradigms might have to be switched. In the above mentioned example of the data structure and the layout paradigms might be used that are similar to style sheets that are decoupled from the concrete content of an XML file. If the decoupling of reference change and content change succeeds, a solution can be found, where updating references does not trigger changes in the content.

Origin Depending on different factors, e.g. when no changeability problems are associated to successors of *creation dependence* or if there is a low probability that *necessary artifact* changes, it is possible that Software Manufacture Model proto-anti-pattern *creation dependence* occurs without causing harm.

However, changes to requirements on usually built software can influence such probabilities or cause changes to the Software Manufacture Model. This can activate negative properties of Software Manufacture Model proto-anti-pattern *creation dependency*. In the mentioned example, the need for a customizable data structure can arise. Thus, it becomes probable that *necessary artifact* “data structure” changes regularly, with all consequences for the need to change *depending artifact* “layout”.

Alternatively, changeability problems with successors might be introduced in the MDE setting. For example, the creation dependency might be part of an automated transformation chain where the product is not touched manually. However, an extension of the feature range for the software that is built by the company might not be supported by the DSL that used as generation basis for *depending artifact* within *depending creation* (i.e. as additional input of *depending creation*). This can cause developers to manipulate *depending artifact* after generation - with the consequence that a *subsequent adjustment* proto-anti-pattern is introduced. In this scenario, the already existing *creation dependency* can amplify the newly introduced changeability problem.

Occurrences Creation dependency was found in three of the SAP case studies. All occurrences seem to profit from mitigating factors such as a low probability that *necessary artifact* changes. In one case the automated creation dependency was not followed by manual activities, i.e. there were no changeability problems that could be amplified.

Related Patterns Anchor might be applicable to prevent a loss of reference and the propagation of changes from *necessary artifact* to *depending artifact*.

Example 10

Figure 7.10 highlights that in the example that was introduced to explain pattern subsequent adjustment in Figure 7.4 a match of pattern depending creation (on activity “generate implementation”) can be identified, too. The “configuration” can be matched to pattern artifact necessary artifact and “code” can be matched to pattern artifact depending artifact. For this example the starting situation that was already introduced in Figure 7.5 can be reconsidered. Again the initial execution of both activities, “generate implementation” and “implement method”, can lead to the artifacts shown in Figure 7.11. Now a change scenario is entered, where the “configuration” is changed such that the name of the database changes (as shown in Figure 7.12(a)). As a consequence, activity “generate implementation” needs to be executed a second time in order to restore the reference from artifact “class code” to the identifier for the database within the configuration. Although this restores the reference (as shown in Figure 7.13) the problem that is caused by the occurrence of pattern subsequent adjustment manifests: the implementation of the method “getLastTradePriceInput” is lost.

Even worse, also an alternative change scenario, where no database-related configurations are changed (e.g. as shown in Figure 7.12(b)), can lead to the new execution of “generate implementation”. This can happen when an automated model management system detects that the configuration file was changed and triggers or proposes the new execution of generate implementation. Thus, content might get lost, even if no inconsistency between the artifacts existed. This holds especially when the decision is made

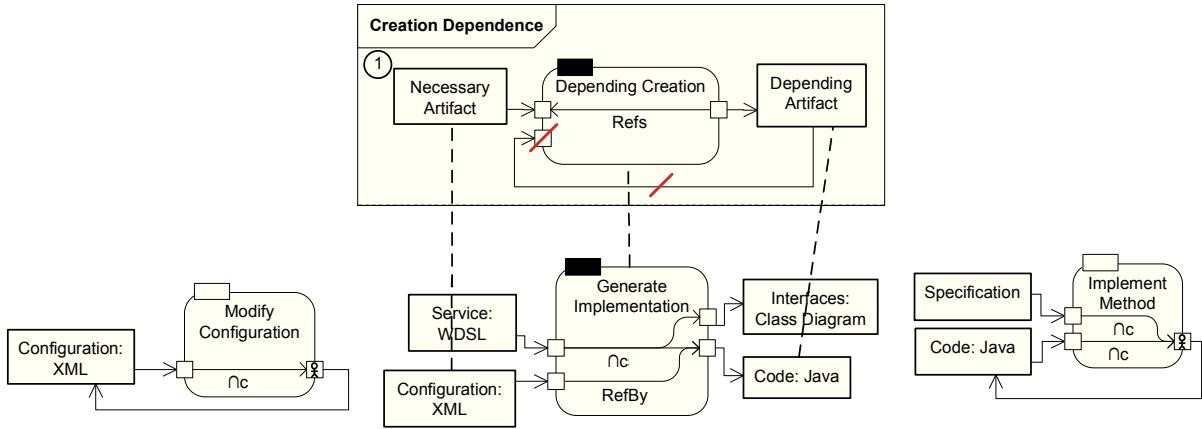


Figure 7.10.: Example Software Manufacture Model extract for creation of a web service implementation. Pattern *creation dependence* is matched.

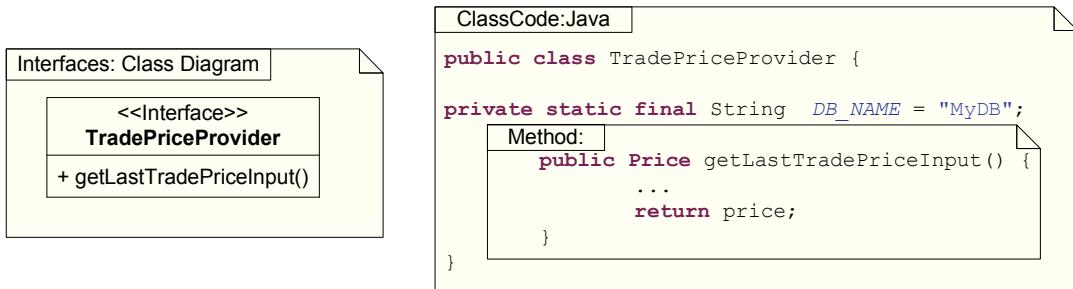


Figure 7.11.: Artifacts created by subsequent application of activities “*generate implementation*” and “*implement method*” from Figure 7.10 on the basis of the artifacts shown in Figure 7.5: a class diagram of the interfaces as well as a source code frame with an implementation of the method.

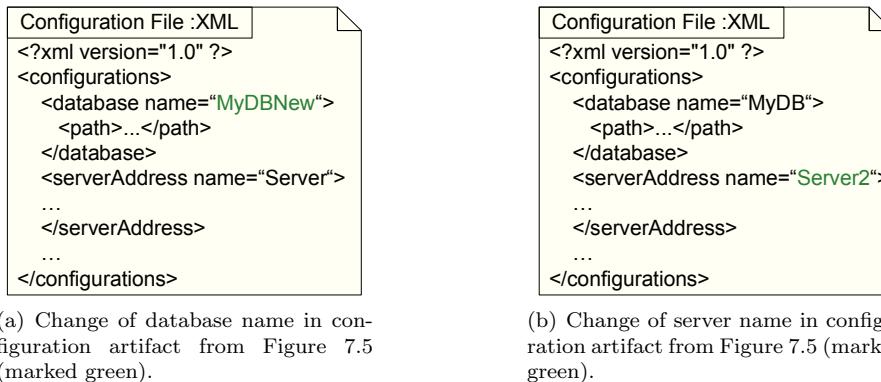


Figure 7.12.: Two example changes on the configuration artifact.

within a build server that automatically executes an automated activity when the input artifacts were modified. In an MDE setting, where the reference creation would not be part of the generation step (e.g. it might be created manually), activity “*generate implementation*” would not be a successor of activity “*modify configuration*”. Thus, “*modify configuration*” does not need to be an additional trigger for “*generate implementation*”. This example shows, how an occurrence of pattern creation dependence can lead to additional triggers for the new execution of activities and thus makes an existing problem with changeability worse.

7. Analysis

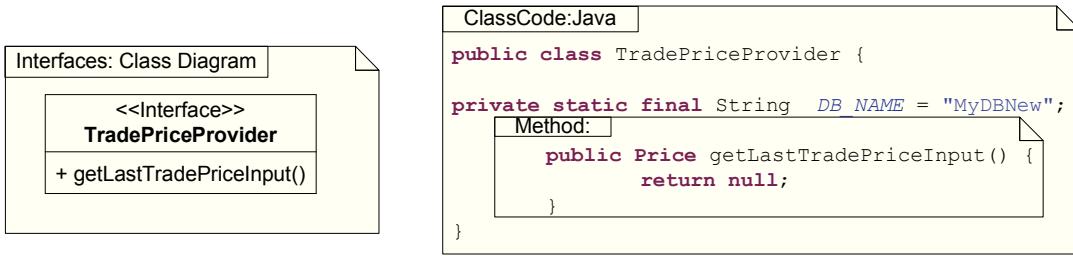


Figure 7.13.: Class diagram of interfaces and code frame after new application of *generate implementation* in order to restore reference to data base configuration in changed configuration file from Figure 7.12(a).

7.1.5. Software Manufacture Model Proto-Pattern Split Manufacture

Split manufacture is a Software Manufacture Model proto-pattern that is associated to the changeability concern *unexpected loss of content*. Techniques to implement this Software Manufacture Model proto-pattern are already known in literature. Nonetheless, this Software Manufacture Model proto-pattern is described here, as the ability to identify it is important for analysis and improvement of Software Manufacture Models.

Description *Split manufacture* consists of two activities: *regenerate* and *manual completion* (see Fig. 7.14). *Regenerate* modifies content of *modified artifact*, but preserves *artifact detail*, which is part of *modified artifact* before and after *regenerate* was executed. *Manual completion* modifies *artifact detail*.

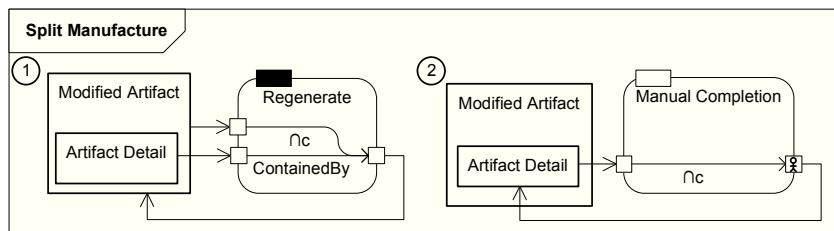


Figure 7.14.: Software Manufacture Model pattern *split manufacture*

Benefits & Risks The content that was added in *manual completion* is preserved when *regenerate* is executed in case of a change. This saves effort and improves changeability. Further, the Software Manufacture Model proto-pattern enforces the developer of the Software Manufacture Model to make explicit which parts of the artifact are touched automatically or manually.

Implementation The implementation of *split manufacture* requires that the technology used for the automation of activity *regenerate* is able to preserve artifact parts, e.g. a technology that supports protected regions. In addition, it has to be defined what parts of the artifact belong to the *artifact detail*. Finally, developer guidelines can ensure that only the *artifact details* are touched in *manual completion*.

Applicability *Split manufacture* is applicable when the language used as basis for the generation cannot express all needed functionality.

Occurrences Implementations of this proto-pattern are broadly known, such as in round-trip engineering or protected regions, e.g. in Xpand². In addition, one occurrence in the SAP case studies was identified.

²<http://wiki.eclipse.org/Xpand> (last access at November 9th, 2013)

Related Patterns Split manufacture can be used to resolve Software Manufacture Model proto-anti-pattern subsequent adjustment.

Example 11

As an example for split manufacture a variant of the example from Figure 7.4 is used here. The pattern is applied as solution of the occurrence of subsequent adjustment in Figure 7.4. Thus, activity “generate implementation” is matched to regenerate, since it is now implemented in a way that an already existing version of the “code” is considered as input. Existing input versions of “method”s are preserved and are contained by the output version of “code” afterwards. Further, activity “implement method” is now modeled, such that it is more explicit that only the “method”s are adapted.

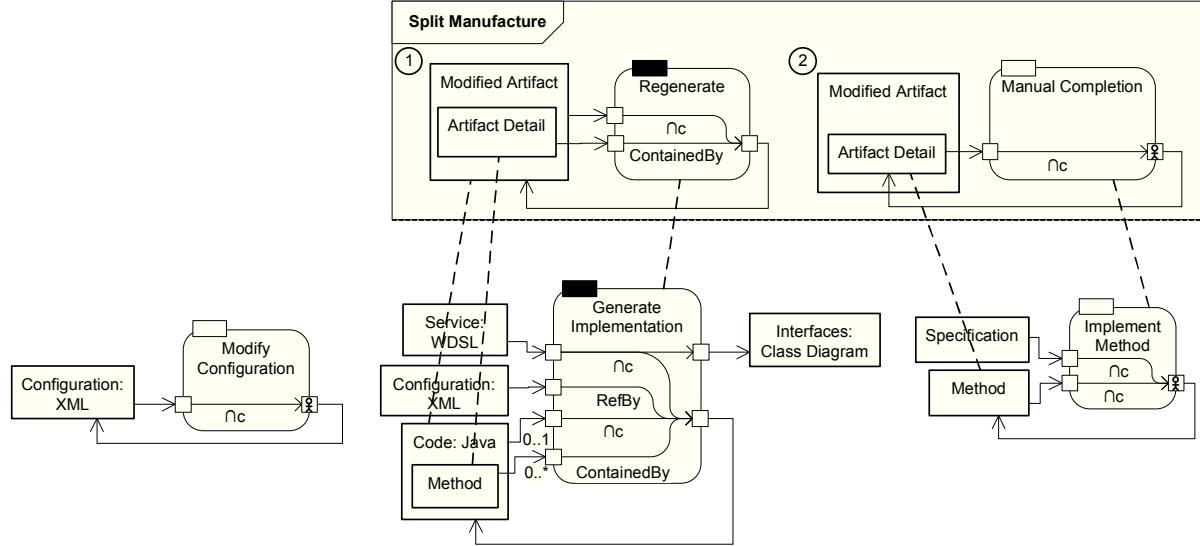


Figure 7.15.: Adapted example Software Manufacture Model extract for creation of a web service implementation. In contrast to version from Figure 7.4, pattern *subsequent adjustment* is not more matched. Instead pattern *split manufacture* is matched.

Now a change in the service description (e.g. as in Figure 7.7(b)) in a situation where the activities shown in Figures 7.5 and 7.11 are available, leads to a new execution of “generate implementation”, too. However, as a result the content of method “getLastTradePriceInput” is preserved (as shown in Figure 7.16).

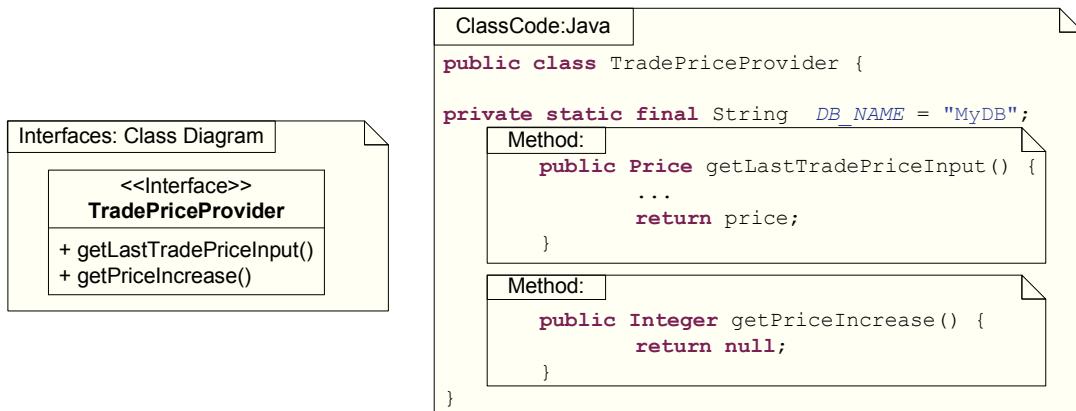


Figure 7.16.: Class diagram of interfaces and code frame after new application of *generate implementation* in order to propagate change of service description that is illustrated in Figure 7.7(b) (previous versions of code file and class diagram were available as shown in Figure 7.11).

7.1.6. Software Manufacture Model Proto-Pattern Anchor

Anchor is a Software Manufacture Model proto-pattern and is associated to changeability concern *unexpected loss of references*.

Description As shown in Fig. 7.17, activity *dock on anchor* creates or changes *tying artifact*, so that *tying artifact* owns a reference to *anchor*. Finally, activity *modify bound artifact* is used to change *bound artifact*. Input *anchor* is passed to the output version of *bound artifact* unchanged.

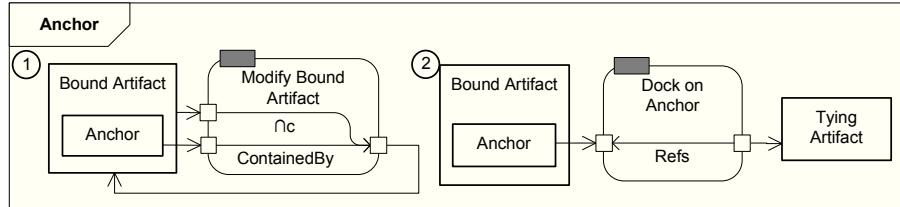


Figure 7.17.: Software Manufacture Model pattern anchor

Benefits & Risks *Anchor* allows applying changes to *bound artifact* without worrying that references from *tying artifact* to *bound artifact* get dangling. This prevents further that a change in one of the artifacts enforces a change in the other artifact. Subsuming, the change becomes local for the developer, which eases the change and reduces the probability of errors that are caused by forgotten references.

Implementation For an implementation an appropriate kind of *anchors* has to be chosen. Ideal *anchors* are not probable to change or change only rarely. To ease the identification of *anchors* during the project or the implementation of automated variants of *dock on anchor* and *modify bound artifact*, *anchors* can be defined based on the structure of the *bound artifact*, e.g. an *anchor* can be a method signature.

If activities *dock on anchor* and *modify bound artifact* are manually, they can be implemented using developer guidelines to ensure that *anchor* is not changed and can be used as target for a reference. If *modify bound artifact* is an automated activity, a technique should be used that can deal with protected regions. The *anchor* is located in the protected region. An automated implementation of *dock on anchor* has to respect that references only target at *anchors*.

Applicability *Anchor* is applicable when consistent references between two artifacts are required and changing the referenced artifact should be supported.

Occurrences One occurrence of *anchor* was identified in the SAP case studies, where it was used to define extensions for customizing a user interface.

Related Patterns *Anchor* can prevent lost references to resolve problems of *creation dependence*.

Example 12

Figure 7.18 shows an adapted version of the example shown in Figure 7.10, where the occurrence of the creation dependence is shown. The creation dependence and the “subsequent adjustment” still exist in this version of the example. However, the *anchor* pattern is applied, too. Activity “generate implementation” is explicitly build in a way that it consumes the “database tag”, only, - independent of the rest of artifact “configuration”. Thus, this activity is only triggered, when the database tag is manipulated.

Further, activity “modify configuration” now reflects a developer convention not to manipulate the database tag (i.e. in context of this activity the database tag is preserved when the configuration is manipulated). This first important aspect of this occurrence of pattern *anchor* is that the reference between “code” and “configuration” will not be lost, since the target of the reference is a stable part of the configuration. The second aspect is that the effect of the creation dependence is reduced. Thus, “generate

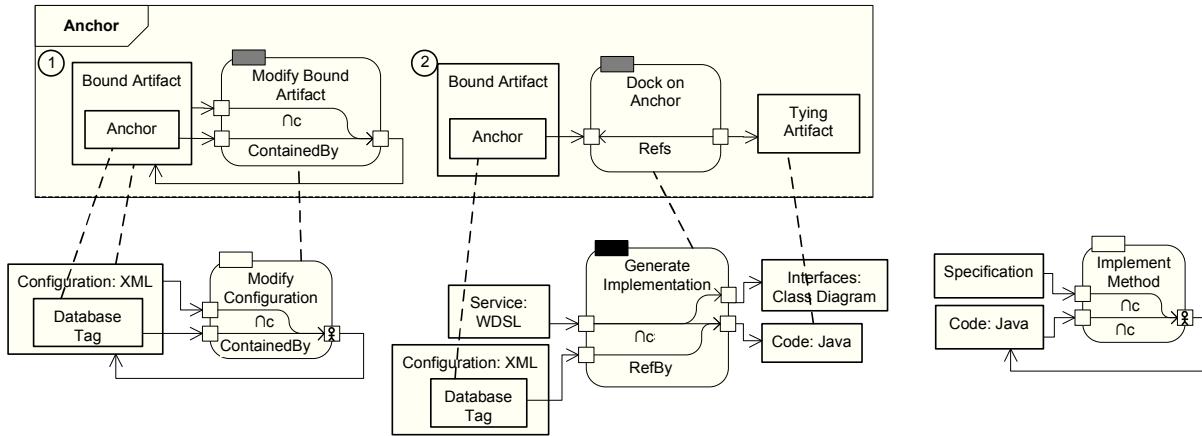


Figure 7.18.: Adapted example Software Manufacture Model extract for creation of a web service implementation. In addition to the pattern matches on the version from Figure 7.10, pattern *anchor* is matched, too.

implementation” is no longer a successor of “modify configuration”. As a consequence changes like the one shown in Figure 7.12(b) will no longer trigger the new execution of “generate implementation”. A change that is performed to the database tag (as in Figure 7.12(a)) in context of other activities can still trigger the new execution of “generate implementation” followed by the problems of the “subsequent adjustment”. Therefore, the anchor pattern does not solve, but only reduces the effects of the creation dependence.

7.2. Analysis of Process Interrelation

In the following, three analysis techniques are introduced that allow assessing the manifestations of the three MDE traits that are introduced in Section 3.3.

7.2.1. Analysis Method for Phases and Synchronization Points

Global synchronization points can easily be identified on the basis of a Software Manufacture Models by considering predecessors and successors of all activities. A global synchronization point is an activity for which all other activities in the MDE setting are either successors or predecessors. Thus, there are no activities that can be executed parallel to the *global synchronization point*.

Correspondingly, it might be interesting to identify local synchronization points, which can be identified similarly, by considering a local focus, which is a subset of the activities in an MDE setting, only. This is useful, when the work done in an MDE setting is split over different sub-teams. Further an MDE setting might include alternative solutions to implement the same set of artifacts. These alternatives would not occur as predecessors or successors of a potential synchronization point, even if they are never executed in parallel, too. For this reason it might be useful to consider a subset of artifacts, only.

Following the definition, synchronization points can have any degree of automation (manual, semi-automated, or automated). It is assumed that a modeled manual activity is done by a single developer. In practice, a single manual activity might be split up between developers, too. However, this can be associated with its own challenges for synchronization, especially, when the same artifact is accessed by two developers. Thus, even when work in a manual synchronization point is split this means special effort for synchronization of work.

The identification of a (local) synchronization point requires information which activities have no order concerning the activity under consideration. Consequently, it is more difficult to identify synchronization points in a model that contains activities for very different concerns (i.e. when the model has no clear focus). In order to simplify the identification of synchronization points, following modeling convention should be fulfilled before this analysis is performed on a Software Manufacture Model:

7. Analysis

MC3 A Software Manufacture Model that is used for the analysis of synchronization points should have a clear focus. When a Software Manufacture Model contains activities for the creation of an additional product that is not in focus or activities that are seldom used to manipulate the MDE setting itself, these activities might be removed to improve clarity.

To more directly regard to the consequences of *global synchronization points* the resulting *number of phases* in the MDE setting can be counted. A *phase* is a non-empty set of all activities between two synchronization points or before or after all synchronization points. Thus, all activities in a phase are successor of one synchronization point and predecessor of another synchronization point, or all activities in a phase are successor of one synchronization point, which has no other synchronization point as successor, or all activities in a phase are predecessor of one synchronization point, which has no other synchronization point as predecessor. A *phase* includes no synchronization points. Correspondingly, a local phase includes no local synchronization points within the respective local focus. An exception from that are activities that are extensive manual tasks that can potentially be split among developers. This can be coding activities or modeling activities when the model includes multiple views. When such activities are synchronization points they are counted as *miniphases*. In case that a miniphase directly follows or is followed by a phase or miniphase, they are counted as one phase. However, the interrelation between the number of synchronization points and phases is complex. The *number of phases* is not necessarily equal to the number of global synchronization points plus one. For example, an exception occurs when in a chain of successive activities all are synchronization points (i.e. multiple activities are executed in a row before parallel work is possible again). While a single synchronization point might split a setting into two phases, other settings with multiple synchronization points in the main focus have a single phase only. The location of the synchronization point plays an important role for the actual impact. Synchronization points at the start or at the very end of the setting, introduce no new forces for the team (as a team is usually synchronized at the start of the development and has to synchronize at the end). Thus, although single synchronization points can have important impacts, the pure number of synchronization points is not as expressive as the number of phases, due to the above discussed exceptions.

Consequently, the *number of phases* might give a more direct hint on the question how often developers possibly need to wait for each other before a synchronization point, compared to the number of synchronization points itself.

Example 13

Figure 7.19 shows an excerpt from BO. The shown excerpt includes a manual activity “adapt business object model” that manipulates an already available model of the business object (“BO Model”). In activity “transform to ESR model” the “BO model” is translated manually to another format (“ESR BO model”). Further, there is an alternative activity to create an “ESR BO model” directly: “create ESR model”. The automated activity “generate code & tables” consumes the “ESR BO model” and creates database “tables” as well as “BO code” that references these “tables”. The implementation is finished manually within activity “write business logic”. In addition, activity “model S&AMs” is used to create a status and action model (“S&AM model”) that is reference from the “BO code”. What is not shown in the extract is that these “S&AM models” are interpreted during “BO code” execution.

Considering the predecessors and successors of the different activities, one global synchronization point can be identified: activity “generate code & tables”, which has three predecessor sets: the first consists of “adapt business object model” and “transform to ESR model”, the second includes “transform to ESR model”, only, and the third consists of “create ESR model”. The other two activities “model S&AMs” and “write business logic” are successors of “generate code & tables”. Both activities, “model S&AMs” and “write business logic”, are successor as well as predecessor for each other. Consequently, they are not synchronization points themselves.

The activities “adapt business object model”, “transform to ESR model”, and “create ESR model” belong to a first phase, while the activities “model S&AMs” and “write business logic” belong to a second phase. Thus, the number of phases that can be identified for this excerpt of case study BO is two. In this example the three predecessor sets of “generate code & tables” reveal that there are alternative ways to retrieve an “ESR BO model”. When searching for local synchronization points this might be used to filter the activities from Figure 7.19, such that only activities that belong to one of the predecessor sets of activity “generate code & tables” remains (i.e. activity “create ESR model”

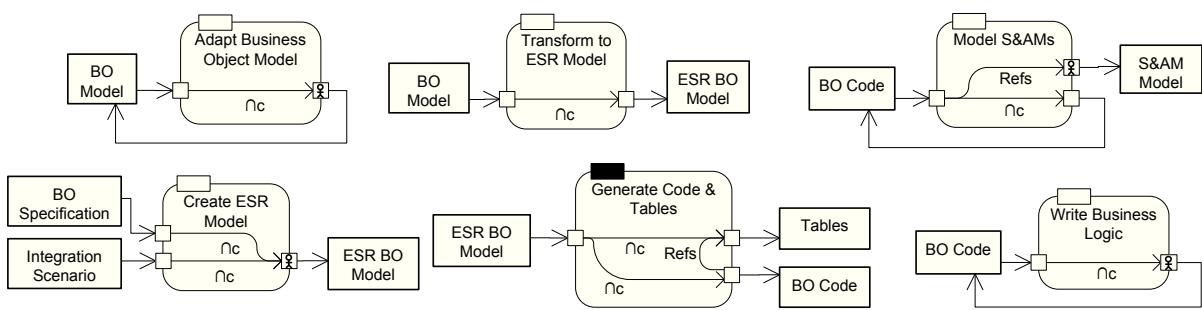


Figure 7.19.: Excerpt from Software Manufacture Model of BO.

is removed from the setting for this consideration). Afterwards activity “generate code & tables” is still a global synchronization point. However, two additional synchronization points can be identified: “transform to ESR model” and “adapt business object model”. Activity “transform to ESR model” has one possible predecessor, which is “adapt business object model”, and three successors, which are “generate code & tables”, “model S&AMs”, and “write business logic”. For activity “adapt business object model” all other activities are successors.

This example shows how local synchronization points might be identified. However, the extract of the case study BO that is considered here includes no activities that are predecessor of “generate code & tables” and successor of “transform to ESR model”, and also no activities that are predecessor of “transform to ESR model” and successor of “adapt business object model”. Thus, there is no additional phase between these new synchronization points.

While the identification of global synchronization points and phases can be fully automated, the identification of relevant local synchronization points and phases can only be supported and requires manual reasoning in the end. The automation support provides a set of candidates for local foci that contain local synchronization points and phases. However, not each local focus is meaningful within an MDE setting and for the same local synchronization point (combined) local foci of different size might be identified. This is the reason why further manual reasoning about the relevance of local synchronization points is required.

A first step is to decide whether a retrieved local focus is meaningful. For example when the activities in the focus can be associated with the creation of a partial product this focus is meaningful. Next the relevant combination of local foci should be chosen. A subset of the automatically identified local foci can be chosen, such that there is one or more local foci for each relevant local synchronization point. A local focus where multiple local synchronization points are included or where multiple phases are created is potentially more relevant when a local focus without phases. It is possible that a local focus is part of another local focus.

Finally, the identified local foci and local synchronization points can be enriched with further information. For example, it has to be identified whether a synchronization point is a miniphase (i.e. an extensive manual modeling or coding activity with the potential that developers work parallel in this activity). Alternatively, an identified phase might consist of alternative activities, only (which is interesting, since no additional options for parallel working can be expected from that phase).

The prototypical implemented identification of (candidates for) synchronization points and phases is described in Section 8.3.2.

7.2.2. Analysis Method for Manual Information Propagations

Manual information propagation can be identified by simply identifying matches of the pattern shown in Figure 7.20 to a Software Manufacture Model. Each occurrence of the pattern describes a pair of artifacts, where content from one artifact is moved in a manual activity to the other artifact (referred to as *manually transformed artifact pair*). The pattern might match multiple times to the same activity.

It is not only interesting to identify activities where manual information propagation occurs. The actual number of the manually transformed artifact pairs provides an impression of the extent of manual

7. Analysis

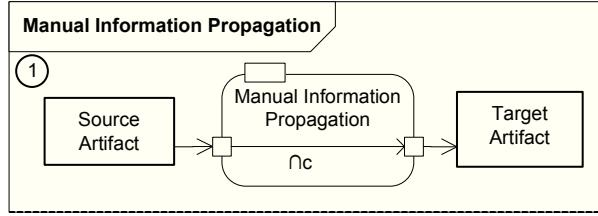


Figure 7.20.: Software Manufacture Model pattern manual information propagation

information propagation. *Manually transformed artifact pairs* are all pairs of artifacts there one is matched to *source artifact* and the other is matched to *target artifact* in the same match of the pattern.

Due to the semantic of the pattern match that is described in Chapter 6, pattern matches can also affect semi-automated activities. When the analyzed models are created such that activities are only modeled as semi-automated, when the respective manual effort is minimal, all matches of the pattern to semi-automated activities can simply be ignored. Manual activities might for example be modeled as semi-automated activities, when the modeler wants to express that an advanced editor supports definition of references between model elements and implementing services. If this happens, each pattern match to a semi-automated activity has to be reconsidered individually to decide whether the match can be ignored or not.

To avoid this effort, following modeling convention should be fulfilled before this analysis is performed on a Software Manufacture Model:

MC4 Activities within a Software Manufacture Model are modeled as semi-automated only, when the respective manual effort is limited and constant with respect to the automatically performed part of the activity.

Example 14

Again an extract of case study BO that is shown in Figure 7.19 is used as example.

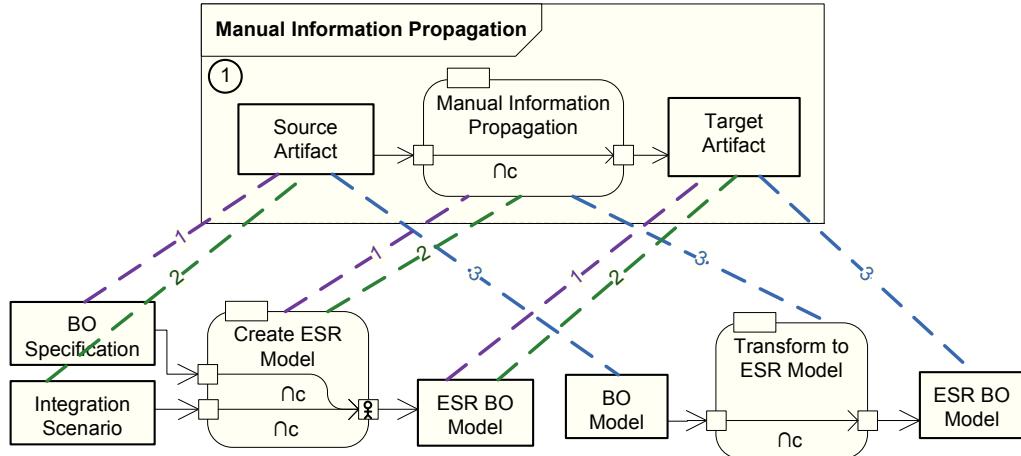


Figure 7.21.: Illustration of three matches of pattern *manual information propagation* on activities from case study excerpt in Figure 7.19

Figure 7.21 shows the three matches of pattern manual information propagation that can be identified on this example. Two matches can be identified on activity “create ESR model” and one match can be identified on activity “transform to ESR model”. The three manually transformed artifact pairs in this extract are: “BO specification” and “ESR BO model”, “integration scenario” and “ESR BO model”, as well as “BO model” and “ESR BO model”.

7.2.3. Analysis Method for Complexity of Activity Chains

As introduced in Section 3.3.3, the measure that is used in this thesis for the complexity of activity chains is the *lengths of activity chains*. The length of an activity chain defines the number of activities that have to be performed to apply a change to one artifact and propagate it to a resulting artifact (e.g. the artifacts that are used for final compilation or interpretation of the product). This length is not fixed number, but has a minimum and a maximum, since alternative chains might exist as well as optional activities within a chain.

For a pair of artifacts, where the change is applied to the first artifact and should be propagated to the second artifact, the length of activity chains (i.e. the number of activities that have to be executed) can be retrieved as follows. All predecessor sets for the second artifact have to be derived. These predecessor sets are filtered, such that predecessor sets are removed that include no activities that consume, manipulate, or produce the first artifact. From the remaining predecessor sets each activity is removed that does not create or manipulate the first artifact directly and that is no successor of an activity that does create or manipulate the first artifact. Now, the minimum length of the activity chain for the studied pair of artifacts is equal to the number of activities in the smallest predecessor set, while the maximum length of the activity chain for the studied pair of artifacts is equal to the number of activities in the biggest predecessor set. If there is no such predecessor set, changes from the first artifact are not propagated to the second artifact in context of the MDE setting.

This complexity measure focuses on the number of context switches for a developer and activities might be modeled with a high degree of detail. To gain comparable results nonetheless, a Software Manufacture Model that is used for this analysis should be modeled such that following convention is fulfilled.

MC5 *In case, multiple automated activities follow each other directly (i.e. no other activity is successor of one of these activities and predecessor of another of these activities, at once) and these automated activities are combined automatically, such that one trigger is sufficient to execute all of these activities, they should be modeled as a single automated activity. In case, multiple manual activities that follow each other directly (i.e. no other activity is successor of one of these activities and predecessor of another of these activities, at once) affect the same set of artifacts (i.e. each of these activities produces or manipulates the same set of artifacts), they should be modeled as a single manual activity.*

Other aspects that can cause misleading results are explicitly modeled workarounds. When activities are included in the model that are not meant to be supported by the MDE setting, but represent workarounds that are applied by developers, the analysis can lead to lengths that are too short (since the workaround is represented instead of the actually required development path). To prevent this effect, following modeling convention should be fulfilled before this analysis is performed on a Software Manufacture Model:

MC6 *A Software Manufacture Model that is used for the analysis of the length of activity chains should not include activities that represent workarounds.*

Example 15

This should be illustrated on the two artifacts “BO model” and “BO code” from the extract of case study BO that is shown in Figure 7.19. Figure 7.22 shows the different predecessor sets for artifact “BO code”. First, the predecessor sets 1 - 4 are removed, since no activity is included that consumes, manipulates, or produces artifact “BO model”. All activities in the remaining predecessor sets are either activities that create or manipulate artifact “BO model” or their successors. The resulting minimum number of activities can be found in predecessor set 5, where only the activities “generate code & tables” and “transform to ESR model” are included. The maximum number of activities can be found in predecessor set 12, where in addition the activities “model S&AM”, “write business logic”, and “adapt business object model” are included. Consequently the length of activity chain for the artifact pair “BO model” and “BO code” is between 2 and 5 for this extract of case study BO. Note that this minimum length includes no activity to apply the change to artifact “BO model”. This is, because this artifact is produced outside the modeled MDE setting and might be changed there, too.

7. Analysis

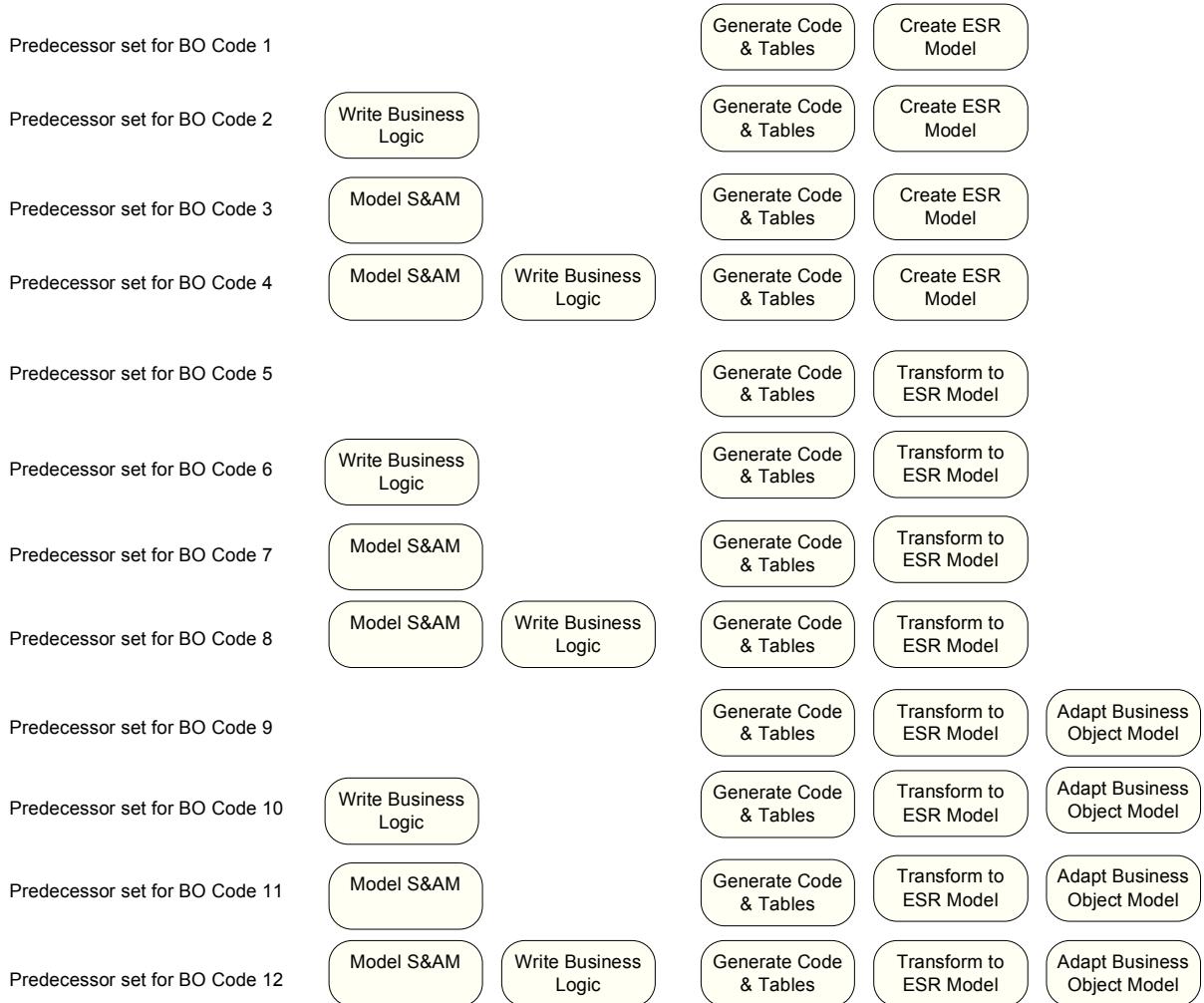


Figure 7.22.: Predecessor sets for artifact “BO code” base on case study excerpt from Figure 7.19

Now the artifact pair “ESR model” and “BO code” can be considered. Again the predecessor sets for “BO code” are taken as basis (Figure 7.22). None of the predecessor sets is removed since all contain at least one artifact that consumes, manipulates, or produces “BO model”. In a next step the activity “adapt business object model” is removed since it neither consumes, manipulates, or produces “BO model”, nor is it a successor of an activity that consumes, manipulates, or produces “BO model”. The resulting minimum number of activities can be found in predecessor sets 1 and 5, where only two activities are included, respectively. The maximum number of activities can be found in predecessor sets 4, 8, and 12, where four activities are included, respectively. Consequently the length of activity chain for the artifact pair “ESR model” and “BO code” is between 2 and 4 for this extract of case study BO.

The calculation of the lengths of activity chains is automated. The automated support is capable of providing the lengths of activity chains for all pairs of artifacts in the MDE setting. However, not all artifacts are actually subject to the introduction of changes, and not all changes are probable. In order to consider only relevant artifact pairs when evaluating a setting, manual decisions have to be met.

Therefore, first two types of artifacts have to be chosen. On the one hand, the analyst decides which artifacts are *change artifacts* (i.e. artifacts that are used for the introduction of changes). On the other hand, the analyst decides which artifacts are *target artifacts* (i.e. artifacts where the change should be propagated to). The artifact pairs that are then considered for the analysis are all pairs of one *change artifact* (as first artifact) and one *target artifact*.

Further, it is possible that some of the change artifacts are only seldom used for changes while the application of changes to other artifacts is more probable. Thus, an analyst can mark *common changes*, which are changes that are not only possible, but that are used regularly. Then a *range* can be retrieved, which describes the minimum and maximum number of activities that have to be applied for the *common changes*. Further, the *maximum length of activity chains* is the maximum number of different activities that have to be applied to implement and propagate a change that was found for a pair of artifacts. The implementation of the automation support for the assessment of the lengths of activity chains is described in Section 8.3.3.

7.3. Analysis Approach

In this Section an analysis approach for the practical application of MDE settings is introduced. This analysis approach comprises a set of use cases for analysis of MDE settings. It shall be explained how the introduced analysis techniques can be used to address the different use cases (as summarized in Figure 7.23).

The use cases can be split into three groups. First, the analysis techniques can be used at the start of a project or when a new or existing MDE setting is taken into operation. Possible goals of an analysis at this point in time are to plan the combination with software development processes and to make a risk assessment. Second, the analysis techniques can be used to decide about refactoring options for an MDE setting that is used regularly in different projects. Finally, the analysis techniques can be used to support evolution decisions, when an existing MDE setting is not sufficient to fit arising needs on the software products of a company or when other motivations for evolution occur.

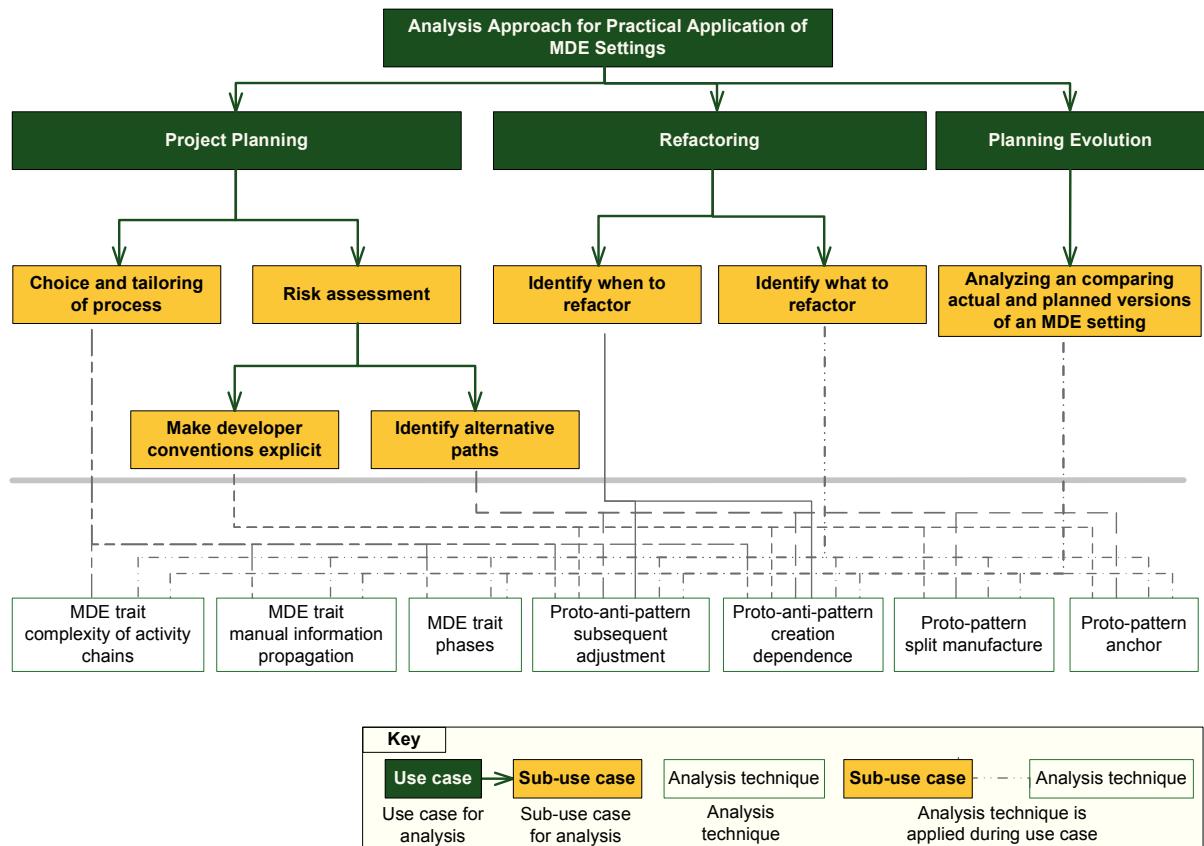


Figure 7.23.: Overview of analysis approach and uses for analysis techniques.

Project Planning

When taking an MDE setting into operation the analysis techniques can be applied to ensure that the right process or process tailoring is combined with the MDE setting. Measuring the length of activity chains as well as the assessment whether hard changeability concerns are affected with the help of the Software Manufacture Model proto-anti-patterns can be used as a basis to decide whether an agile process can be combined with the MDE setting. As described in Section 3.3, the identification of process-relevant manifestations of different MDE traits can motivate an adaptation of the process: When manual transformations are identified, the process tailoring should be done, such that the corresponding need for additional quality assurance can be addressed. Further, the identification of synchronization points should be considered when planning how work is split along a team and how release cycles are planned.

Further the analysis techniques can be used for an initial *risk assessment*. The basis for such a risk assessment is an analysis how changeability concerns are affected. A risk assessment for an MDE setting that is already chosen for a project has two reasons:

First, especially proto-patterns (not anti-patterns) encode expectations on manual activities. For example, for an occurrence of *split manufacture* only specific parts of the modified artifact are allowed to be touched manually. Further, occurrences of proto-anti-patterns provide hints which artifacts should not be changed without care and agreement within the team. For example, for an occurrence of *subsequent adjustment*, changes should not be applied to artifacts that can cause the new execution of *initial creation*. Consequently, a couple of necessary developer conventions, on what artifacts or artifact parts cannot be modified safely, follow from the MDE setting. The risk analysis helps to identify these conventions and to make them explicit for developers.

Second, when a proto-anti-pattern occurrence is identified and can be associated with risk, the MDE setting should be analyzed for alternative ways to implement a change. Goal of a change is to propagate the change to the resulting artifacts. The activities that are actually matched to the proto-anti-patterns might not be part of each predecessor set of the targeted resulting artifacts. Thus, there are alternative, risk-free, paths of activities that can be used to create or manipulate the required artifact. One task of a risk assessment is the identification of such alternatives, such that the developers gain the information how changes can be applied safely.

Refactoring

A main reason to analyze an MDE setting is to identify potential for refactoring. In general, a refactoring can happen for different reasons. However, the identification of proto-anti-patterns can help to predict situations that can make a refactoring necessary. During an analysis of changeability concerns with the proto-patterns, each occurrence of a proto-anti-pattern is further analyzed for mitigating factors that prevent a manifestation of the associated changeability problem. Mitigating factors that are identified as relevant in a situation should be documented explicitly. A change in such a mitigating factor at any time can lead to the manifestation of the problem. Consequently, the mitigating factors need to be observed, since their omission can enforce the need to remove the proto-anti-patterns from the MDE setting. For example, an artifact “*configuration file*” that is matched to *base artifact* in an occurrence of *subsequent adjustment* might never be changed. However, a change in the offered options to customize the software product can lead to frequent changes of this “*configuration file*”. In this case, the changeability problem of the pattern can manifest and refactoring is required.

Apart from the question when a refactoring becomes necessary, the analysis techniques directly help to identify what should be addressed in a refactoring. Occurrences of proto-anti-patterns are natural candidates for refactoring. The description of these proto-anti-patterns already includes resolution strategies that can be used for refactoring. Proto-patterns already document possible solutions.

Also an identified process-relevant manifestation of an MDE trait can be target of refactoring efforts. For example, a manual transformation might be supplemented by automated support or even be substituted by a full automation. Also the length of activity chains or the existence of synchronization points can be chosen as target of a refactoring (i.e. in order to enable the use of the MDE setting with agile processes).

Planning Evolution

As described in Section 3.4 structural evolution might happen to MDE settings for the most diverse reasons. For example, it might be planned to automated additional parts of the development.

Structural evolution can change the properties of an MDE setting, such that the influence of changeability changes or that the manifestations of the MDE traits changes. Thus, it is important to ensure that an evolution step does not lead to a worse changeability or has unexpected effects on the manifestation of MDE traits.

A general approach to support evolution decisions is to model the MDE setting that will result from the evolution step as Software Manufacture Model, too. On this model the analysis techniques presented in this chapter can be applied, to identify how changeability support of the new MDE setting will be and what process-relevant manifestations of MDE traits will occur. This way, different variants of the new MDE setting might be analyzed and compared to the actual setting. In future it might, in addition, become possible to collect “best practices” for evolution steps that can be safely applied to handle specific evolution needs.

8. Implementation

In this chapter, tools and automation support for the languages and analyses presented in Chapters 6 and 7 is presented, as summarized in Figure 8.1. The Software Manufacture Model language and the Software Manufacture Model pattern language are supported by editors. For both languages non-graphical tree editors were developed. In addition, a graphical editor for the new version of the Software Manufacture Model language was developed in cooperation with eight students in a seminar.

An initial version of the pattern matcher was developed on the basis of the initial Software Manufacture Model meta model. This pattern matcher was adapted for the new Software Manufacture Model meta model, again in cooperation with students in the seminar. During the seminar the students were encouraged to experiment with the notation of the Software Manufacture Model language. As a consequence the notation within the graphical editor differs from the notation that was presented in Section 6.2. This concerns the annotation of the degree of automation, the notation of the set pins as well as the notation of the links including link direction and relations. The differences in the notation are explained in Appendix D.

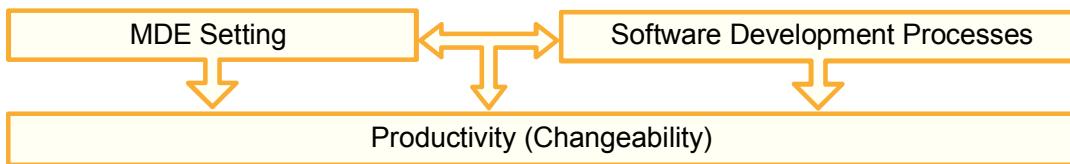


Figure 8.1.: Overview of this chapter: implementation supports modeling of MDE settings as well as analysis of an MDE setting's impact on changeability and process interrelation.

In the following the editor support is shortly described (Section 8.1), followed by an introduction of the automated pattern matcher (Section 8.2). Afterwards the automation support for the order-based analysis methods is presented.

This chapter is partially based on [P2].

8.1. Editors

The tree editors for both, Software Manufacture Model language and Software Manufacture Model pattern language, are developed based on the Eclipse modeling framework (EMF version 2.8.1¹) in Eclipse Juno². Both meta models were modeled as Ecore meta model. On that basis the tree-editors were generated with EMF. Figures 8.2 and 8.3 show screen shots of the Software Manufacture Model tree editor (showing parts of case study Carmeq) and the pattern editor (showing pattern *Creation Dependence* (see Section 7.1.4)).

The graphical editor for the Software Manufacture Model language was developed on the basis of the tree editor with the graphical modeling framework (GMF Runtime 1.6.0³) and Eugenia⁴. Figure 8.4 shows a screenshot of the graphical editor (showing parts of case study Carmeq).

8.2. Pattern Matcher

The pattern matcher is implemented in Java on the basis of the described editors. For the identification of a pattern match, for each activity within the considered Software Manufacture Model and each pattern

¹Eclipse Modeling Framework (<http://www.eclipse.org/modeling/emf/> (last access at November 9th, 2013))

²Eclipse Juno version 4.2.0 (<http://www.eclipse.org/> (last access at November 9th, 2013)); Java 1.7.0 SE Runtime Environment

³Graphical Modeling Framework (<https://projects.eclipse.org/projects/modeling.gmp.gmf-tooling> (last access at November 9th, 2013))

⁴Eugenia (<http://www.eclipse.org/epsilon/doc/eugenia/> (last access at November 9th, 2013))

8. Implementation

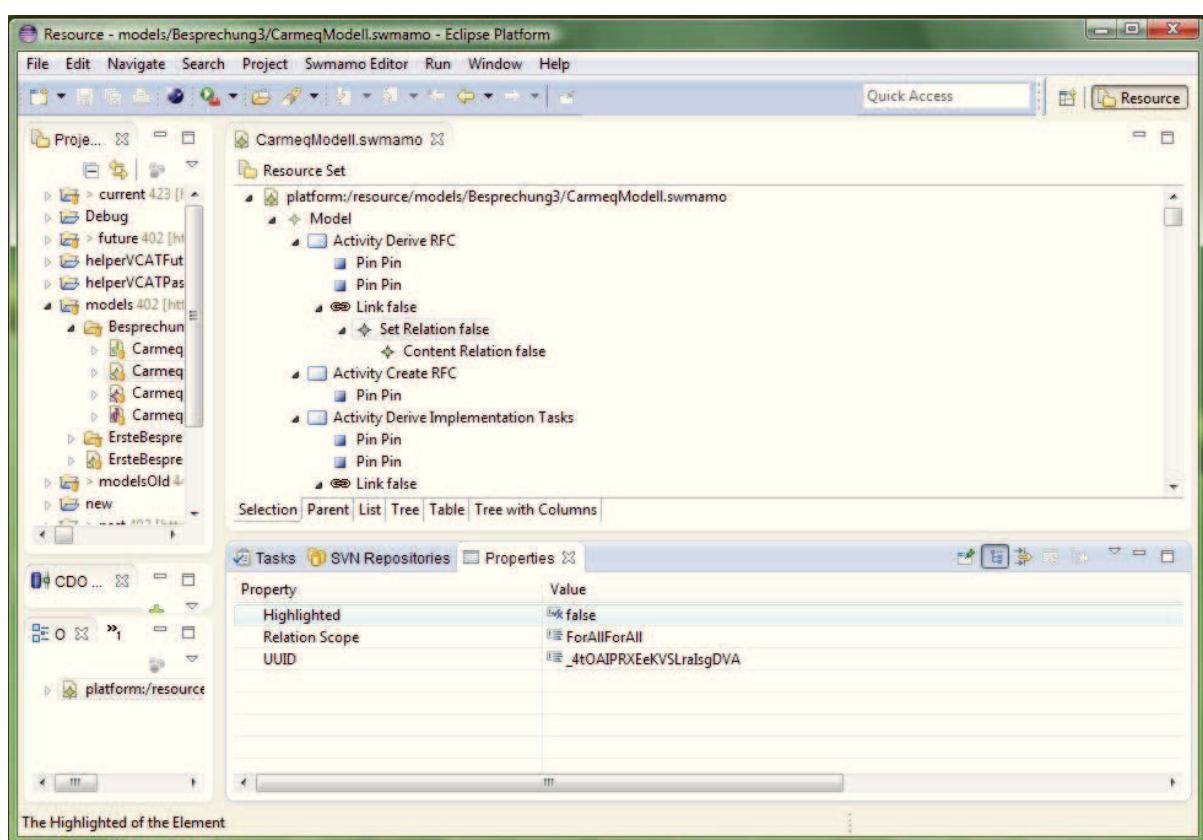


Figure 8.2.: Screenshot of Software Manufacture Model tree editor

activity a graph is created. This transformation is done in a way that different node types are created for the different elements of the Software Manufacture Model (i.e. artifact, pin, connector, and link). Also for each connected activity a node is created. In addition, links of different types are transformed to different node types, such that the identification of graph matches becomes faster. For pattern activities with negative structure two graphs are created: one that includes elements of the activity that do not belong to the negative structure, only, and one that includes all elements of the activity.

After this transformation a library for graph matching is used to retrieve an isomorphic match⁵. For each identified match a mapping between the matched pattern activity and the activity of the Software Manufacture Model is created and returned. This mapping includes detailed information which artifacts, connectors, links, and pins for a Software Manufacture Model activity are matched to which artifacts, connectors, links, and pins of a pattern activity. Then, the mappings of links for each identified activity mapping are evaluated. This is done to remove mappings, where the mapped link does not refine the link for the pattern activity. For example, a mapping of a *refines* link to a *refines* link in the pattern activity is incorrect when the direction of both links is different (i.e. start pin of the link in the activity is mapped to the end pin of the link in the pattern activity and the other way around).

As a next step for each identified mapping of a pattern activity with negative structure, the corresponding mapping of the pattern activity without negative structure (i.e. with the same mapping of activity elements to elements of the pattern activity) is removed.

Finally, it is tested, which identified matches of pattern activities can be combined to matches of the whole pattern. Therefore it is necessary to check whether artifact roles that are produced or consumed by multiple pattern activities of the same pattern are mapped to the same artifacts in the mappings that should be combined to a pattern match. If this is the case a pattern match is identified. Resulting matches are automatically highlighted in the graphical editor. A screenshot of the output of the pattern matcher is shown in Figure 8.5.

⁵http://www.hpi.uni-potsdam.de/giese/gforge/mdeLab/?page_id=819 (last access at November 9th, 2013)

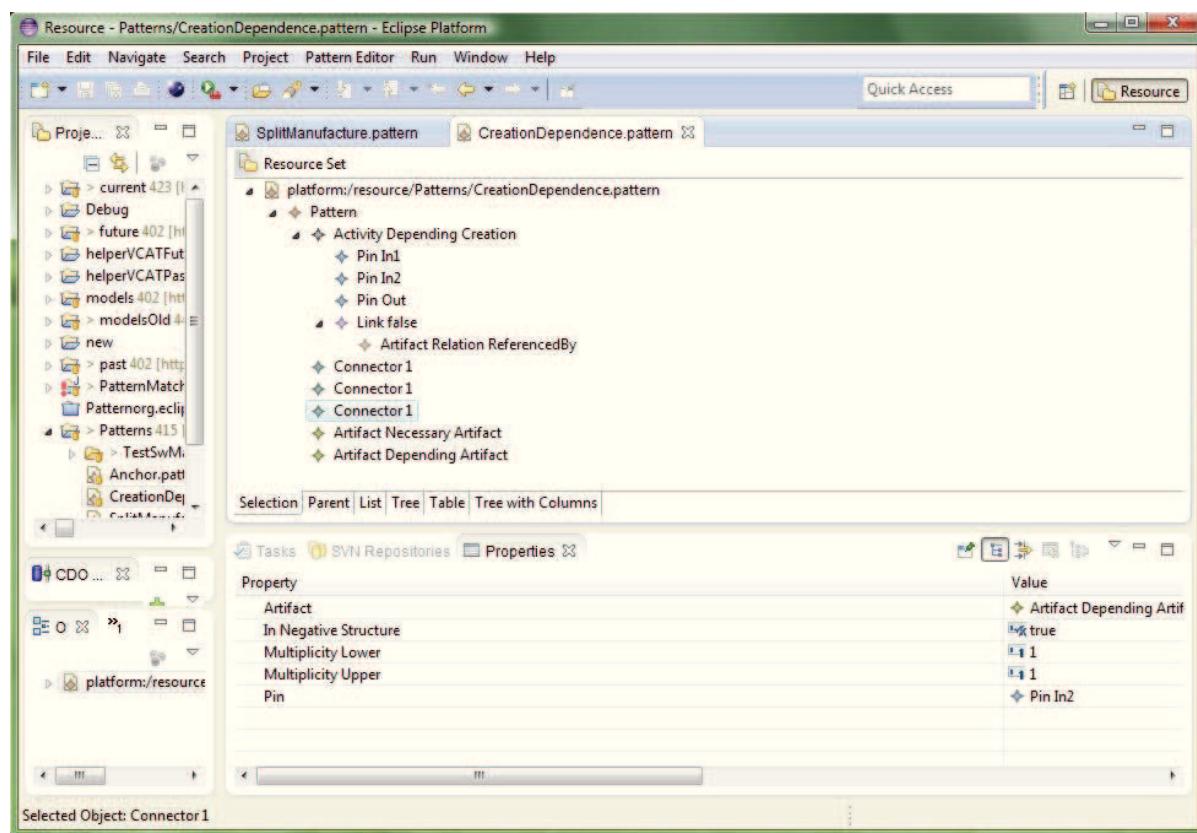


Figure 8.3.: Screenshot of pattern editor

8.3. Automation Support for Order-Based Analysis Techniques

In the following, the implementation of the identification of predecessor sets is described (Section 8.3.1). Finally, the automation support for the assessment of synchronization points and phases (Section 8.3.2) and lengths of activity chains (Section 8.3.3) are introduced.

8.3.1. Automated Identification of Predecessor Sets, Predecessors, and Successors

In Section 6.2.4 the terms predecessor set, predecessor, and successor are introduced. The information about predecessor sets of activities within an MDE setting is the basis for the assessment of *phases* and *lengths of activity chains*. Therefore, automation support for those two analyses requires an automated identification of predecessor sets (and with it of predecessors and successors). In this section, it is explained how predecessor sets, predecessors, and successors are automatically derived on the basis of a Software Manufacture Model.

For the illustration of the identification of predecessor sets, the EMF example (see Section 2.6) is used in the following. The Software Manufacture Model of the corresponding activities is shown in Figure 8.6. As indicated in the introduction of the EMF example in Section 2.6 three additional activities shall be added for better illustration: “*create genmodel*”, “*create documentation*” and “*merge into*”. Activity “*create documentation*” is a manual activity to create a documentation, while “*merge into*” is a semi-automated activity that allows to include the “*ecore model*” into the documentation. “*Create genmodel*” is an alternative activity for the creation of artifact “*emf generator model*”. *Start activities* in this example are “*create ecore model*”, “*create genmodel*” and “*create documentation*”. The resulting artifacts in the example are “*interface*”, “*adapter factory*”, “*interface implementation*”, and “*documentation*”.

8. Implementation

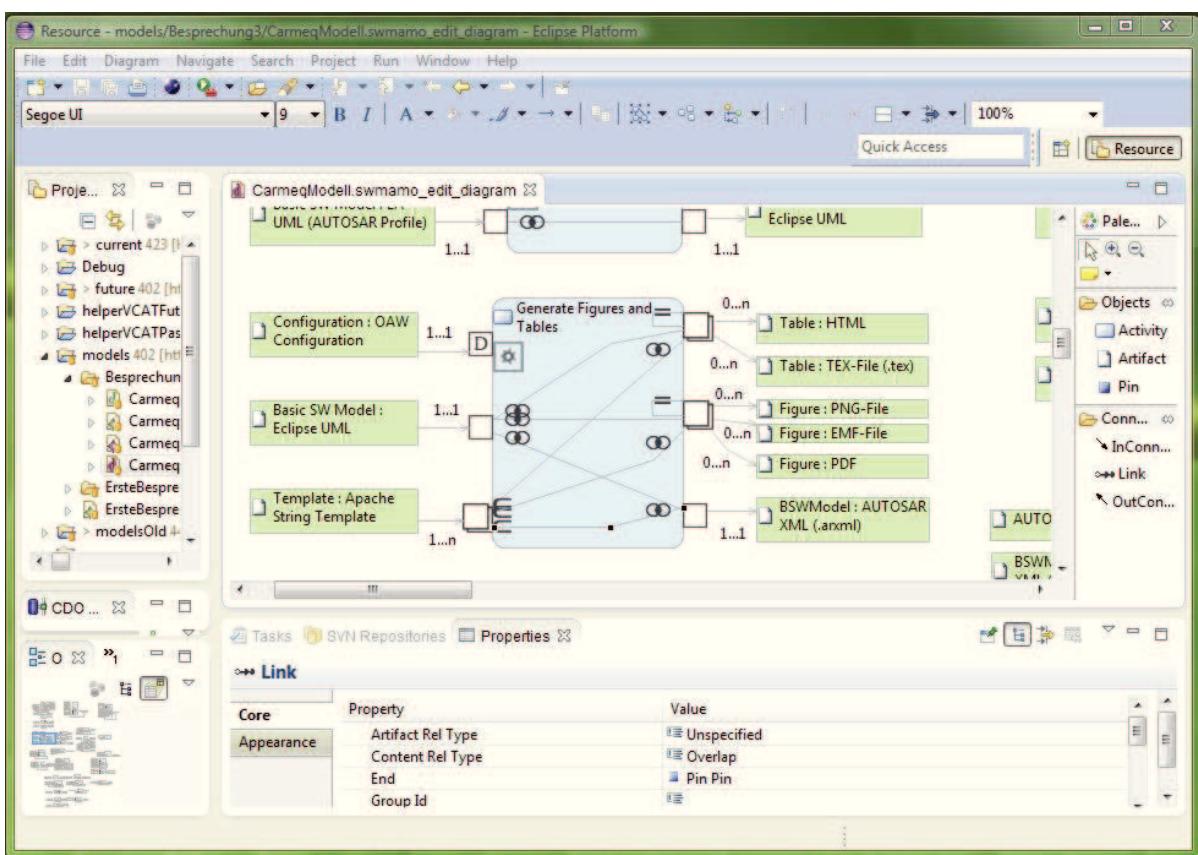


Figure 8.4.: Screenshot of graphical Software Manufacture Model editor

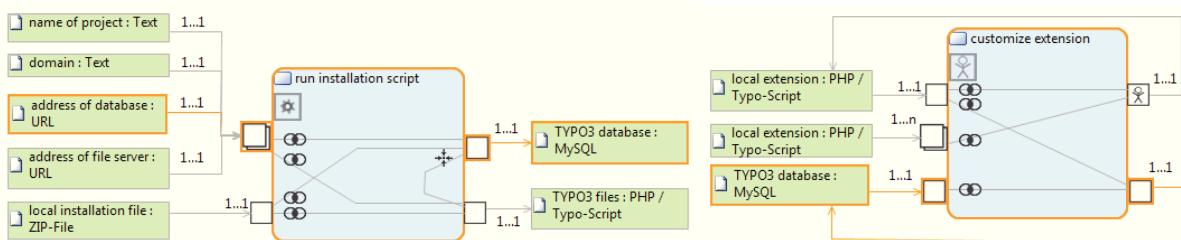


Figure 8.5.: Highlighted match of pattern *subsequent adjustment* (see Section 7.1.3) in activities of case study VCat

Identifying Predecessors

To identify the *predecessor sets* of a given activity each artifact that is consumed by this activity is considered. For each artifact two sets of activities are collected: first, all activities that produce one or more of the artifacts that are consumed by the given activity and second, all activities that manipulate one or more of the artifacts that are consumed by the given activity. Similarly the identification of *predecessor sets* for a given artifact starts with the identification of activities that create or manipulate this artifact. For example, the set creating activities for artifact “*interface implementation*” includes “*generate java code*”, while the set of manipulating activities includes “*manipulate implementation*”.

On the basis of these two sets of activities all possible candidate predecessor sets are created for all artifacts. A possible candidate predecessor set includes for the considered artifact (i.e. in case of a given activity: an artifact that is consumed, and in case of a given artifact: this artifact) one activity that creates this artifact (if there is such an activity in the MDE setting) as well as an arbitrary combination of activities that manipulate a considered artifact. In case that an artifact is optional input of the activity

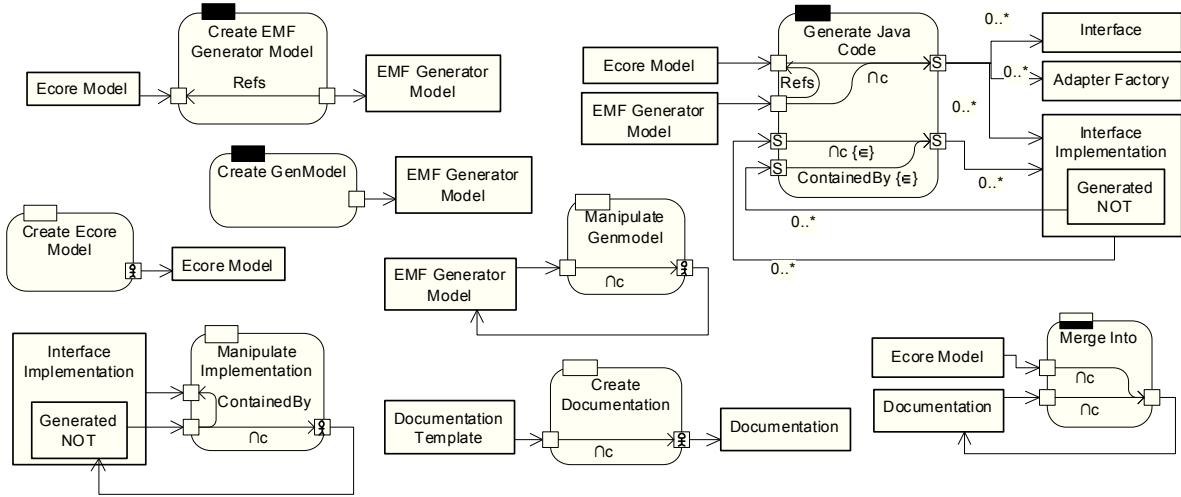


Figure 8.6.: Software Manufacture Model of activities from the *extended EMF example* (Section 2.6).

an empty predecessor set candidate is added as alternative to the set of predecessor set candidates for this artifact. For example, the two candidate predecessor sets that are shown in Figure 8.7 can be created.



Figure 8.7.: Initial candidate predecessor sets for artifact “*interface implementation*”.

Here a first filter is applied to remove predecessor set candidates that contain the activity under study, since an activity cannot be its own predecessor. Afterwards all combinations of candidate predecessor sets for the different consumed artifacts are retrieved (one candidate predecessors set per consumed artifact) to gain all initial predecessor set candidates for the activity under study.

In a next step these possible candidate predecessor sets are further filtered (in case of a given activity). For each candidate predecessor set it is checked whether the relations that belong to the precondition of the activity can be created within this candidate predecessor set. The candidates that cannot fulfill these preconditions are removed. Precondition links that are connected to optional input artifacts do not need to be fulfilled, when the optional input artifact is not provided by the predecessor set.

Finally, the consumed artifacts and the activity can be registered as “handled”.

Now for each of the retrieved initial candidates all activities are examined. All artifacts are identified that are consumed by one of the activities in the candidate and that were not yet investigated for predecessors. For those artifacts the above described steps are repeated to identify their candidate predecessor sets. In order to optimize the algorithm, results from previous computations for activities from other candidates can be reused. However, this optimization is not part of the current implementation.

For example, for “*initial candidate set 1*” from Figure 8.7 the artifacts “*ecore model*” and “*emf generator model*” need to be further considered. Artifact “*interface implementation*” which is consumed by “*manipulate implementation*” was already checked for predecessors. As a result, four candidate sets can be identified for artifact “*emf generator model*” and one candidate set can be identified for artifact “*ecore model*”. These candidate sets are shown in Figure 8.8.

Following, all combinations of the initial candidate predecessor with the candidates created for the identified artifacts can be retrieved. Multiple new versions of the initial candidate predecessor set might be created, since each identified artifact might have multiple candidate predecessor sets, too. For example, Figure 8.9 shows the new versions of “*initial candidate set 1*”.

These new combinations for initial candidate predecessor sets are further filtered. Each activity that already was in the initial candidate set before the new versions were created and that was not yet checked

8. Implementation

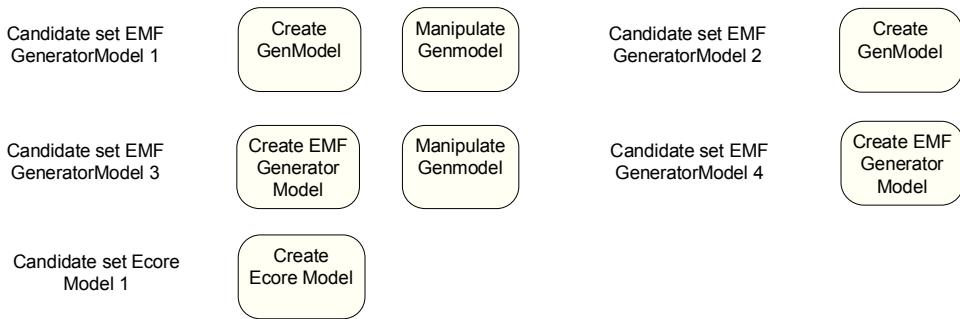


Figure 8.8.: Candidate sets for artifacts “*emf generator model*” and “*ecore model*”.

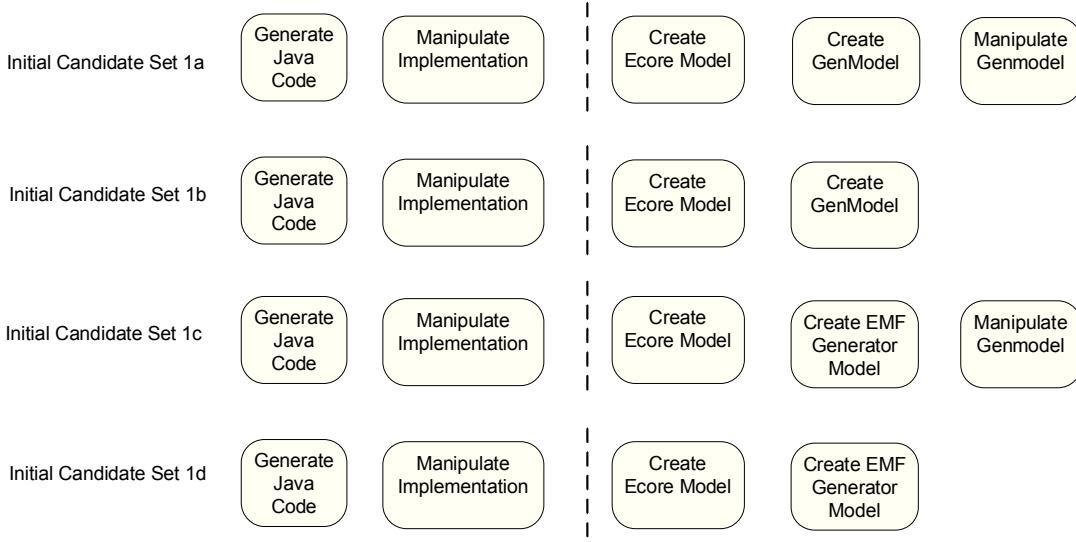


Figure 8.9.: Combinations first initial candidate set from Figure 8.7 and candidate sets from Figure 8.8.

is checked. This check concerns the question whether the relations that belong to the precondition of the activity can be created within the new version of the candidate predecessor set. When a new version cannot fulfill these preconditions it is removed. In the example in Figure 8.9, none of the candidate sets include activities that are already checked. However, the activities “*generate java code*” and “*manipulate implementation*” were already part of the initial candidate set before. Consequently, each candidate set is checked whether it is possible to fulfill the precondition of both activities with this candidate set. The precondition of “*generate java code*” (a reference between two input artifacts) cannot be created within the first two candidate sets. Consequently, only the candidate set shown in Figure 8.10 are further treated.

As a special case this check has to consider that some artifacts in the Software Manufacture Model are *start artifacts*. Preconditions that belong to these start artifacts (i.e. start artifacts as well as required relations to or from start artifacts) do not need to be fulfilled to pass the check. The underlying assumption is that such relations might already be created outside the MDE setting. For example, Figure 8.11 shows two initial candidate sets for artifact “*documentation*”. Activity “*create documentation*” consumes the start artifact “*documentation template*”. Thus, although the candidate sets include no activities that can create “*documentation template*”, the activity “*create documentation*” passes the check (marked as *initial* with an “i”, here).

Due to the combination of predecessor sets for different artifacts, it might occur again that the initial activity is contained in one of the resulting predecessor sets. Thus, the corresponding filter is applied again. Afterwards, all activities can be marked as handled.

This step (the identification of artifacts that have no creating activity in the candidate set and the subsequent identification and integration of its candidate predecessor sets) is repeated until the initial

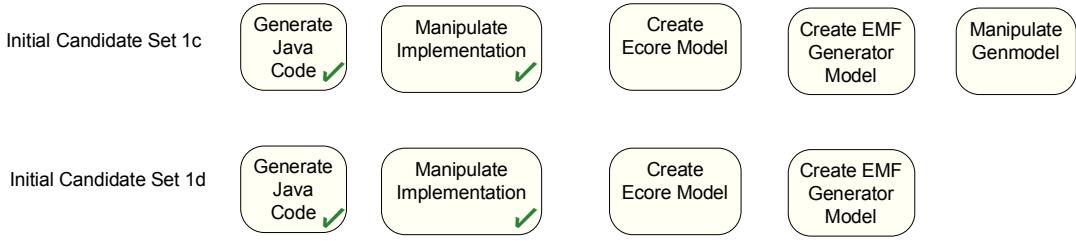


Figure 8.10.: Remaining initial candidate sets after check of activities “*generate java code*” and “*manipulate implementation*”.

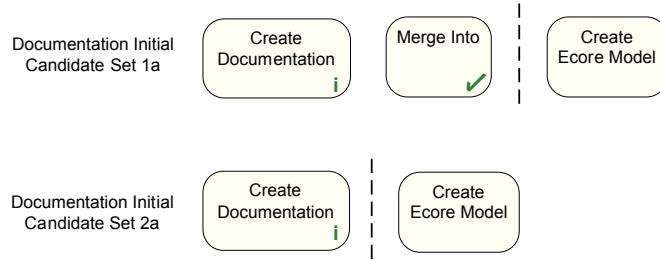


Figure 8.11.: Example candidate predecessor sets for artifact “*documentation*”.

candidate predecessor sets do not change anymore and no new variations are created. In the example, for the new activities in both of these remaining candidate sets no consumed artifacts can be identified that were not yet investigated for predecessors. Thus, the unchecked activities in the candidate sets are checked whether their preconditions can be fulfilled. Figure 8.12 shows all resulting candidate sets (including the candidates that are created on the basis of “*initial candidate set 2*”).

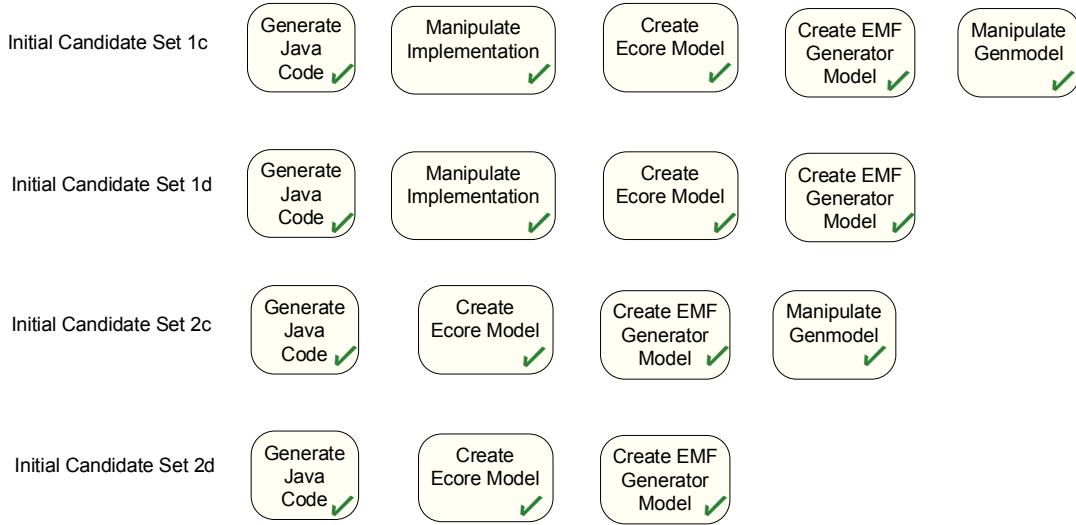


Figure 8.12.: Candidate predecessor sets for artifact “*interface implementation*”, where all activities passed the precondition check.

Some final filters are applied. First, equal candidate sets can result from the combination and extension of candidate sets for the different artifacts. Such clones are removed from the candidate sets. Second, when two artifacts can be created simultaneously by the same activity or alternatively by other activities it is possible that multiple activities that can be used for the creation of the same artifact are combined in one predecessor set (since the predecessor sets are combinations of predecessor sets of the different

8. Implementation

artifacts). Therefore, a last filter removes predecessor sets with such “creation overkill”.

The resulting candidates are the *predecessor sets* of the initially investigated activity or artifact. Figure 8.13 shows the resulting predecessor sets for artifact “*interface implementation*”. Consequently the predecessors for this artifact are “*create.ecore model*”, “*create.emf.generator.model*”, “*manipulate.genmodel*”, “*generate.java.code*”, and “*manipulate.implementation*”.

The presented algorithm can be found in detail in Appendix F.1.

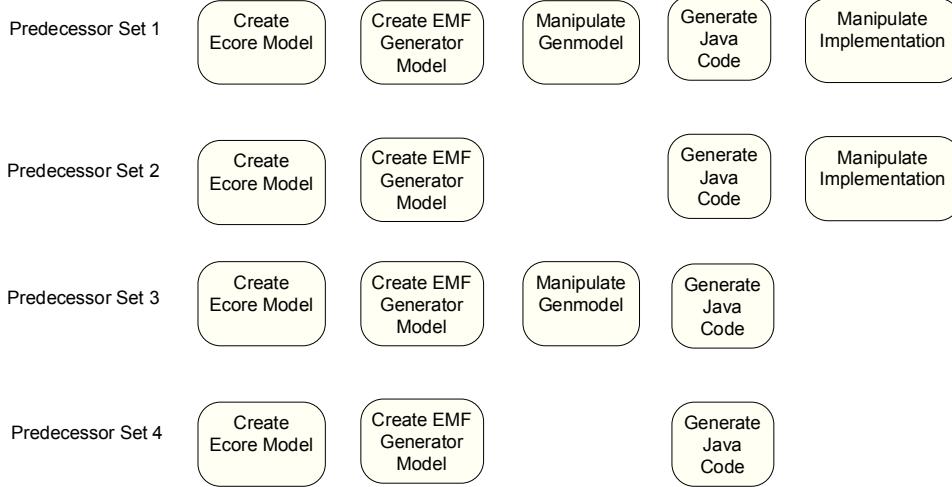


Figure 8.13.: Summary of predecessor sets of artifact “*interface implementation*”.

Identifying Successors

The successors on an activity can be examined straightforwardly when the predecessor sets of all activities are known. In the example of activity “*create.ecore model*” all candidates are successors. However, activity “*create.genmodel*” is not a predecessor for the activities “*generate.java.code*” and “*manipulate.implementation*”. Thus, the only candidate that actually is a successor of “*create.genmodel*” is “*manipulate.genmodel*”.

Discussion

Goal of this algorithm is to identify all possible predecessor sets for a given activity or artifact. A predecessor set (of a specific activity or artifact) is defined as a set of activities that are sufficient to create the precondition for the specific activity or to create the specific artifact. This includes each subset of activities in an MDE setting that might be used to create the required preconditions. However, subsets with activities that cannot be used together (e.g. because they are alternatives to create the same artifacts) are excluded.

There is a more straight forward solution, compared the presented algorithm. This simple solution to retrieve the predecessor sets would be to form all possible subsets of activities in the MDE setting and filter them. First for the complete MDE setting, *start artifacts* are identified. Then, the subsets would be checked whether for each artifact in the set only one activity exists that can be used to create this artifact (“creation overkill”). Second, the subsets would be checked whether for each activity, the preconditions can be fulfilled (i.e. whether the consumed artifacts are start artifacts or can be created within the subset, and whether the required relations are relations to start artifacts or can be created within the subset). However, this simple solution has an exponential complexity, since all subsets of the activities in an MDE setting have to be studied (i.e. the power set). For MDE settings with up to 30 activities, this is not applicable. Consequently, an optimized algorithm was required.

The presented algorithm is this an optimization, although it has still an exponential complexity. The applied optimization results from a reduction of the size of activity sets that are subject to the computation of power sets (and with it to the part of the algorithm that has the exponential complexity).

Consequently, the main difference between the simple solution and the presented algorithm is that in the presented algorithm the predecessor sets are built up recursively. Still the power sets of subsets of the activities in the MDE setting are examined. However, the algorithm aims at reducing the size of these subsets. Only activities that create or manipulate a given consumed artifact are used to build *candidate sets for artifacts* (i.e. the power set of manipulating activities combined with creating activities). Further, due to the recursive algorithm, only a subset of artifacts is studied in each cycle. Thus, the following combination of *candidate sets* for the considered artifacts to *candidate predecessor sets* is limited in the extent, too. The resulting candidates are directly filtered to ensure that impossible *candidate predecessor sets* are not further considered.

This filter includes a check whether preconditions for the activities that were already part of the *initial candidate set* can be fulfilled, with a given *candidate predecessor set*. This check can be done correctly at this point, since all relevant activities (i.e. all activities in the MDE setting that could be used to fulfill the preconditions) were considered for the creation of the *candidate predecessor sets*. Consequently, if a combination of activities can be used to fulfill the precondition there is a *candidate predecessor set* that includes this combination. The break condition for the recursion is the consideration of start artifacts and activities. Finally, the applied filters exclude surplus *predecessor candidate sets* that result from the stepwise combinations. Thus, *predecessor candidate sets* that are removed include “creation overkill”, are clones of other predecessor candidate sets (i.e. equal), or contain the activity that was initial input of the algorithm. Therefore, the presented algorithm leads to the required *predecessor candidate sets* and with it also to information about predecessors and successors.

The benefit of this recursive approach is that the presented algorithm – although still of exponential complexity – is applicable to MDE settings as they were captured during the first and third study, see Chapter 4. Here applicable means that the algorithm returned for each of the case studies within three minutes (or less).

In future, the presented implementation might be substituted by adapting standard techniques for analysis of paths within network, like the Program Evaluation and Review Technique (PERT) or Bernard Roy’s metra potential method (MPM) [223], such that relations between activities are considered.

8.3.2. Automation Support for Analysis of Phases and Synchronization Points

In this section the automation support for the identification of synchronization points and phases is described. As described in Section 7.2.1, the identification of global synchronization points and phases can be fully automated. For the identification of local synchronization points and phases automated support is provided.

For both goals, first, predecessor sets of all activities (and with it a list of predecessors and successors of an activity) need to be identified (which is automated as described in Section 8.3.1).

Global Synchronization Points

The identification of *global synchronization points* are rather straight forward. For each activity in the model it is checked whether this activity is a local synchronization point within a focus that covers all activities of the model.

To check whether such a candidate activity is a local synchronization point within the focus of activities, it is evaluated for each activity in the given focus whether this activity is either predecessor or successor (and not both) of the candidate. If this is true for all activities in the focus the activity is a local synchronization point.

Local Synchronization Points

For the automated identification of *local synchronization points* different strategies are possible. The straight forward strategy would be to retrieve all combinations of an arbitrary number of activities from the model as potential local focus and check for each activity in such a local focus whether it is a local synchronization point in that focus. However, this straight forward strategy suffers from its high computational complexity (which would be exponential). Instead an algorithm was implemented that approaches the identification from a more pragmatic perspective. Therefore, for each activity in the model it is checked whether it is a potential local synchronization point. This check bases on the

8. Implementation

identification of activities that are known to be in an order concerning the candidate activity. On the one hand activities are collected that have the candidate as successor. On the other hand activities are collected that require the candidate before they are executed (i.e. the candidate is part of all predecessor sets of these activities). A candidate is further considered as potential local synchronization points, if both sets include activities. Thus, only activities are considered that have the potential to split a local focus into different phases.

For activities that pass this check a local focus is constructed as a combination of the two sets of activities and the candidate (i.e. activities that have the candidate as successor and activities that require the candidate as predecessor). For example, Figure 8.14 illustrates a local focus that would be constructed for the activity “*Generate Java Code*” from the *extended EMF example* that is shown in Figure 8.6.

For the constructed local focus it is checked whether the candidate actually is a local synchronization point. It is possible that this is not the case, since the local focus is created out of activities that have an order to the candidate, but is not checked before whether the activities are only successor or only predecessor. Further it is for each other activities in the local focus checked whether it is an additional local synchronization point in that focus.

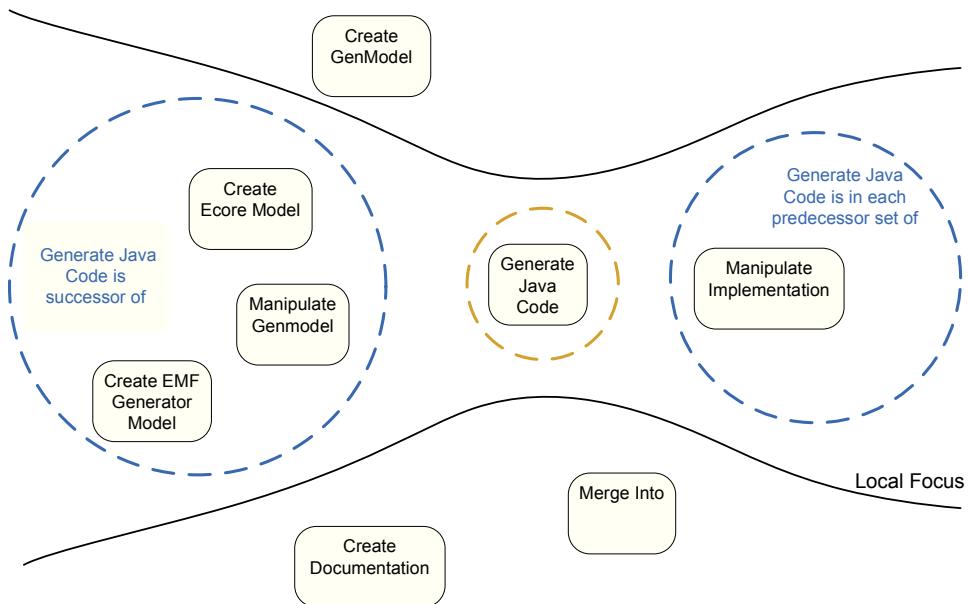


Figure 8.14.: Example construction of a local focus for the potential synchronization point “*Generate Java Code*” within the *extended EMF example* (Section 2.6).

If the local foci of two or more synchronization points overlap in most activities, it might be possible that this overlap represents a local focus, where all of these activities are local synchronization points. To identify such local foci with multiple synchronization points all combinations of two or more already identified local foci are considered. Therefore, it is first checked, whether a combination of a given set of local synchronization points with corresponding local foci is possible at all. This is the case if each local synchronization point is part of the all considered local foci. If this is not the case for one of the local synchronization points a combination is not possible. Then, the new combined local focus is constructed as the overlap of all combined local foci (i.e. it is the combination of all activities that are in each local focus). This way it is ensured that all local synchronization points remain local synchronization points in the new combined focus. Finally, all activities in the new constructed local focus are checked again whether they are additional local synchronization points in that focus.

Phases

Based on the information about global or local synchronization points and the corresponding local foci it is possible to retrieve the resulting phases. Therefore, all synchronization points within a local focus

are put into an order (predecessors followed by their successors). Then these ordered synchronization points are iterated. All activities in the considered focus that are a) predecessors of the current activity and b) not yet part of a phase are combined to a phase. After consideration of all synchronization points the remaining activities (which are successors of all synchronization points) are combined to the last phase.

The presented algorithm can be found in detail in Appendix F.2.

Discussion

On the basis of the identified predecessor sets the identification of global synchronization points or local synchronization points for a given focus is straight forward. Similarly, phases can be automatically calculated for a given focus and a given set of synchronization points.

However, as discussed in Section 8.3.1 the identification of miniphases and meaningful local foci remains subject to manual decisions.

Basically, there are no “wrong local foci”, i.e. each subset of activities in the MDE setting can be a local focus. However, some local foci are more meaningful than others. A part of the above presented algorithm aims at identifying candidates for such foci. The notion of “meaningful” is approximated by two ideas. First, from the viewpoint of the analysis, a local focus is only interesting, when it includes synchronization points that split the focus into two or more phases. Therefore, the identification of local foci starts with the search for activities that are successor for some activities and always predecessor for other activities. Second, a local focus is not considered as meaningful, when the contained activities are uncoupled (i.e. when the activities in the focus can be split into two or more groups, such that each activity from one group is neither predecessor nor successor of any activity in another group). Therefore, the local foci are constructed by collecting predecessors and successors of the potential synchronization point. This initial approach to identify local foci has polynomial complexity.

The result of this initial approach is a local focus with a set of activities, such that the considered potential synchronization point remains a synchronization point. However, sometimes local foci that are slightly smaller contain additional synchronization points. In order to identify such additional local foci, an additional part of the above presented algorithm creates overlaps between the identified local foci. The synchronization points of each of the combined foci remain synchronization points within such overlaps. This approach is implemented straight forwardly and has an exponential complexity. However, due to the limited number of candidate local foci, from the first approximation step, this algorithm turned out to be applicable for the MDE settings that were collected within the first and third study (see Chapter 4).

Again the approach is an approximation. Thus, on the one hand there is a good chance to identify local foci that are split into two or more phases. On the other hand this approximation often leads to additional local foci that contain only one or no phase, but many synchronization points that are strictly ordered. Subsuming, the identification support for local foci can only approximate what local foci are meaningful. Nonetheless, during the application of the algorithm to the case studies from the first and third study (see Chapter 4) the actually implemented approach led to the identification of all expected synchronization points (i.e. all synchronization points that were identified manually, too).

8.3.3. Automation Support for Analysis of Complexity of Activity Chains

In this Section the automation support for the assessment of the lengths of activity chains is described.

First, it shall be described how the lengths are automatically retrieved. As for the identification of synchronization points an automated identification of predecessor sets and successors is required as a first step (the automation of this identification is described in Section 8.3.1).

The automated analysis is performed for all pairs of artifacts in the model. Therefore, first a list of all artifacts in the model is created. This includes a small filtering, since it is assumed that an artifact might occur multiple time within the model (in form of technically distinct model elements) in order to improve readability. Two artifact model elements are defined to represent the same artifact if they have the same role name and type. Based on that list all pairs of artifacts are retrieved. For each of these pairs of artifacts the length of the activity chains between them is calculated. This calculation is done in both directions (i.e. such that both artifacts are once the first and once the second artifact within this calculation).

8. Implementation

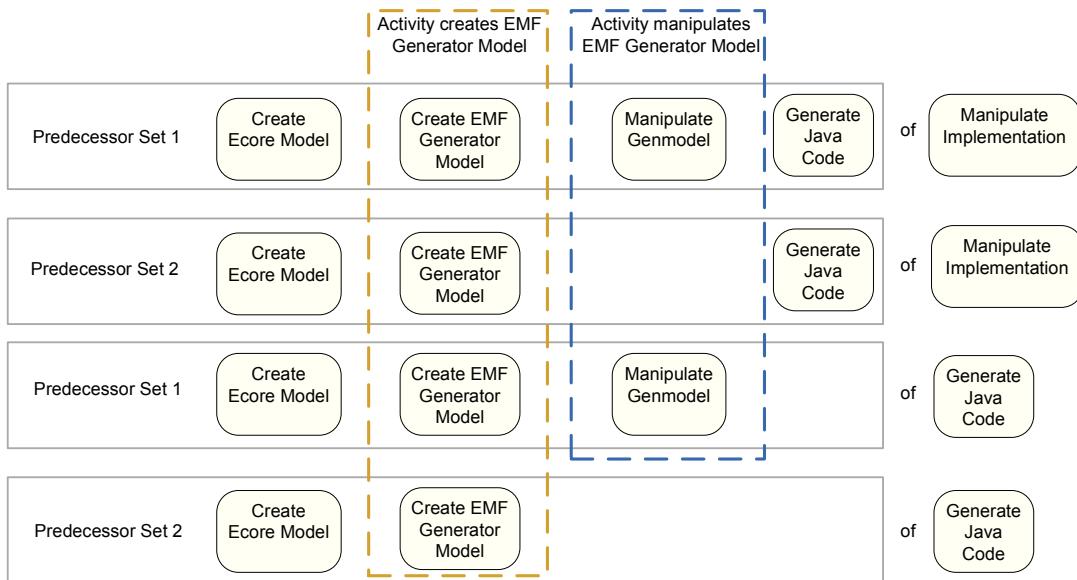


Figure 8.15.: Example predecessor sets of activities that create or manipulate artifact “*Interface Implementation*” within the *extended EMF example* (Section 2.6) that is shown in Figure 8.6.

To calculate the length, first all activities are retrieved that are relevant for the creation, manipulation, and/or consumption of the first and second artifact. Then, all predecessor sets of the second artifact are collected in order to identify all activities that produce or manipulate second artifact. This includes the predecessor sets of all activities that create or manipulate the second artifact. For example, Figure 8.15 shows the predecessor sets for activities that create or manipulate artifact “*Interface Implementation*” from the MDE setting of the *extended EMF example* (Section 2.6) shown in Figure 8.6.

The calculation of the length of activity chains can be based on these predecessor sets, since they represent all sets of activities that can be used to propagate changes to the second artifact. Thus, for all changes that might be propagated from an artifact to the second artifact and all activity chains that can possibly be used for this propagation there is a predecessor set that represents this path.

Only some of the predecessor sets have to be taken into account. Relevant predecessor sets include activities that create, manipulate, or consume the first artifact. For example, in Figure 8.15 the activities that create or manipulate artifact “*EMF Generator Model*” are marked. In addition, the special case that an activity directly consumes the first and produces the second artifact. Note that this second case needs to be handled separately, since an activity is not part of its own predecessor set. If a predecessor set is relevant and not empty a copy is added to the selection of predecessor sets.

Not all activities within a relevant predecessor set might be relevant, too. Therefore, each predecessor set is filtered, such that only activities remain that are used for a change of the first artifact or the propagation of such a change to the second artifact. These are on the one hand activities that manipulate or create the first activity or on the other hand activities that are successors of activities which manipulate or create the first activity. For example, the filtered predecessor sets for the artifact pair “*EMF Generator Model*” and “*Interface Implementation*” are shown in Figure 8.16. Besides the filtering of the predecessor sets it is checked whether an activity creates or manipulates the second artifact directly consumes the first artifact.

If no connection between both artifacts can be identified (i.e. no predecessor set is identified and the check for an activity that directly propagates changes between both artifacts fails) the algorithm can stop for the considered pair of artifacts.

Otherwise the minimum and maximum number of activities that have to be executed to change and propagate a change from first artifact to the second artifact can be retrieved. The size of such an activity chain is equal to the size of a filtered predecessor set, since the filtered predecessor set includes exactly the activities that are sufficient and actually used to propagate a change from the first artifact to the second. The last activity in the chain (i.e. the activity that manipulates or creates the second

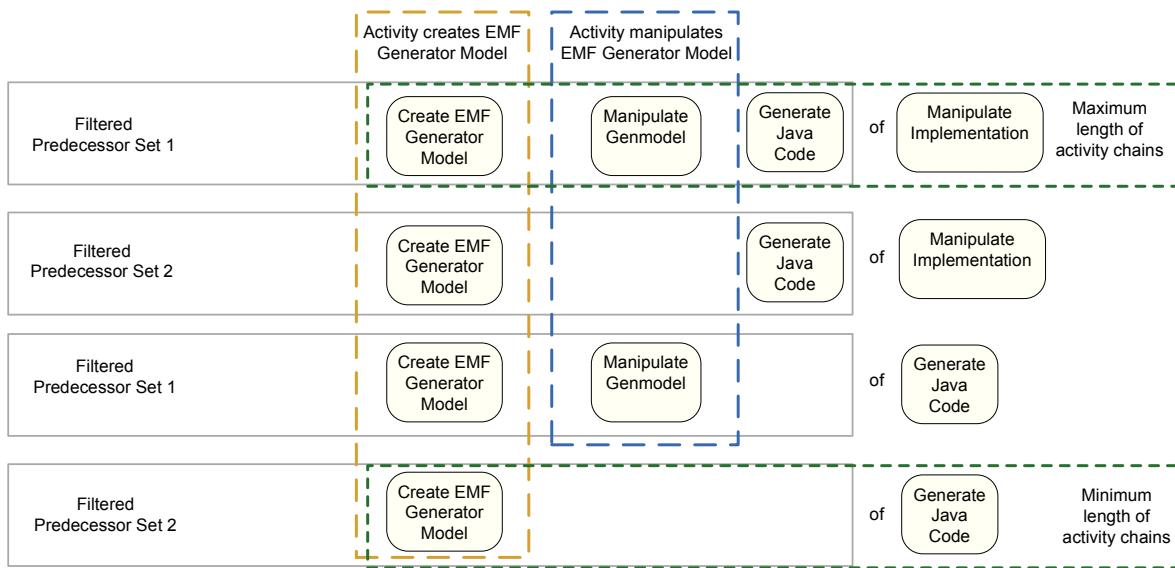


Figure 8.16.: Example assessment of lengths of activity chains for artifact pair “*EMF Generator Model*” and “*Interface Implementation*” within the *extended EMF example* (Section 2.6) that is shown in Figure 8.6. Shown are the filtered predecessor sets of activities that create or manipulate artifact “*Interface Implementation*”.

artifact) is not part of the predecessor set as described above. This is the reason why the lengths that are calculated on the basis of the size of the predecessor sets have to be increased by 1. The difference between minimum and maximum length results from the differences between the predecessor sets. Note that in some cases the first artifact in the considered pair stems from outside the MDE setting and is only used within the setting. In these cases, no activity for changing this artifact might be counted.

Thus, as indicated in Figure 8.16 the lengths of activity chains for artifact pair “*EMF Generator Model*” and “*Interface Implementation*” is 2 to 4 in the example.

The presented algorithm can be found in detail in Appendix F.3.

As described in Section 7.2.3, not all pairs of artifacts need to be considered. The current version of the presented algorithm still computes the lengths of activity chains for all pairs of artifacts (which has square complexity ($O(n^2)$)). In future, the input of the analyst might be used to only calculate lengths for relevant pairs of artifacts.

Further, the above described calculation of lengths for a given ordered pair of activities has a cubic complexity ($O(2n^3)$), only. Consequently, the length of activity chains can be calculated automatically (under the assumption that information about predecessor sets of the activities of the MDE setting is given).

9. Study Results: Employing Modeling and Analysis

As shown in the overview picture in Figure 9.1 this thesis focuses on the questions how MDE settings can influence changeability (Figure 9.1 a) and how MDE settings are interrelated with software development processes (Figure 9.1 b). These questions were discussed on a theoretical level in Chapter 3. In that context the hypotheses $H_{changeability}$ and H_{traits} , which state that MDE settings in practice differ in the studied characteristics of MDE (i.e. the influence on hard changeability concerns and the interrelation with software development processes) were formulated. In order to investigate these characteristics for MDE settings, the Software Manufacture Model language and analysis techniques were introduced in the Chapters 6 and 7. In this chapter the evaluation of the hypotheses $H_{changeability}$ and H_{traits} is approached on the basis of the introduced modeling and analysis techniques and the case studies from the first and third study (see Chapter 4).

Another focus of this thesis is on the question how evolution affects the above mentioned characteristics of MDE settings (Figure 9.1 c). So far it was theoretically discussed that substantial structural evolution can alter these characteristics (see Section 3.4.2). Further, it was shown that structural evolution and substantial structural evolution actually occur in practice and are common (see Chapter 5).

As a final evaluation step it will be investigated in Section 9.4 whether the case studies include examples, where structural evolution steps actually altered the different characteristics on MDE settings. Therefore, the results from the analysis of changeability influence and process interrelation (that will be presented in Sections 9.3 and 9.2) will be combined with the data on evolution steps from the third study (see Chapter 4).

Before these different evaluations are presented, it is first discussed in Section 9.1 how the data from the case studies was prepared for the analysis.

This chapter is partially based on [P2, P6, P4] and [P3].

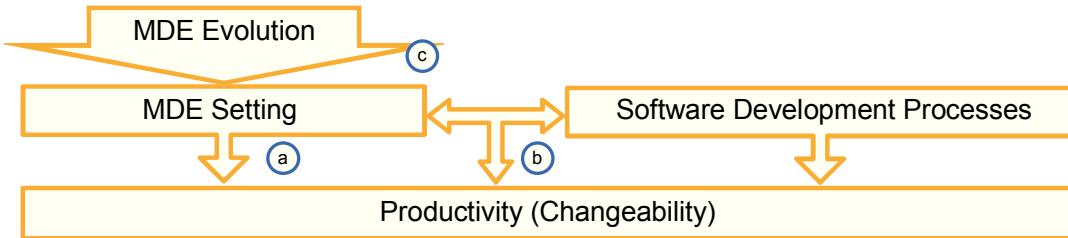


Figure 9.1.: Overview of investigated aspects of MDE

9.1. Preparation of Models for Analysis

The Software Manufacture Models on which the analysis techniques are applied in the following stem from the eleven case studies from the first and third study (see Chapter 4): BO, BRF, BW, Oberon, SIW, VC, VCat, Carmeq, Cap1, Cap2a, and Cap2b (for case study Ableton the MDE setting was not captured in detail). For BO the investigated Software Manufacture Model represents the MDE setting at the evolution stage that was captured during the first study (i.e. the MDE setting between the fourth and fifth evolution step). For the other models with captured evolution history (VC, VCat, Carmeq, Cap1, Cap2a, and Cap2b) the version that was in use at the time of the interviews is used. As described before, the models were captured in context of the interview phases, which happened partly without the Software Manufacture Model language or on the bases of the old Software Manufacture Model metamodel. In addition, most of the models were captured before the analysis techniques were introduced.

However, in the Sections 6.2.4 and 7.2 a couple of modeling conventions for the correct application of the analysis are defined. To apply the analysis, the captured models partly needed to be adapted,

such that these modeling conventions are fulfilled. In the following these preparations are described. Subsequently, possible impacts of the model preparation are discussed.

9.1.1. Model Preparation

First, the models are adapted to ensure a basic model quality. As already described in Chapter 4, some of the captured models had to be adapted to the actual version of the Software Manufacture Model metamodel. Some of the SAP case studies included activities that explicitly model the choice of a specific artifact out of a set of artifacts for further treatment (e.g. the choice of a “*floorplan template*” out of multiple instances of “*floorplan templates*” in case study Oberon). In context of the adaption to the new metamodel, such activities were removed.

Further, to ensure that activities and artifacts of similar, but different roles can actually be distinguished, some names had to be changed. For example, in case study BO it is now differentiated between “*BO model*” and “*other BO model*”, in order to differentiate between the model for the actually developed business objects and models that are used as example from previous projects.

Partly containment relations were not made explicit as links in the models (following the abstraction rules from Software Manufacture Models in Section 6.2.2). Since the corresponding abstraction technique is currently not supported by the editor, several activities were enriched with links to explicitly express these containment relations. To enable the formulation of these links, it was necessary to split some of the *internal sets*, such that containing and contained artifacts were connected to different pins.

Second, the models are adapted to fulfill the modeling conventions for the correct application of the predecessor analysis (as introduced in Section 6.2.4). In the absence of more explicit information, the currently implemented algorithm differentiates on the basis of the multiplicity of connectors whether an activity that consumes and produces instances of the same artifact role is a creating or manipulating activity for this artifact (i.e. an artifact that is created is optional input (lower multiplicity 0) and always output). With respect to the resulting modeling convention MC1 some of the connector multiplicities were adapted.

Further, a problem for the automated identification of predecessor sets is the fact that the Software Manufacture Models so far include no concept to differentiate between normal artifacts and artifacts that are used to store and report on identified errors. Thus, when error reports and their optional consumption in modeling or coding activities are modeled explicitly, this affects the identified predecessors and successors, such that cycles for correction activities are included. This in turn can affect the identified length of activity chains. To fulfill modeling convention MC2 for preventing such problems, some simple quality assurance activities or the artifacts that model the error reports were removed from the models.

Third, some modeling conventions were defined to ensure that the analysis techniques can be applied (as introduced in Section 7.2). To fulfill modeling convention MC3 the focus of most models was limited, which led to the removal of some activities in the models. This concerns some “meta” activities that focused on the manipulation of artifacts that are used to specify how automated activities are executed. For example, there were activities in case study Carmeq that describe the manipulation of the “*word macros*” for the integration of “*figures*” into the “*word standard document*”. Further, some case studies include activities for the creation of alternative results. For simplicity some of these alternatives were removed. For example, in case study Cap2b different documents might be generated based on the model (“*Result1*” - “*Result9*”). Although different generators are used, the corresponding activity chains are similar. Thus, the removal of the alternatives, does not affect the properties that can be identified for this MDE setting.

In addition, all activities that were modeled as manual or semi-automated activities were reconsidered to decide about the actual degree of automation following modeling convention MC4. Some activities that were modeled as semi-automated were changed to manual activities, when the degree of automation for these activities did not exceed an intelligent editor support.

To reach comparability of the number of activities, the models were further normalized in the degree of detail (following modeling convention MC5). On the one hand manual activities that access the same artifacts or partial artifacts that are contained by the same artifacts were combined. Especially in case studies like VC this led to a massive change, since the initially captured activities on the manipulation of the “*VC Model*” were very detailed. On the other hand automated activities that are in a chain (i.e. activities that are always executed together and where not each single activity has to be triggered explicitly by a developer) were combined. For example, the automated activities “*ConvertToDBEntry*”

and “*GenerateRuleClass*” in case study BRF were combined to one activity. In that context, also one complex semi-automated activity was split into two activities that concern different artifacts: the activity “*integrate Figures and Tables*” in case study Carmeq was split such that one activity for the integration of figures and one activity for the integration of tables were modeled.

Finally, some activities that explicitly represented workarounds were removed, to fulfill modeling convention MC6. For example, the initial model of Cap2a included an activity that represented a manual synchronization of changes, which are applied to the resulting artifacts, back to the initial model.

To summarize, the nine modeling conventions were applied to the models of the case studies. As a result the number of manual and automated activities within the Software Manufacture Models changed. Table 9.1 provides an overview about the prepared models.

Table 9.1.: Summary of prepared models of case studies from the first and third study (see Chapter 4).

	number of activities in prepared model versions	number of activities in original model version
BO	18	19
BRF	13	23
BW	10	19
Oberon	9	25
SIW	8	16
VC	10	30
VCat	10	10
Carmeq	18	25
Cap1	14	16
Cap2a	7	18
Cap2b	11	27

9.1.2. Resulting Threats to Validity of Analysis Results

Main motivation for the modeling conventions and, with it, for the applied changes is to reach a higher comparability among the captured MDE settings. From the three categories of threats to validity (Wohlin et al. [214]) that are relevant for this thesis (as discussed in Section 4.4), two are not directly affected. These are the external validity (i.e. the question for *the extent to which results can be generalized*, as discussed in Section 4.4.2) and conclusion validity (i.e. the question *whether the conclusions that are drawn on the basis of the data are correct*, which shall be discussed separately on the results of the analysis in Sections 9.2.4, 9.3.5, and 9.4.5). However, the modification of the data can have some impact on the construction validity (i.e. on the question *whether data and effects are captured appropriately*).

Therefore, it is discussed in this section, how the applied preparation influences the effects that will be analyzed on the basis of the Software Manufacture Models. This discussion is done for each of the effects that will be analyzed for the MDE settings in the remainder of this chapter. These are the occurrences of the patterns for changeability and the assessment of manual information propagation, the lengths of activity chains, and the phases and synchronization points.

Pattern matches First, the above described changes might reduce the number of pattern matches that can be identified during the analysis. This is mainly caused, by the removal of activities (when the removed activities might have been matched to pattern activities). Further, the normalization of the degree of automation and the combination of activities for the normalization of the degree of detail might lead to removed matches. For example, in case study Cap2a two activities within a fully automated transformation chain could have been matched to proto-anti-pattern *creation dependence*. These matches are no longer visible for the combined activity that describes the whole transformation chain.

However, no links or artifacts are added during the preparation. Also combined activities that are created during the normalization can only describe links that are already part of the model. Thus, while it is possible that a pattern match is substituted by a match to the combined activity, no additional pattern matches are cause.

Subsuming, pattern matches that are identified within the prepared models are not caused by the preparation of the models, but reflect effects in the actual MDE settings.

Lengths of activity chains The removal of activities (e.g. choice activities) can affect the lengths of activity chains, which might be decreased by the removed activity. Further, the lengths of activity chains can be affected by the application of modeling conventions to differentiate between manipulating and creating activities or to remove correction cycles. In all cases the assessed lengths of activity chains might be decreased, only. However, it is important to notice that the changes aim at increasing the comparability between different MDE settings. Thus, the application of the changes to all studied MDE settings, decreases differences that are caused by different modelers' decisions to include details into a Software Manufacture Model or not.

The main effect on the length of activity chains can be caused by the combination of activities during the normalization of the degree of detail. As for the other preparation steps, the length of activity chains might be decreased, only. For example, a single activity is counted instead of multiple activities that form an automated transformation chain. Again, as described in Section 7.2.3, this normalization increases comparability of different MDE settings, since individual modeler's decisions on the captured degree of detail are compensated. In addition, this normalization step ensures that the assessed lengths of activity chains actually reflect the context changes that are experienced by developers (see Section 7.2.3). Subsuming, the lengths of activity chains that will be measured are most probably shorter than for the original models, for the benefit of a better comparability of the lengths assessed for the different MDE settings. Thus, differences in the lengths reflect actual differences in the complexities developers are confronted with.

Phases and synchronization points The removal of activities changes the global foci of the Software Manufacture Models. Consequently, these changes can lead to changes in the number of global phases and global synchronization points. Similarly, potential local foci might be removed. However, no potential local foci are added (since no activities are added). Thus, the local foci that are identified and considered as relevant during the analysis, including the main focus, are part of the analyzed MDE settings. Therefore, the comparison of the MDE settings shall mainly base on the main foci.

The local phases and synchronization points within the identified local foci are not affected by most preparation steps. Only, the combination of activities during the normalization of the degree of detail, can affect the phases and synchronization points. The number of synchronization points might be decreased, when several synchronization points are combined to one activity (e.g. within a fully automated transformation chain). However, this decrease in synchronization points does not affect the number of phases. In contrast, the combination of multiple manual activities within a phase might lead to the removal of this phase (since a single activity substitutes a set of parallel activities). However, in this case the combined activity is rated as extensive manual activity and therefore occurs as miniphase.

Subsuming, the number of phases that can be identified for the main focus and other local foci is not affected by the preparation of the models.

Conclusion of threats To sum up, the preparation of the models neither introduces additional pattern matches nor increases the number of phases and lengths of activity chains. The changes in the assessed lengths of activity chains are required to reach comparability. Further, the consideration of main local foci compensates that the applied changes might affect assessed phases within the global focus of an MDE setting. Finally, it is possible that some pattern matches of removed activities are lost.

However, none of the identified effects (e.g. pattern occurrences, overlong activity chains, or huge numbers of phases) can be caused by the applied model preparation. Further, the preparation has carefully been applied by the modeler of the Software Manufacture Models on the basis of the interview records, in order to ensure that no relevant activities are removed from the models. This decreases the probability that relevant effects (e.g. pattern matches to relevant parts of the MDE settings) are no longer captured in the data.

9.2. Findings on Changeability

This section aims at evaluating whether MDE setting in practice differ in their influence on changeability. In this context, it is also evaluated whether the proto-patterns that are introduced in Section 7.1 are relevant in practice.

For this evaluation occurrences of the proto-patterns were searched in the case studies from the first and third study (see Chapter 4). It is discussed to which amount risks of proto-anti-patterns manifest in practice. Subsequently, the proto-patterns are used to investigate an open source case study. Finally, the hypothesis $H_{changeability}$ from Section 3.5.1 is evaluated and threats to validity are discussed.

9.2.1. Changeability Analysis of Case Studies

For the analysis how hard changeability concerns are affected, matches of the four proto-patterns that were presented in Section 7.1 were searched with the pattern matcher (see Section 8.2) on the Software Manufacture Models of the eleven case studies from the first and third study (see Chapter 4) that were prepared for the analysis as described in Section 9.1.

In the following, an overview of the results of these pattern matches is provided. First, the matches of the two proto-anti-patterns *subsequent adjustment* and *creation dependence* are summarized. Subsequently, one of the matches of the proto-anti-pattern *creation dependence* is discussed in detail. Finally, the matches of the two proto-pattern *split manufacture* and *anchor* are summarized and further identified structures that support changeability are listed.

Occurrence of Proto-Anti-Pattern

As summarized in Table 9.2 and illustrated in Figure 9.2 the two proto-anti-patterns *subsequent adjustment* and *creation dependence* occur several times in the eleven case studies.

Table 9.2.: Summary of identified matches for proto-patterns from Section 7.1 in the case studies from the first and third study (see Chapter 4).

	Subsequent Adjustment	Creation Dependence	De-	Split	Manu-	Anchor
BO	3	1				
BRF						
BW						
Oberon	5					1
SIW	3			2		
VC	3	1				
VCat	3			2		
Carmeq						
Cap1	1	1				
Cap2a	1					
Cap2b		4				
All	19	7	4		1	
Average	1.72	0.63	0.36		0.09	

Subsequent adjustment was matched 19 times within 7 of the eleven case studies. For 13 of these occurrences mitigating factors that prevent the manifestation of the associated risks could be identified. A first observed mitigating factor is that a change of the *base artifact* is not probable. This holds for 9 of the matches. The *base artifact* is a template (i.e. reused) in 4 of the matches. In one of the matches a mitigating factor is that only a minimal amount of content is lost. Further, for one of the matches the mitigating factor is that there is no need to maintain the resulting artifacts throughout the whole live cycle. A last mitigating factor, which was observed for two matches within the same case study, is

9. Study Results: Employing Modeling and Analysis

that the MDE setting includes an alternative activity for the one matched to the pattern activity *initial creation*. Consequently, a matched activity is never executed to implement a change.

However, for six of the matches (in 4 case studies) no mitigating factors could be identified. Here the records of the interviews partly included hints that the associated loss of content already occurred or that developers use a copy and paste workaround to preserve manually created content. For example, in case study Cap1 a new execution of activity “*Create Constructionmodel*” to propagate a change of artifact “*Specification model*” can lead to the destruction of changes that were applied manually to artifact “*Structural Models*” during activity “*Manipulate Constructionmodel*” before.

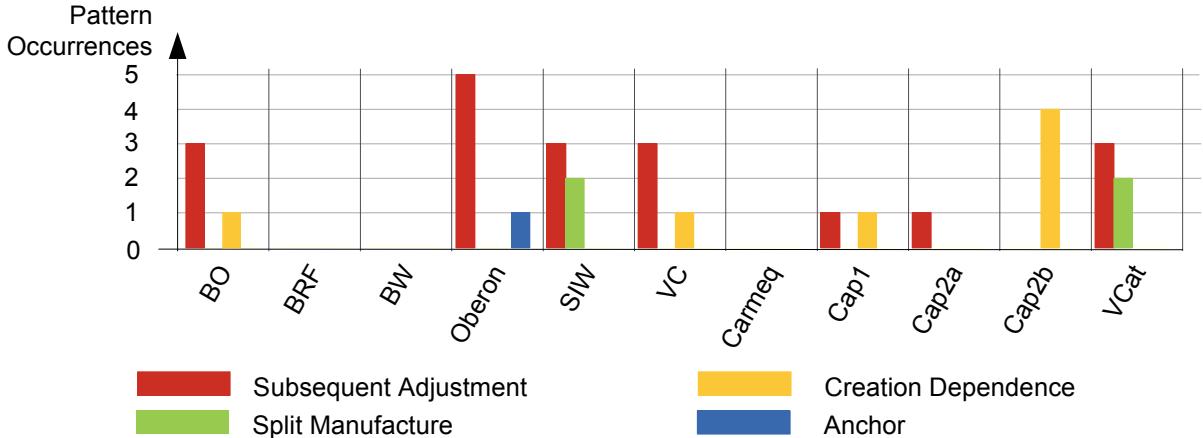


Figure 9.2.: Occurrences of patterns on changeability concerns within the case studies.

Creation dependence was matched 7 times in 4 of the eleven case studies. For 5 of these matches no subsequent match of proto-anti-pattern *subsequent adjustment* could be identified. However, for two of the matches (in two different case studies) no mitigating factor can be identified. Here the pattern occurrence amplifies the effects of occurrences of proto-anti-pattern *subsequent adjustment*. For example, in case study VC the occurrence of *creation dependence* implies that changes in artifact “*DataService*” are additional triggers for a new execution of activity “*Generate process Context*”, which can lead to loss of content of artifact “*VC Model*” if it was semi-automatically enriched during activity “*Create Service Component*” before.

Prediction of Risks on the Basis of Proto-Anti-Patterns

While mitigating factors can be identified for the most of the occurrences of *subsequent adjustment* and *creation dependence*, such “harmless” pattern matches can be useful to predict what evolution steps on the MDE setting can lead to problems. Following it is shown on an example, how a match of *creation dependence* in one of the case studies was used to predict future risk.

The final automated generation activity in case study Cap2a is built as a transformation chain. Within this transformation chain the first three activities translate and enrich the “*Model*” stepwise into different intermediate models. During these transformations a reference to another input artifact is propagated through each intermediate model. The intermediate transformation activities match to the proto-anti-pattern *creation dependence* (i.e. the respective result of a transformation steps is always created together with a reference from this result to the “*Plain Text Part*”). Finally two further transformations are used to produce the two resulting artifacts, respectively. For the creation of the “*Result2*” the “*Plain Text Part*” is required, while the transformation step that produces the “*Result1*” ignores the references to the “*Plain Text Part*”.

Due to the match of *creation dependence* to the first three transformations, not only a change in the “*Model*”, but also a change in the “*Plain Text Part*” that affects the target of the reference, can lead to a new execution of the whole transformation chain. This is necessary and valid to ensure that the change is propagated to the “*Result2*”. However, a side effect of this transformation is that also the “*Result1*” is generated new. Thus a change in the “*Plain Text Part*” can trigger the new generation of the “*Result1*”, although this artifact does not depend on the content of the “*Plain Text Part*”. Currently,

this additional trigger does not lead to problems, since the generation of the “*Result1*” is fully automated and does not require further manual actions. Thus, the occurrence of *creation dependence* does not lead to a problem, since no negative effects on changeability exist, which might be amplified.

However, with the knowledge about this pattern occurrence it is possible to predict what change of the MDE setting will result in the occurrence of a problematic situation. In such a hypothetical scenario, the standard requirements on the “*Result1*” (e.g. concerning the layout) change in future. A solution is to apply necessary changes manually, which is directly available and probably easier to implement than a change in the transformation. However, while the technology that is used for the transformation chain does not support protected regions or similar techniques to prevent that hard changeability concerns are affected, this introduction of a manual activity for manipulating the “*Result1*” leads to an occurrence of pattern *subsequent adjustment*. This scenario would lead to a situation, where the risks that are associated to the *creation dependence* (i.e. the amplification of a changeability problem), which is already part of the current MDE setting, manifests. Thus, a change in the “*Plain Text Part*”’s can trigger the new execution of the transformation chain and thus lead to the destruction of the manually created content – although the “*Plain Text Part*” has no effect on the content of the “*Result1*”.

Subsuming, only a single change to the currently used MDE setting is sufficient to get into this problematic situation: the need to manually apply changes to the “*Result1*”. The example shows that even occurrences of proto-anti-patterns without manifested problem can be helpful, since they allow to predict what concrete evolution steps on the MDE setting will lead to the manifestation of the problems.

Occurrence of Positive Influences on Changeability

While for the proto-anti-patterns a relative high number of matches could be identified, only 4 matches of *split manufacture* in two case studies and one match of *anchor* were identified. This includes the occurrences that were the original of these two proto-patterns (i.e. the occurrences that led to the identification of these structures). Thus, for proto-pattern *anchor* no additional occurrence could be identified. Similarly, for proto-pattern *split manufacture* only one additional case study included additional occurrences.

However, further investigation of the interview records revealed that other strategies were applied to support changeability. For example, in Cap1 a generation gap pattern [78] is applied to ensure that manually created content of the *code* is not overwritten, when the code generation is reapplied to propagate changes from the “*Construction Model*”. In Cap2b the word templates that are used for the generation include specified regions that are target of the generation. Manually created content that is created outside these regions is not overwritten during generation.

9.2.2. Changeability Analysis of an Open Source Example

To demonstrate how the pattern can be used in combination for analyzing a Software Manufacture Model, the generation of java code within the *EMF example* (introduced in Section 2.6) is used here as a very simple and commonly accessible case study¹. This set of MDE activities was applied during a project at the *system analysis and modeling* research group², where up to five student assistances were employed.

Above in Fig. 9.3 the activities are shown as they were initially perceived by the students. There are four activities (omitting the creation of the *ecore model* here). Activity *create EMF generator model* takes the *ecore model*, which is basically an EMF version of a class diagram that describes a domain, and generates an *EMF generator model*, with a reference to the used *ecore model*. The *EMF generator model* contains configurations, which further have to be adapted manually (activity *manipulate generator model*), e.g. to define the name space by filling attribute *Base Package*. Activity *generate java code* takes the *EMF generator model* together with the *ecore model* and generates *adapter factories*, *interfaces*, and *interface implementations* for the concepts specified in the *ecore model*. Subsequently, the *interface implementations* are manipulated manually, e.g. to define default values for attributes. The presented activities only show an excerpt, as is also possible to generate edit, editor, and tests code.

¹In the here described project EMF 2.6 was used.

²https://www.hpi.uni-potsdam.de/giese/projects/dtr_virtual_multi_user_software_prototypes.html?L=1 (last access at November 9th, 2013)

9. Study Results: Employing Modeling and Analysis

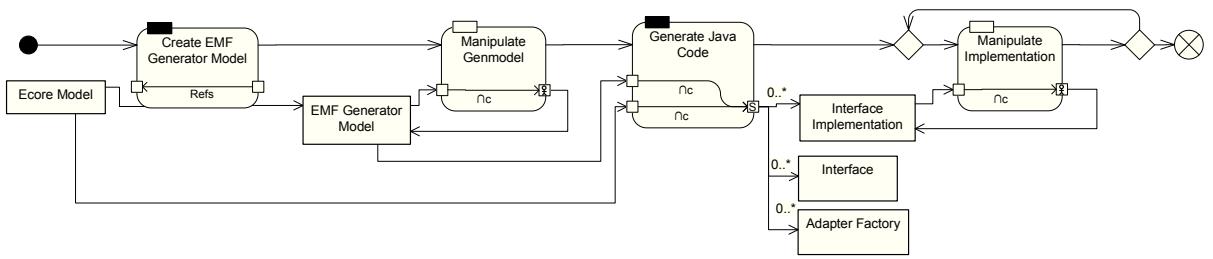


Figure 9.3.: Simplified version of the *EMF example* with initially perceived Software Manufacture Model activities (repetition of Figure 6.25)

When analyzing Software Manufacture Model activities in Fig. 9.3 both Software Manufacture Model proto-anti-patterns can be identified (as already indicated in Figure 9.3). First, the proto-anti-pattern *subsequent adjustment* can be matched.

Actually, the students had to experience the loss of the manually added content, which is associated to the proto-anti-pattern, multiple times. Interestingly, some of the students recognized that the generation supports protected regions (marked with the flag “@generated NOT”). In fact, activity *generate java code* looks like shown in Fig. 9.4(a). However, there was no communication about this opportunity and the students still lost code. It took a while until they started to communicate that the protected regions should be respected. This shows that developer guidelines can be as important for a successful solution as the technical opportunities.

The students ended up improving their process, such that activity *manipulate implementation* complies with Fig. 9.4(b). The resulting solution conforms to the Software Manufacture Model proto-pattern *split manufacture*: *generate java code* matches to pattern activity *regenerate*, *manipulate implementation* matches to *manual completion*, *interface implementation* matches to *modified artifact*, and *Generated NOT* (referring to the marked parts of the artifact) matches to *artifact detail*.

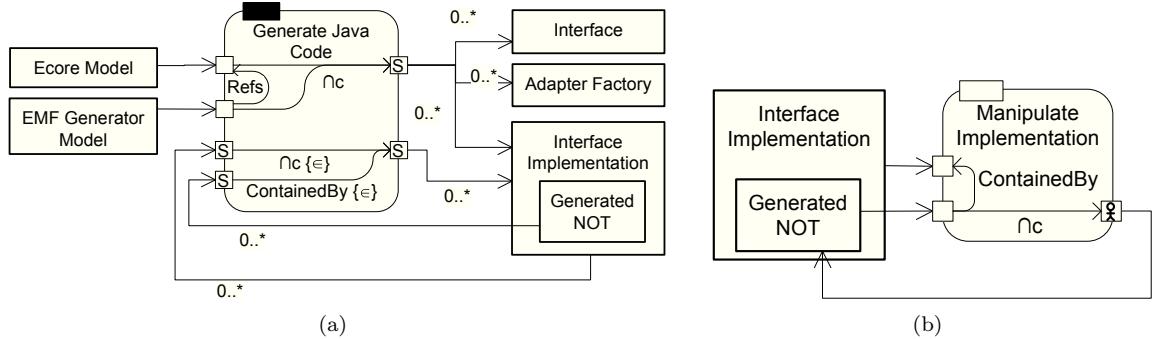


Figure 9.4.: Corrected Software Manufacture Model activities of the *EMF example* (as already shown in Figure 8.6)

Second, the activity *create EMF generator model* (Fig. 9.3) matches to pattern activity *creation* of proto-pattern *creation dependence*, where *ecore model* matches to artifact role *necessary artifact* and *EMF generator model* matches to artifact role *depending artifact*. Actually *EMF generator model* depends strongly on the *ecore model* and *EMF* triggers an update to changes in the *EMF generator model* automatically when the referenced *ecore model* is changed. For this example it was not possible to specify how this update activity looks like, as no documentation could be found, which parts of the *EMF generator model* are preserved. This preservation seems to affect at least the attribute *Base Package*.

However, there is a change that leads to a destruction of the reference from the *EMF generator model* to the *ecore model*: changing the file name, which leads to the need for a new application of *create EMF generator model*. In this case the attribute *Base Package* is not preserved and this information is lost. Obviously, there is potential to resolve this drawback in future, as information can be preserved as long as restoring the reference is not necessary. It seems that the file name of the *ecore model* is in most

projects not changed, which explains why this Software Manufacture Model proto-anti-pattern is still kept in EMF. However, if developers work in a fashion that requires regular refactoring the identified restriction can be experienced as a reduction of changeability.

Summing up, the EMF case study shows how the presented pattern can be used to identify risks for changeability. Further, the example illustrates that it can be important to be aware of pattern occurrences during development.

9.2.3. Evaluation of Hypothesis $H_{\text{changeability}}$

In Section 3.5.1 the hypothesis $H_{\text{changeability}}$ is formulated. The hypothesis states that MDE settings in practice differ in their influence on changeability (“*MDE settings in practice vary considerably in their influences on changeability. Thus, there are MDE settings that strongly affect the hard changeability concern unexpected loss of content (in more than 5% of all MDE settings) and also MDE settings that do not affect the hard changeability concern unexpected loss of content (in more than 5% of all MDE settings).*”). To evaluate the hypothesis $H_{\text{changeability}}$ it shall be examined to what percentage MDE setting form practice include occurrences of the proto-anti-pattern *subsequent adjustment*. As defined in hypothesis $H_{\text{changeability}}$ a phenomenon is considered to be not seldom, if it exists in more than 5% of all MDE settings.

Two pairs of hypothesis shall be tested. One the one hand, it shall be tested whether MDE settings with occurrences of *subsequent adjustment* are seldom. Correspondingly, the null hypothesis h_{0_match} is that the percentage of MDE settings with occurrences of *subsequent adjustment* is below or equal to 10%, while the alternative hypothesis h_{1_match} is that the percentage of MDE settings with occurrences of *subsequent adjustment* is greater than 10%. On the other hand, it shall be tested whether MDE settings without occurrences of *subsequent adjustment* are seldom. Correspondingly, the null hypothesis $h_{0_noMatch}$ is that the percentage of MDE settings without occurrences of *subsequent adjustment* is below or equal to 10%, while the alternative hypothesis $h_{1_noMatch}$ is that the percentage of MDE settings without occurrences of *subsequent adjustment* is greater than 10%. The tested percentage of 10% substitutes the required 5% here, in the interest to approach the actual percentage. When h_{0_match} or $h_{0_noMatch}$ can be rejected this is an even stronger confirmation of $H_{\text{changeability}}$.

To test the hypotheses the binomial test is used. The test allows checking for small samples whether data points conform to one of two categories for a certain percentage of the data set. These categories are “*subsequent adjustment pattern matched*” and “*subsequent adjustment pattern not matched*”. For the test a significance level of 5% is applied. As summarized in Table 9.3 the tests are performed for a 10% percentage as well as a 20% percentage.

Table 9.3.: Results of binomial test to check whether probabilities for MDE setting with and without occurrences of the proto-anti-pattern *subsequent adjustment* exceed 10%

	# occur- rences	sample size	h_0	h_1	p-value	95% con- fidence in- terval	Result
occurrences included	7	11	$p \leq 10\%$	$p > 10\%$	2.29e-05	0.3498 - 1	h_{0_match} is rejected
	7	11	$p \leq 20\%$	$p > 20\%$	0.001965	0.3498 - 1	h_0 is re- jected
no occurrences included	4	11	$p \leq 10\%$	$p > 10\%$	0.01853	0.1351 - 1	$h_{0_noMatch}$ is rejected
	4	11	$p \leq 20\%$	$p > 20\%$	0.1611	0.1351 - 1	h_0 is plau- sible

Hypothesis h_{0_match} can be rejected. The test revealed that the probability that 7 MDE settings with occurrences of *subsequent adjustment* are identified for a sample size of 11, while the expected percentage is below or equal to 20%, is less than 5% ($p\text{-value} = 0.001965$). Consequently, h_{0_match} is not plausible.

Also hypothesis $h_{0_noMatch}$ can be rejected. The test revealed that the probability that 4 MDE settings without occurrences of *subsequent adjustment* are identified for a sample size of 11, while the expected percentage is below or equal to 10%, is less than 5% (p-value = 0.01853). Consequently, $h_{0_noMatch}$ is not plausible. Further the test showed that the probability that 4 MDE settings without occurrences of *subsequent adjustment* are identified for a sample size of 11, while the expected percentage is below or equal to 20%, is greater than 5% (p-value = 0.1611).

Subsuming, there are to a comparable extent MDE settings that affect the hard changeability concern “*unexpected loss of content*” as well as MDE settings that do not affect the hard changeability concern “*unexpected loss of content*”. Therefore, hypothesis $H_{changeability}$ is supported by the results of this analysis. Table 9.4 summarizes the result.

Table 9.4.: Summary of hypothesis on changeability

Hypotheses	Percentage of MDE settings with matches of proto-anti-pattern subsequent adjustment	Percentage of MDE settings without matches of proto-anti-pattern subsequent adjustment	
$H_{changeability}$ MDE settings in practice vary considerably in their influences on changeability. Thus, there are MDE settings that strongly affect the hard changeability concern unexpected loss of content (in more than 5% of all MDE settings) and also MDE settings that do not affect the hard changeability concern unexpected loss of content (in more than 5% of all MDE settings).	$p > 20\%$	$p > 10\%$	✓

9.2.4. Threats to Validity

In the following, threats to conclusion validity are discussed for the presented evaluation of hypothesis $H_{changeability}$ (for the discussion of other threats to validity see Section 4.4). With 11 MDE settings, the data set is too small to make resilient statements on the frequency of MDE settings with (or without) occurrences of the proto-anti-pattern. Fortunately, the MDE settings stem from four different companies and occurrences of the proto-anti-pattern were found in different companies, too. As discussed in Section 9.2.3, this allows concluding that the proto-anti-pattern exists in practice and are not company specific. Similarly, the identification of 3 MDE settings in 2 companies without matches of the two proto-anti-pattern (4 MDE settings in 3 companies without matches of *subsequent adjustment*), is sufficient to conclude that MDE settings without matches of the proto-anti-pattern exist in practice and in different companies, too. Thus, the data is sufficient for the evaluation of hypothesis $H_{changeability}$.

9.3. Findings on Process Interrelation

In the hypotheses formulated in Section 3.5.2 it is stated that MDE settings in practice differ in a way that for each of the MDE traits some MDE settings include process-relevant manifestations and some MDE settings include process-neutral manifestations of this trait.

In this section, these hypotheses are evaluated. Therefore, the analysis methods presented in Section 7.2 are applied on the eleven models from the first and third case study, which have been prepared for the analysis as described in Section 9.1.

9.3.1. Phases and Synchronization Points in Case Studies

In order to evaluate whether MDE settings in practice have different or rather similar characteristic the analysis of synchronization points and phases, which was introduced in Section 7.2.1 was applied

to the prepared Software Manufacture Models. In addition, to the automated part of the analysis, two manual tasks were necessary. First, activities that are potential miniphases had to be identified. Second it had to be evaluated evaluate, which of the identified local foci represent meaningful parts of the MDE setting.

For the identification of potential *miniphases*, the models and the records from the interviews were used to search for extensive manual activities. 29 modeling activities, 9 activities on natural language documents, 9 coding activities, and 5 quality assurance activities (reviews and testing activities) were identified as potential miniphases, altogether. Examples are the specification of a concept document (activity “*Create Concept Document*”) in case study Carmeq, which is done by a team of experts, the activities for definition of layout and data retrieval in case study BW (e.g. activities “*Define Layout*” and “*Model Data*”), the enrichment of the “*Construction Model*” (which consists of different UML models for structural and behavioral aspects) or the subsequent implementation of code bases in case study Cap1 (activities “*Manipulate Constructionmodel*” and “*Write Code*”), or the modification of rules and decision tables in BRF (activity “*ChangeRuleSet*”). All these activities represent work on multiple artifacts of the same role or artifacts that can be divided in multiple sub artifacts and thus can be subject to division of work between multiple developers.

The decision, which of the automatically identified local foci are considered as relevant and meaningful, was done on the basis of the interview records, too. On that basis it was also evaluated whether and which one of the local foci represents a main development path.

Data on Local Foci

Table 9.5 summarizes the identified and selected local and main foci for the eleven case studies. As examples, global phases, synchronization points, and local foci of the three case studies Cap1, BW, and SIW are illustrated in Figure 9.5. For case study Cap1 the third focus is the main focus, since it captures the creation of the actual software product. For the same reason, the main focus of case study SIW is the first local focus. Also the main focus of BW is the first local focus is (here the activities that are left out of the local focus are only preparation activities).

In only some of the case studies, the sum of the different selected foci covers the complete MDE setting (i.e. each activity of this MDE setting is at least part of one local focus). For example, the first and third foci of case study BW cover two different aspects of the development. They share only the automated interpretation activity that is the global synchronization point at the end of the MDE setting. In other case studies the selected foci do not completely cover the corresponding MDE setting. For example, in Cap1 a validation activity is part of the global MDE setting, but not included in one of the three selected foci. Another reason that activities are not in any of the selected local foci, is that these activities are alternatives for activities that are part of the main focus. For example, in SIW the two activities that are not part of the main focus are activities for the regeneration of “code” and “proxy”. These activities are alternatives for activities that can be used for the initial creation of both products. Finally, as in case study BO activities that are used to embed the product of the MDE setting into a context (e.g. activity “*Test Scenarios*”) might not be part of a local focus.

Different local foci of the same MDE setting can have different relations to each other. As described above, they can represent different groups of tasks that have to be performed to create a common result (e.g. as the first and third focus in BW). Alternatively, different foci can represent the creation of different results. For example in Cap1 the three foci are split into the creation of the “*UI-Prototype*”, the “*Specification*” (as documentation), and the software product. Finally, in some cases one local focus is local to another local focus. For example, the second local focus in SIW is local to the first local focus. This more detailed focus reveals that two of the three activities in the first phase have a strict order. Similarly, the second focus in case study BW is local to the first focus. Here the activities for the specification of the data transformation are removed.

Data on Synchronization Points

Figure 9.6 summarizes the identified global synchronization points and phases. Some of the manual synchronization points are identified as miniphases. All in all eleven global synchronization points, four of them miniphases, were identified. As specified in Section 7.2.1, a miniphase that is located directly

9. Study Results: Employing Modeling and Analysis

Table 9.5.: Summary of prepared models of case studies from the first and third study (see Chapter 4).

local focus			size (number of activities)	is main focus
BO	a	Preparation of new (extended) templates for the development	4	
	b	Main development path	11	✓
BRF		Global path	13	✓
	a	Preparation of process for configuration	8	
BW	a	Development of layout of reports including transformation of data	8	
	b	Development of layout of reports (local to development of layout of reports including transformation of data)	6	
	c	Configuration of data retrieval	3	
Oberon	a	Main path	8	✓
SIW	a	Main path	6	
	b	Local to main path: preparation of project	5	✓
VC		Global path	10	✓
VCat	a	Main path	7	✓
CarmeQ	a	Work on standard document without figures and tables	11	
	b	Main path	15	✓
	c	Creation of tables	11	
	d	Creation of figures	11	
Cap1	a	Development of UI-Prototype (which is later reused to write code)	4	
	b	Development of Specification as additional product	7	
	c	Main development path	9	✓
Cap2a		Global path	7	✓
Cap2b	a	Development of Result2	4	
	b	Development of Result3	4	✓

before or behind another phase (or miniphase) is counted as part of this phase in the following. Otherwise the miniphase is counted as own phase.

Thus, in three of the case studies (BRF, Carmeq, and Cap2a) two global phases can be identified. In only one of these cases (BRF) both phases are not miniphases, but have a comparable size. In Figure 9.7 for each case study the main local focus is summarized. For three of the case studies (BRF, VC, and Cap2a) no local focus was identified as main focus. In these cases the whole MDE setting is the main focus (i.e. global synchronization points and phases are repeated in Figure 9.7).

All in all 32 (local and global) synchronization points can be identified for the main foci. 12 of the synchronization points are miniphases. The resulting number of phases for the eleven main foci is 22. From these eleven main foci three are split into more than two (i.e. three or four) phases (BO, Carmeq, and Cap1). Four of the cases have a single phase, only. Miniphases represent a relevant part of the phases. 8 of 22 phases in the main foci consist of miniphases, only and further three phases are a combination of multiple phases including one or more miniphases.

The Tables 9.6 and 9.7 summarize data about the identified synchronization points and phases. Details on the identified synchronization points and phases can be found in Appendix G.4.

Excursus: observed effects on the process

Altogether there are less global phases than phases in the main foci. Case study BRF is the only example, where the overall setting is clearly split into two phases, which are not miniphases. Actually, this split of the MDE setting was created by intention. The two phases are used by different developers with different skills. While the first part of the setting is used for development of the main part of the software product within SAP, the second part of the setting is used for the customization of the setting, which can be done by consultants or even experts at the customer's company.

Also for the case studies Cap1 and Carmeq, which have between three and four phases in their main focus, the split of phases is reflected in the processes and team structure. While the first of the four

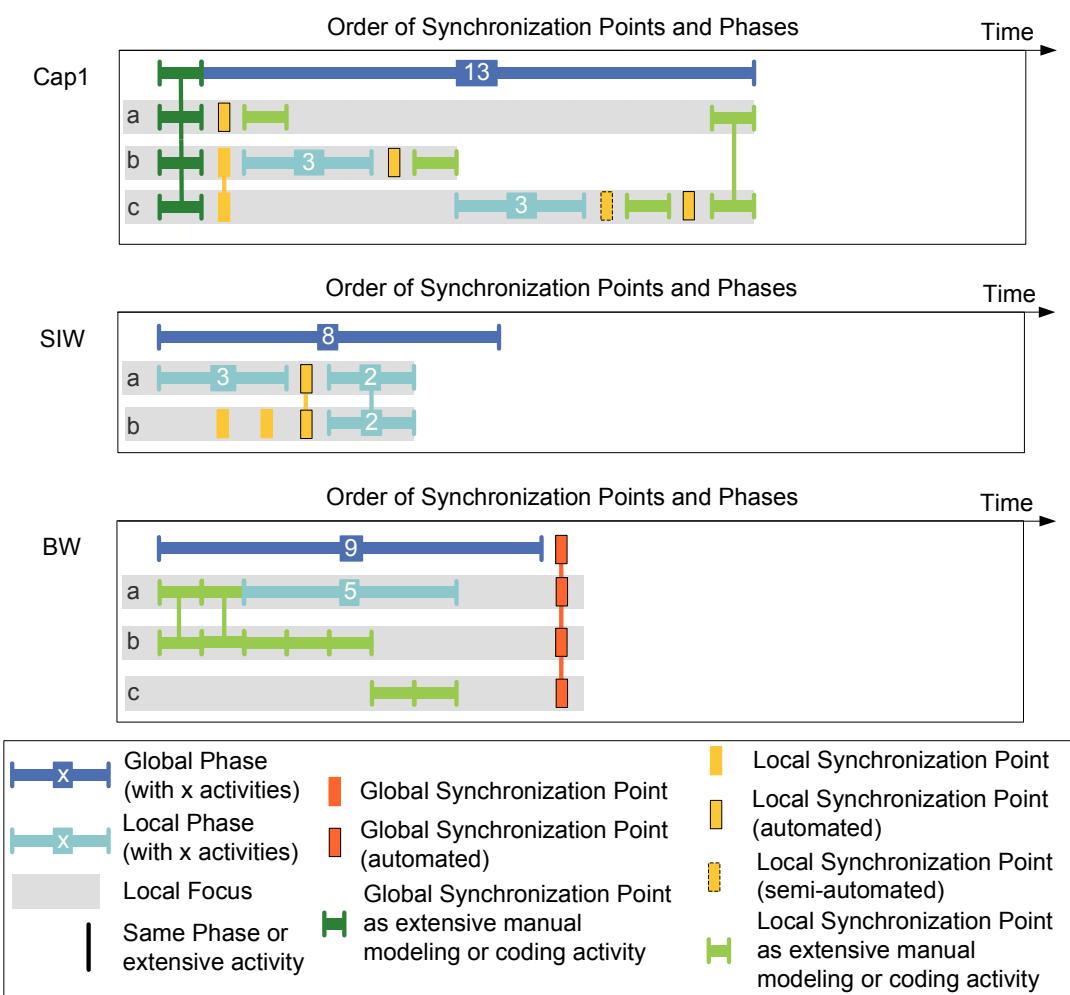


Figure 9.5.: Order of synchronization points and phases for the case studies Cap1, BW, and SIW

Table 9.6.: Summary of *synchronization points* in the case studies from the first and third study (see Chapter 4).

	# global synchronization points	# synchronization points in main focus	# global phases	# local phases in main focus
BO	0	6	1	3
BRF	1	1	2	2
BW	1	3	1	1
Oberon	1	3	1	1
SIW	0	1	1	2
VC	0	0	1	1
VCat	0	2	1	1
Carmeq	2	3	2	3
Cap1	1	6	1	4
Cap2a	3	3	2	2
Cap2b	2	4	1	2
All	11	32	14	22
Average	1	2.9	1.27	2

9. Study Results: Employing Modeling and Analysis

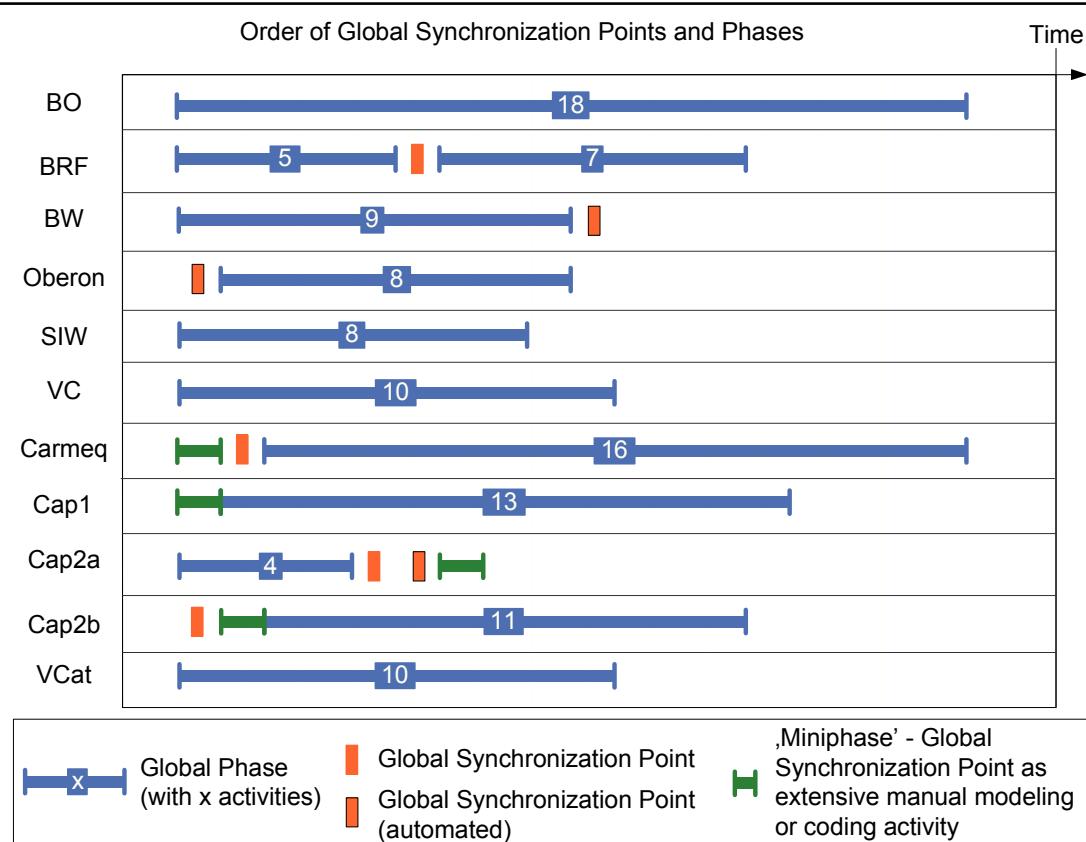


Figure 9.6.: Order of global synchronization points and phases in the different case studies

phases in the main focus of case study Cap1 is only seldom used for the development of new releases, the other three are regularly used to implement the respective changes. Activities in the first of these three phases are organized around a model that captures the specification of the required system version. In the next phase the architecture and behavior of the system are modeled and finally the last phase includes work on the code. The developers came to terms with the phases by building the release cycles around them. At each time three releases are in development. Each release is maintained by an own team and in a different phase (the most current one is in the last phase, the next in the phase that is second to the last, and so on).

In Carmeq the phases of the main focus reflect a similar effect as in case study BRF. The first and second phases in the main focus are performed by different teams with different skills and responsibilities. Within the local focus Carmeq c, the third phase of the main focus is further split into two phases. The first “subphase” of this third phase is performed by the team that is responsible for the second phase of the main focus, too. The second “subphase” of the third phase is performed by a third team (which has different skills and responsibilities, too).

Subsuming, it can be observed that a split of phases is in some cases reflected in the team structure. However, this is not a necessary effect. For example, for BO, which has three phases in the main focus, too, such a split of the team structure was not recorded.

The definition of a synchronization point does not differentiate between manual and automated tasks. Actually, from the ten sets of synchronization points that split phases in the main foci, 3 consist of one or more automated activities, one is semi-automated activity, four consist of one or more manual activities, and in two cases manual as well as automated activities are involved. Further, the case studies reveal that an automated activity is not necessarily a synchronization point. Out of 34 automated and 17 semi-automated activities in the models, only one semi-automated and nine automated activities are synchronization points in the main foci of the case studies.

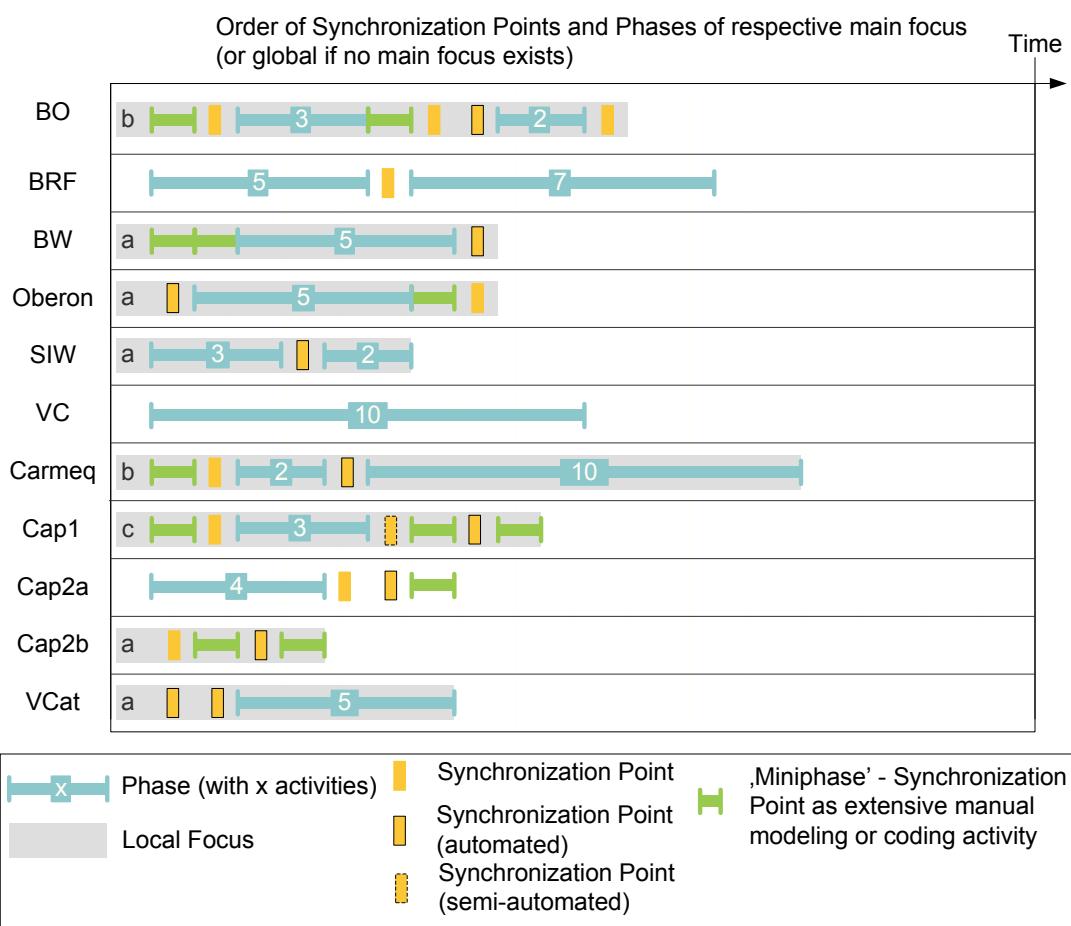


Figure 9.7.: Order of synchronization points and phases of respective main focus (of global setting where no main local focus was identified)

Evaluation of Hypothesis H_{phases}

In the following it is discussed whether the presented data support hypothesis H_{phases} (“*In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait phases, in more than 5% of all MDE settings, each.*”).

Process-neutral and process-relevant manifestations The numbers of synchronization points per main focus of the settings differ between 0 (in VC) and 6 (in BO or Cap1). Basically the number of synchronization points provides a hint on how strict the order of activities is. Thus, in a setting with multiple synchronization points in the main focus developers working in the same team are enforced to synchronize (e.g. as in the example of Cap1).

Synchronization points have different impacts on phases. For example, in BRF a single synchronization point splits the setting into two phases. In contrast, in the main phase of Oberon three synchronization points frame a single phase. Several synchronization points lead to no new phases, since they are located at the start of the very end of their MDE settings: three of the 11 MDE settings end with a synchronization point that is not a miniphase and three of the 11 MDE settings start with a synchronization point that is not a miniphase.

The number of phases for the respective main focus differs between 1 and 4. As it was discussed in Section 3.3.3 and illustrated with anecdotal evidence within the excursus above, the existence of three or four phases can lead to or reflect splits in the team structure and even release cycles. Thus, the existence of three or more phases can be considered as process-relevant manifestation of the MDE trait *phases*. In contrast an MDE setting with a single phase, such as VC or VCat is neutral in this respect

Table 9.7.: Summary of *synchronization points* within local foci of the case studies from the first and third study (see Chapter 4).

	local focus	# global synchronization points	# local synchronization points	# local phase per focus
BO	a	0	4	1
	b	0	6	3
BRF	a	1	2	2
BW	a	1	2	1
	b	1	5	1
	c	1	2	1
Oberon	a	1	2	1
SIW	a	0	1	2
	b	0	3	1
VCat	a	0	2	1
CarmeQ	a	2	1	2
	b	2	1	3
	c	2	5	4
	d	2	5	4
Cap1	a	1	3	2
	b	1	3	3
	c	1	5	4
Cap2b	a	2	2	2
	b	2	2	2
All		11	56	40
Average		1	2.94	2.1

(process-neutral manifestation of the MDE trait *phases*). The summarization in table 9.6 reveals that for four of the 11 captured MDE settings the number of phases in the main focus is one, while for three of the 11 captured MDE settings the number of phases in the main focus is three or four, actually. Thus, there are at least four process-neutral manifestations and at least three process-relevant manifestations of the MDE trait *phases* within the 11 case studies.

Hypothesis testing In this section the hypothesis H_{phases} is evaluated. As defined for hypothesis H_{phases} a phenomenon is considered to be not seldom, if it exists in more than 5% of all MDE settings. On the one hand, it is tested whether the percentage of process-neutral manifestations of MDE trait *number of phases* exceeds 5%. Therefore, the null hypothesis $h_{0_neutral}$ is that the percentage of process-neutral manifestations is below or equal to 5%. The corresponding alternative hypothesis $h_{1_neutral}$ is that the percentage of process-neutral manifestations is greater than 5%. On the other hand, it is tested whether the percentage of process-relevant manifestations of MDE trait *number of phases* exceeds 5%. Correspondingly, the null hypothesis $h_{0_relevant}$ is that the percentage of process-relevant manifestations is below or equal to 5%, while the alternative hypothesis $h_{1_relevant}$ is that the percentage of process-relevant manifestations is greater than 5%.

To test these hypotheses the binomial test is applied. This test is used for small samples to check for two categories whether data points that conform to one of the two categories occur to a certain percentage within the data set. These categories are “*process-neutral*” and “*not process-neutral*” for $h_{0_neutral}$ and “*process-relevant*” and “*not process-relevant*” for $h_{0_relevant}$. The significance level that is chosen for the tests is 5%. As summarized in Table 9.8, the test was not only performed for a 5% percentage of process-neutral and process-relevant manifestations (which is sufficient for the check of the hypothesis H_{phases}), but also for a 10% percentage.

For process-neutral manifestations of MDE trait *phases* the null hypothesis can be rejected. The probability that 4 process-neutral manifestations are identified for a sample size of 11, while the percentage of process-neutral manifestations is below or equal to 10%, is less than 5% ($p\text{-value} = 0.01853$). Thus,

$h_{0_neutral}$ is not plausible. For the process-relevant manifestations the null hypothesis can be rejected, too. Here, the probability that 3 process-relevant manifestations are identified for a sample size of 11, while the percentage of process-relevant manifestations is below or equal to 5%, is less than 5% (p-value = 0.01524). Thus, $h_{0_relevant}$ is not plausible, too. However, the probability that 3 process-relevant manifestations are identified for a sample size of 11, while the percentage of process-relevant manifestations is below or equal to 10%, is above 5% (p-value = 0.08956). Consequently, it is plausible that the percentage of process-relevant manifestations is below or equal to 10%.

Table 9.8.: Results of binomial test to check whether probabilities for process-neutral and process-relevant manifestations of MDE trait *number of phases* exceed 5% (or 10% respectively)

Manifestation	# occurrences	sample size	h_0	h_1	p-value	95% confidence interval	Result
process-neutral	4	11	$p \leq 5\%$	$p > 5\%$	0.001552	0.1350 - 1	$h_{0_neutral}$ is rejected
	4	11	$p \leq 10\%$	$p > 10\%$	0.01853	0.1350 - 1	h_0 is rejected
process-relevant	3	11	$p \leq 5\%$	$p > 5\%$	0.01524	0.0788 - 1	$h_{0_relevant}$ is rejected
	3	11	$p \leq 10\%$	$p > 10\%$	0.08956	0.0788 - 1	h_0 is plausible

Summary Subsuming, for both, the number of synchronization points and the number of phases, it can be shown that the MDE settings differ. Further, the investigation of phases reveals that both, MDE settings with a low number of phases (i.e. process-neutral manifestations) as well as MDE settings with a high number of phases (i.e. process-relevant manifestations of the MDE trait *number of phases*), are not seldom. With this, the analysis results support hypothesis H_{phases} .

9.3.2. Manual Information Propagation in Case Studies

For the analysis of manual information propagation, an automated search for matches of the Software Manufacture Model pattern *manual information propagation* on the prepared models was performed. Since it was ensured during the preparation described above in Section 9.1 that the modeling convention MC4 is fulfilled in the prepared models, matches to semi-automated activities could be simply ignored.

As summarized in Table 9.9 and Figure 9.8 the pattern was matched on 29 manual activities. 42 manually transformed artifact pairs could be identified. On average each of the MDE settings includes 3.82 manually transformed artifact pairs within 2.63 activities. In the case study VC and Cap2b no such pair was found, while in other case studies (e.g. BO, Cap1, or Carmeq) between 7 and 10 manually transformed artifact pairs could be identified.

The number on manually transformed artifact pairs differs from the number of activities for two reasons. On the one hand, some activities like, “model” in case study Carmeq, consume and create multiple artifacts, which combine to multiple pairs for the same activity. On the other hand, some activities affect the same pair of artifacts. For example, the activities “model” and “model change” in case study Carmeq are alternatives and affect mainly the same artifacts.

Excursus: coverage by quality assurance activities

Theoretically, each manually transformed artifact pair needs to be covered by some form of quality assurance. This is addressed differently. For example, for case study BO 7 manually transformed artifact pairs can be identified. A direct automated check is included in the MDE setting for one of the pairs (“BO Model” and “ESR BO Model”), only. Further, for two pairs (both include the artifact “Integration Scenario”) an indirect check is performed. The “Integration Scenario” is used to create

9. Study Results: Employing Modeling and Analysis

“Test Case” which are applied to the product that is derived on the basis of the other artifacts. Here, the creation of the “Test Case” is manual, too. Therefore, “Test Case” and “Integration Scenario” are a manually transformed artifact pair, too. This approach for testing (i.e. checking whether a derivation is correct by comparing it to another derivation) is a usual and widespread approach.

For the remaining three manually transformed artifact pairs an indirect test was captured. The artifact pairs are “Requirements” and “BO Specification”, “BO Specification” and “BO Model” as well as “BO Specification” and “BO Template”. All three pairs are not checked directly. Instead (considering the quality assurance activities that were captured in the records, only), faults that are introduced during these manual information propagations can only be identified, when the resulting product is tested against the requirements at the end of the development. Note that the records from the interviews cover no information about possible manual review activities here. Thus, such activities cannot be excluded for the example.

However, the example illustrates that the quality assurance that is used to address manually transformed artifact pairs can differ. On the one hand, quality assurance activities are directly embedded into the MDE setting (e.g. the comparison of “BO Model” and “ESR BO Model”). This leads to smaller correction cycles during the implementation, but also mixes development and quality assurance activities. On the other hand, quality assurance is done indirectly on the basis of the product. While this bypasses a blending of development activities and quality assurance activities, there are the risks that correction cycles include multiple activities and that the error source (i.e. the manual transformation, where the fault was introduced) cannot be identified.

Details on the identified pattern matches can be found in Appendix G.2.

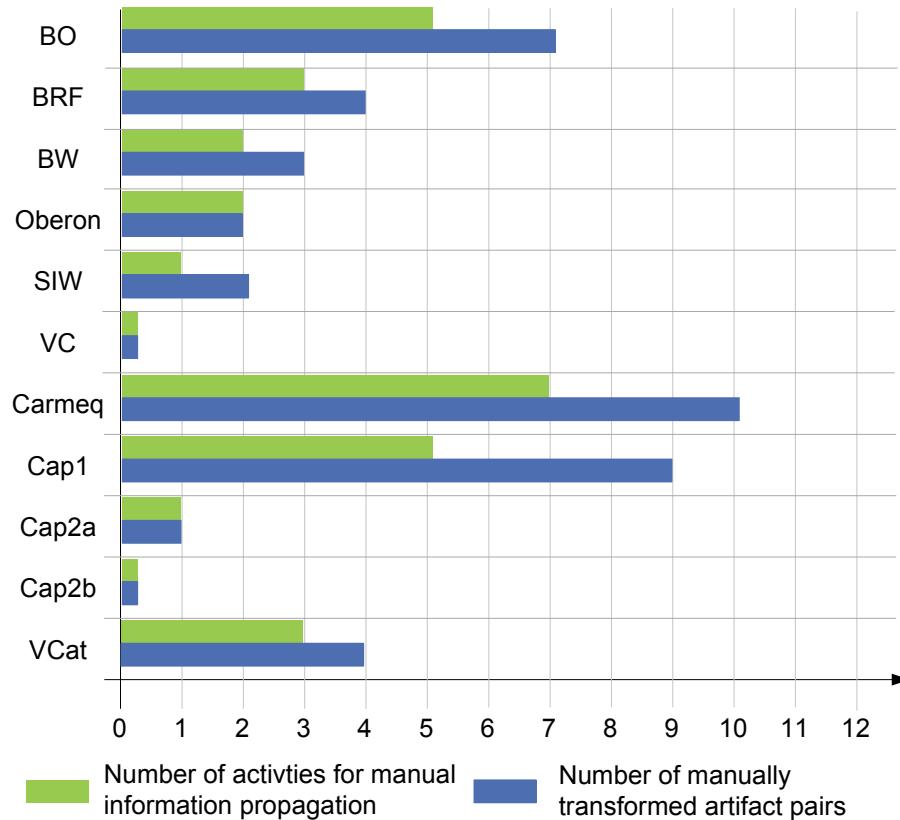


Figure 9.8.: Manual information propagation and manually transformed artifact pairs within the case studies.

Table 9.9.: Summary of identified matches for pattern *manual information propagation* in the case studies from the first and third study (see Chapter 4).

	# Activities that match pattern manual information propagation	# manually transformed artifact pairs
BO	5	7
BRF	3	4
BW	2	3
Oberon	2	2
SIW	1	2
VC	0	0
VCat	3	4
Carmeq	7	10
Cap1	5	9
Cap2a	1	1
Cap2b	0	0
All	29	42
Average	2.63	3.82

Evaluation of Hypothesis $H_{\text{manualInformationPropagation}}$

In this section, it is discussed whether this data supports hypothesis $H_{\text{manualInformationPropagation}}$ (“*In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait manual information propagation, in more than 5% of all MDE settings, each.*”).

Process-neutral and process-relevant manifestation For each of the eleven case studies between 0 and 10 manually transformed artifact pairs were identified. As discussed in Section 3.3.3 and on the example of BO above, manually transformed artifact pair are partly handled directly, which enforces developers that apply the manual information propagation to suspend their implementation work to perform the additional (potentially manual) quality assurance activities. 3 of the 11 case studies include between 7 and 10 manually transformed artifact pairs, which can be considered as process-relevant manifestation of the MDE trait *manual information propagation*.

Within the eleven case studies 2 include no manually transformed artifact pairs, and for a further one (Cap2a) the information propagation of the only manually transformed artifact pair happens in context of a quality assurance activity. Thus, at least 3 case studies include a process-neutral manifestation of the MDE trait *manual information propagation*.

Hypothesis testing In the following hypothesis $H_{\text{manualInformationPropagation}}$ is evaluated. As defined for hypothesis $H_{\text{manualInformationPropagation}}$ a phenomenon is considered to be not seldom, if it exists in more than 5% of all MDE settings. For the evaluation, two tests are made: First, it is tested whether the percentage of process-neutral manifestations of MDE trait *manual information propagation* exceeds 5%. The used null hypothesis $h_{0_neutral}$ is that the percentage of process-neutral manifestations is below or equal to 5%. Correspondingly, the alternative hypothesis $h_{1_neutral}$ is that the percentage of process-neutral manifestations is greater than 5%. Second, it is tested whether the percentage of process-relevant manifestations of MDE trait *manual information propagation* exceeds 5%. Here, the used null hypothesis $h_{0_relevant}$ is that the percentage of process-relevant manifestations is below or equal to 5%. Correspondingly, the alternative hypothesis $h_{1_relevant}$ is that the percentage of process-relevant manifestations is greater than 5%.

As mentioned above, the binomial test is used as a test that can be used to check for a small sample whether data points that conform to one of two categories occur to a certain percentage within the data set. For $h_{0_neutral}$ these categories are “*process-neutral*” and “*not process-neutral*”, while for $h_{0_relevant}$ the categories are “*process-relevant*” and “*not process-relevant*”. The applied significance level is 5%.

9. Study Results: Employing Modeling and Analysis

A summary of the results is shown in Table 9.10. For process-neutral manifestations the null hypothesis can be rejected. Thus, the probability that 3 process-neutral manifestations are identified for a sample size of 11, while the percentage of process-neutral manifestations is below or equal to 5%, is less than 5% (p-value = 0.01524). Thus, $h_{0_neutral}$ is not plausible. Also for process-relevant manifestations the null hypothesis can be rejected. The probability that 3 process-relevant manifestations are identified for a sample size of 11, while the percentage of process-relevant manifestations is below or equal to 5%, is less than 5% (p-value = 0.01524). Thus, $h_{0_relevant}$ is not plausible, too.

Again, not only a 5% percentage of process-neutral and process-relevant manifestations (which is sufficient for the check of the hypothesis $H_{manualInformationPropagation}$) are tested but also a 10% percentage. Here, the corresponding null hypothesis could not be rejected for a significance level of 5%. Thus, it is plausible that 3 process-neutral (or process-relevant) manifestations are identified for a sample size of 11, while the percentage of process-neutral (or process-relevant) manifestations is below or equal to 10%.

Table 9.10.: Results of binomial test to check whether probabilities for process-neutral and process-relevant manifestations of MDE trait *manual information propagation* exceed 5% (or 10% respectively)

Manifestation	# occurrences	sample size	h_0	h_1	p-value	95% confidence interval	Result
process-neutral	3	11	$p \leq 5\%$	$p > 5\%$	0.01524	0.0788 - 1	$h_{0_neutral}$ is rejected
	3	11	$p \leq 10\%$	$p > 10\%$	0.08956	0.0788 - 1	h_0 is plausible
process-relevant	3	11	$p \leq 5\%$	$p > 5\%$	0.01524	0.0788 - 1	$h_{0_relevant}$ is rejected
	3	11	$p \leq 10\%$	$p > 10\%$	0.08956	0.0788 - 1	h_0 is plausible

Summary To summarize the results, neither MDE settings with a process-neutral manifestation nor MDE settings with a process-relevant manifestation of the MDE trait *manual information propagation* are seldom. Therefore, the results of this analysis support hypothesis $H_{manualInformationPropagation}$.

9.3.3. Complexity of Activity Chains in Case Studies

The length of activity chains was measured for the prepared models following the method described in Section 7.2.3. Based on the interview records and models *change artifacts* and *target artifacts* were identified. Further, for the change artifacts it was marked which are used for the introduction of common changes (i.e. what changes are probable).

For example, *change artifacts* in BRF are the “BusinessRuleService”, the “SAPPProcess”, and the “Ruleset” (which contains “DecisionTables” and “Rules”). *Target artifacts* of this MDE setting are “DBEntry” and “RuleClass”, which are generated of the basis of the “Ruleset” and accessed during runtime by the “SAPPProcess” via the “BusinessRuleService”. The MDE setting of this case study is designed in a way that the “Rulesets” are tied initially to the “BusinessRuleServices” and the “SAPPProcess” and can be changed later on to customize the process to the business needs. Thus, only changes to “Rulesets”, “DecisionTables”, and “Rules” are probable. Now, 4 to 6 activities are required to propagate a change from the “SAPPProcess” to the “RuleClass”, while only 2 to 3 activities are required to propagate a change from the “Ruleset” to the “RuleClass”. Consequently, the maximum length of activity chains (6) differs in this setting from the range for the length of activity chains for the implementation and propagation of common changes (2 to 3 activities).

Figure 9.9 summarizes the maximum and common lengths of activity chains for the eleven case studies. Further, the ranges of manual activities that are necessary for a common change, as well as the maximum set of manual activities (i.e. the amount of manual activities for a change with the maximum number of activities) are provided.

While in some case studies the length of activity chains for a common change is relatively low (e.g. between 2 and 3 in BRF or 1 and 5 in SIW) it is higher in other case studies (e.g. between 8 and 16 in Carmeq). For the most case studies (10 out of 11) the minimum number of activities that can be used to implement a common change is between 1 and 3 (average 2.54 activities, mean 2 activities). In a single outlier (CarmeQ) this number is higher (8 activities).

The maximum length of activity chain varies between 4 and 16 (3 and 16 for common changes) and is on average 7.81 (6.81 for common changes). The mean is 8 for the maximum and 6 for the maximum length for common changes. The difference between minimum and maximum length of activity chain for common changes varies between 1 and 8 (average 4.27 activities, mean 4 activities).

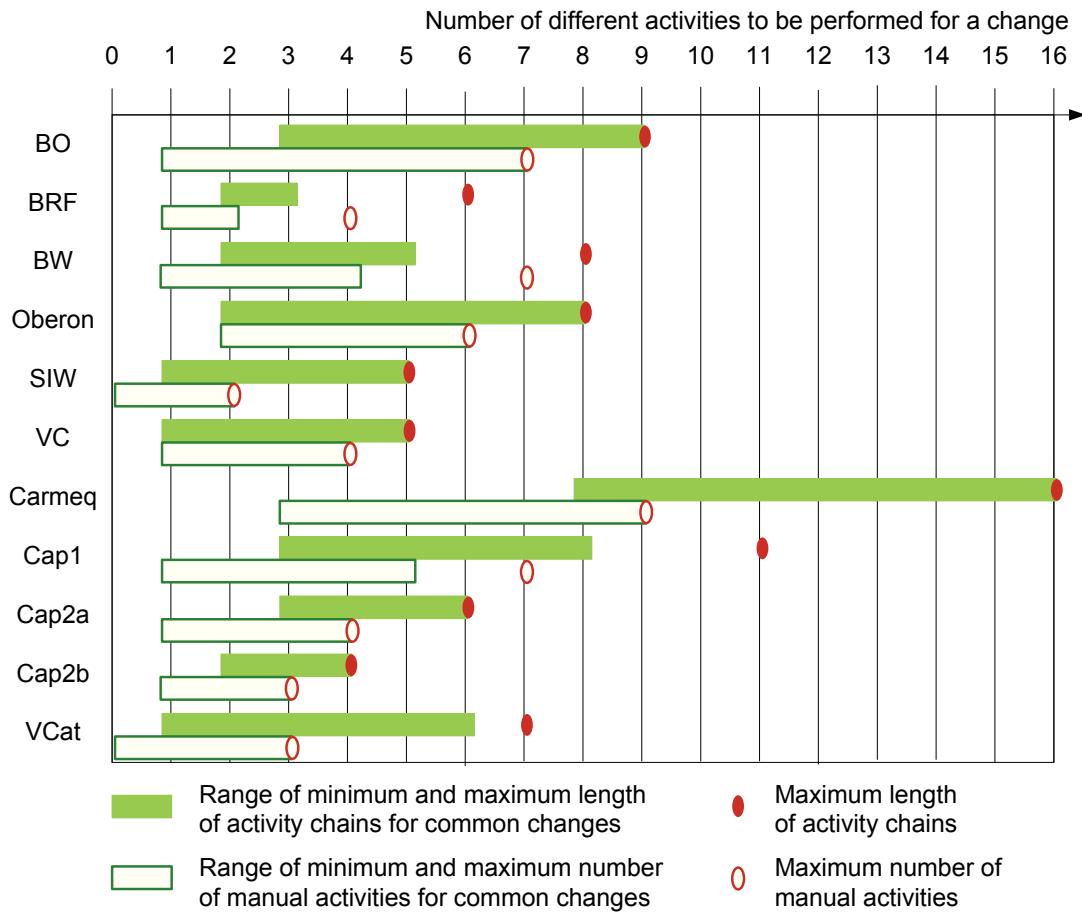


Figure 9.9.: Overview of the lengths on activity chains for the different case studies.

Excursus: improbable maximum

In only four of the 11 case studies (VCat, Cap1, BRF, BW) the maximum length of activity chains differs from the maximum length for common changes. This difference is in these four case studies between 1 and 3 activities (on average 2.5 activities, mean 3 activities). All of these four MDE settings reflect decisions to optimize changeability for certain changes at the expense of other changes (which are less probable or not required). For example, the MDE setting of case study BRF is optimized for changes in the “Ruleset” with the trade-off that changes in the “SAPPProcess” or the “BusinessRuleServices” are more expensive.

A comparison of the average maximum length of activity chains and maximum length of activity chains for common changes of these 4 case studies to the average maximum length of activity chain for

9. Study Results: Employing Modeling and Analysis

all eleven case studies provides an impression of the effect of this trade-off. While the maximum length of activity chains is with 8 activities slightly above the average (7.72 activities), the maximum length of activity chains for common changes (4 activities) is less than 60% of the average.

Considering only manual activities, the minimum number of activities that can be used to implement a common change is between 0 and 3 (average 1.09 activities, mean 1 activity). The maximum number of manual activities in an activity chain varies between 2 and 9 (2 and 9 for common changes) and is on average 5.09 activities (4.45 for common changes). The mean for the number of manual activities is 4 for the maximum and 4 for the maximum length for common changes. The difference between minimum and maximum number of manual activities within an activity chain for common changes varies between 1 and 6 (average 3.45 activities, mean 3 activities).

For two case studies the minimum number of manual activities for the propagation of a change is 0. In these cases the start artifact is an artifact that is manipulated outside the setting. For example, artifact “*local extension*” in VCat can be retrieved and installed automatically from a “*global extension*”. The propagation of changes at such an external “*global extension*” can be done fully automatically.

Table 9.11 summarizes the data on lengths of activity chains. Details on the considered artifact pairs can be found in Appendix G.3.

Table 9.11.: Summary of data on lengths of activity chains.

	minimum	maximum	average	mean
Minimum length of activity chain that can be used to implement (one of the) common changes	1	8	2.54	2
Minimum number of manual activities that can be used to implement (one of the) common changes	0	3	1.09	1
Maximum length of activity chain	4	16	7.81	8
Maximum number of manual activities in an activity chain	2	9	5.09	4
Maximum length of activity chain for common changes	3	16	6.81	6
Maximum number of manual activities in an activity chain for common changes	2	9	4.45	4
Difference between minimum and maximum length of activity chain for common changes	1	8	4.27	4
Difference between minimum and maximum number of manual activities within an activity chain for common changes	1	6	3.45	3

Evaluation of Hypothesis $H_{\text{complexActivityChains}}$

In the following it is discussed whether the presented data supports hypothesis $H_{\text{complexActivityChains}}$ (“*In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait complexity of activity chains, in more than 5% of all MDE settings, each.*”).

Process-neutral and process-relevant manifestations First of all, the difference between minimum and maximum length of an activity chain varies for the eleven case studies between 1 and 8 activities. A small difference supports developers in planning and predicting the effort that is necessary for the application of changes. In contrast a big difference makes a prediction more difficult. Three of the eleven case studies have a difference between 1 and 3 activities, only, while three other case studies have a difference between 6 and 8 activities.

The minimum length of activity chains to implement one of the common changes varies for the eleven case studies between 1 and 8. Similarly the maximum length of activity chains to implement one of the common changes varies for the eleven case studies between 4 and 16. As discussed in Section

3.3.3 a complex activity chain with many activities to be performed can prevent the application of agile processes. In contrast, a low number of activities supports changeability and with it can allow the application of agile processes. For example, the case studies BRF and VC (both with maximum 5 activities for common changes) aim at lightweight implementation and change. Thus, a maximum length of activity chains of 5 might still be considered as process-neutral manifestation of MDE trait *complexity of activity chains*.

In the example of case studies Carmeq, which requires between 8 and 16 activities for a change, our interviewee reported that an agile application of the setting was considered as not applicable. An anecdotal evidence that already a maximum length 6 activities might be experienced as complex by the developers can be found in case study Cap2a. There, the interviewees reported on a workaround that is used by some of the developers in order to gain results directly without adapting the model first. Consequently, a maximum length of activity chains of 6 or more can already be considered as process-relevant manifestation of MDE trait *complexity of activity chains*.

The eleven case studies include five MDE settings with a maximum number of 5 or less activities for the application of a common change. In contrast, for six of the MDE settings up to 6 activities and more might be required for the application of a change (for four of the MDE settings, even up to 8 and more activities might be required).

Hypothesis testing To test hypothesis $H_{complexActivityChains}$ it shall be tested whether both, the percentage of process-neutral and the percentage on process-relevant manifestations of MDE trait *complexity of activity chains*, exceed 5% of the MDE settings. As defined for hypothesis $H_{complexActivityChains}$ a phenomenon is considered to be not seldom, if it exists in more than 5% of all MDE settings.

Consequently two pairs of hypotheses are formulated. For process-neutral manifestations the null hypothesis $h_{0_neutral}$ is that the percentage of process-neutral manifestations is below or equal to 5%, while the alternative hypothesis $h_{1_neutral}$ is that the percentage of process-neutral manifestations is greater than 5%. Similarly, for process-relevant manifestations the null hypothesis $h_{0_relevant}$ is that the percentage of process-relevant manifestations is below or equal to 5%, while the alternative hypothesis $h_{1_relevant}$ is that the percentage of process-relevant manifestations is greater than 5%.

Each null hypothesis is tested with the binomial test, as mentioned above. This test can be used for small samples to check whether data points conform to one of two categories for a certain percentage of the data set. The two categories are “*process-neutral*” and “*not process-neutral*” for the test of $h_{0_neutral}$ and “*process-relevant*” and “*not process-relevant*” for the test of $h_{0_relevant}$, respectively. For the test a significance level of 5% is applied.

In Table 9.12 the results of the tests are summarized. In addition to the check for a 5% percentage of process-neutral and process-relevant manifestations, also a 10% percentage was tested. For both, process-neutral and process-relevant manifestations, the null hypotheses can be rejected. Thus, the probability that 5 process-neutral manifestations are identified for a sample size of 11, while the percentage of process-neutral manifestations is below or equal to 10%, is less than 5% (p-value = 0.002751). Consequently, $h_{0_neutral}$ is not plausible. Similarly, the probability that 6 process-relevant manifestations are identified for a sample size of 11, while the percentage of process-relevant manifestations is below or equal to 10%, is less than 5% (p-value = 0.0002957). Thus, $h_{0_relevant}$ is not plausible, too.

Summary To summarize, the results of this analysis reveal that both, MDE settings with a process-neutral manifestation of MDE trait *complexity of activity chains* and MDE settings with a process-relevant manifestation of MDE trait *complexity of activity chains* exists and are not seldom. This data supports the hypothesis $H_{complexActivityChains}$.

9.3.4. Evaluation of Hypothesis H_{traits}

Above the analysis methods from Section 7.2 were applied to the models of the eleven case studies from the first and third study (see Chapter 4) to evaluate the hypotheses H_{phases} , $H_{manualInformationPropagation}$, and $H_{complexActivityChains}$ from Section 3.5.2.

All three hypotheses are part of the comprehensive hypothesis H_{traits} (*In practice the identified MDE traits occur within a spectrum that includes process-relevant manifestations as well as process-neutral manifestations. Thus, there are MDE settings with process-relevant manifestations for some or all of the MDE traits, but there are also MDE settings with process-neutral manifestations for some or all MDE*

9. Study Results: Employing Modeling and Analysis

Table 9.12.: Results of binomial test to check whether probabilities for process-neutral and process-relevant manifestations of MDE trait *complexity of activity chains* exceed 5% (or 10% respectively)

Manifestation	# occurrences	sample size	h_0	h_1	p-value	95% confidence interval	Result
process-neutral	5	11	$p \leq 5\%$	$p > 5\%$	0.0001119	0.1996 - 1	$h_{0_neutral}$ is rejected
	5	11	$p \leq 10\%$	$p > 10\%$	0.002751	0.1996 - 1	h_0 is rejected
process-relevant	6	11	$p \leq 5\%$	$p > 5\%$	5.801e-06	0.2712 - 1	$h_{0_relevant}$ is rejected
	6	11	$p \leq 10\%$	$p > 10\%$	0.0002957	0.2712 - 1	h_0 is rejected

traits). Since all three hypotheses (H_{phases} , $H_{manualInformationPropagation}$, and $H_{complexActivityChains}$) could be supported by the data, H_{traits} is supported, too. Thus, the evaluation of these three hypotheses confirms that, there are MDE settings with process-relevant manifestations for some of the MDE traits, but there are also MDE settings with process-neutral manifestations for some MDE traits.

This evaluation is further strengthened by the fact that there are also MDE setting with process-relevant manifestations of all three MDE traits (e.g. BO, Carmeq, and Cap1). Further, one MDE setting with process-neutral manifestations of all three MDE traits was identified (VC).

Table 9.13 summarizes these results.

Table 9.13.: Summary of hypotheses on process interrelation

Hypotheses	Percentage of process-neutral manifestations	Percentage of process-relevant manifestations	
H_{phases} In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait phases, in more than 5% of all MDE settings, each.	> 10%	> 5%	✓
$H_{manualInformationPropagation}$ In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait manual information propagation, in more than 5% of all MDE settings, each.	> 5%	> 5%	✓
$H_{complexActivityChains}$ In practice there are MDE settings with a process-neutral manifestation and MDE settings with a process-relevant manifestation of MDE trait complexity of activity chains, in more than 5% of all MDE settings, each.	> 10%	> 10%	✓
H_{traits} In practice the identified MDE traits occur within a spectrum that includes process-relevant manifestations as well as process-neutral manifestations.			✓

9.3.5. Threats to Validity

In the following, threats to conclusion validity are discussed for the presented evaluation of hypotheses H_{phases} , $H_{manualInformationPropagation}$, $H_{complexActivityChains}$, and H_{traits} (for the discussion of other threats to validity see Section 4.4). The classification of manifestations into process-relevant and process-neutral manifestations was carefully done on the basis of anecdotal evidences. Consequently, often not the whole spectrum of an MDE trait is categorized. For example, when 2 - 6 manually transformed artifact pairs are identified the manifestation of *manual information propagation* is neither classified as process-neutral nor process-relevant. Here more data would be preferable to ensure that all process-relevant manifestations are categorized as such.

In general, a bigger size of the data set would be preferable. However, the size of 11 MDE settings allows at least some conclusions on the frequency of process-relevant manifestations and process-neutral manifestations of the different MDE traits. Fortunately, for each MDE trait process-relevant manifestations were assessed in MDE settings from multiple companies. Similarly, the data is sufficient to show that the existence of process-neutral manifestations of the MDE traits is not company specific, too. Therefore, the data is sufficient for the evaluation of the hypotheses H_{phases} , $H_{manualInformationPropagation}$, $H_{complexActivityChains}$, and with this H_{traits} .

9.4. Findings on the Influence of Evolution on Changeability and Process Interrelation

In this thesis the nature of MDE is investigated from different perspectives. On the one hand, the influence of MDE settings of changeability and the interrelation to software processes is discussed. It was shown that MDE settings from practice differ (see Sections 9.3 and 9.2). This includes the observation that there are MDE settings that have negative influences on changeability or lead to constraints on software development processes.

On the other hand, it was theoretically discussed that structural and substantial structural evolution can change these characteristics of an MDE setting (see Section 3.4.2). Further, it was shown that structural and substantial structural evolution actually occurs in practice.

What are still missing are actual practical examples on evolution steps that caused the introduction of negative influences on changeability or constraints on software development processes into an MDE setting. The final examination in this thesis is applied to the question whether the collected data already includes hints on or actual evidences for such evolution steps. Therefore, first the method of this investigation of the data is presented. Then it is discussed to what extent the introduction of pattern occurrence can be ascribed to specific evolution steps. Afterwards, it is discussed how captured evolution steps might have changed manifestations of the different MDE traits. Finally, the results of this final examination are summarized and the validity of the conclusions is discussed.

9.4.1. Method

For this investigation the data from the third study is used. This includes the six case studies where a Software Manufacture Model of one version of the MDE setting is available: BO, VCat, Carmeq, Cap1, Cap2a, and Cap2b.

Although data on the evolution steps for these MDE settings is available, only for respectively one (historical) version of each MDE setting a complete Software Manufacture Model is available. This is in the most cases the currently used version of the MDE setting. An exception is cases study BO where an older version of the MDE setting was captured in detail. This version resulted from the fourth evolution step (S4) and was replaced with the fifth evolution step (S5). The Software Manufacture Model captured for VCat represents the current version. However, this version is result of the first evolution step (S1), while the second captured evolution will be applied in future to replace this current setting.

The evolution steps are captured in form of change descriptions, and allow no detailed reconstruction of the actual activity details. Therefore, the approach to completely reconstruct the Software Manufacture Models for all historical versions is not applicable.

However, it is possible to stepwise consider the evolution steps (starting at the MDE setting that was captured in detail) and rate whether these evolution steps had the potential to affect the manifestations of MDE traits and pattern occurrences that were analyzed in Section 9.3. The results can only be a

rating of tendencies and potentials for *synchronization points*, *phases*, and the *complexity of activity chains*, which are traits that base on the rather global information of predecessor sets. In contrast, it is easier to identify evolution steps that lead to the introduction of the identified occurrences of proto-anti-patterns, since patterns are identified based on local information, only. Nonetheless, this approach cannot be used for the identification of proto-anti-pattern occurrences in historical versions of the MDE setting that were later on removed.

As a result, the described analysis approach provides data that allow estimating to what extent occurrences of proto-anti-patterns resulted from evolution steps and to what extent evolution steps affected manifestation of the different MDE traits.

9.4.2. Effects of Evolution on Changeability

In the following it is examined whether a direct influence of evolution on occurrences of the proto-anti-pattern *subsequent adjustment* and *creation dependence* can be observed within the six available case studies from the third study.

Within the Software Manufacture Models that are captured for the six case studies, 6 occurrences of proto-anti-pattern *creation dependence* could be identified in Section 9.2. As summarized in Table 9.14 and Figure 9.10, for 5 of these 6 occurrences, it is possible to identify evolution steps that introduced the matched activities into the MDE setting. For example, the activity “*ApplyChecks*” in BO that is matched to *creation dependence* was most probably introduced in the third evolution step (S3), where a couple of validation activities were added in context of a technological change. In Cap2b, the matched activity “*Generate Result8*” was introduced in the fourth evolution step (S4), together with the introduction of the generation of “*Result8*”. Only one of the occurrences of *creation dependence* (in case study Cap1) was most probably part of the corresponding MDE setting from the start.

For proto-anti-pattern *subsequent adjustment* 8 occurrences within the Software Manufacture Models captured for the six case studies could be identified in Section 9.2. For 5 of these 8 occurrences introducing evolution steps can be identified. For example, the automated activity “*Generate Code & Tables*” was already introduced to BO in the first evolution step (S1). Later the activity “*ModelS&AM*” was introduced in BO during the second evolution step (S2) in context of the introduction of the concept of Status and Action Management (SAM). Together “*Generate Code & Tables*” and “*ModelS&AM*” match to *subsequent adjustment*.

Three of the occurrences of *subsequent adjustment* were most probably part of the corresponding MDE setting from the start. For example, following our records, the use of the “*Construction Model*” in Cap1 and with it the match of *subsequent adjustment* on the activities “*Create Constructionmodel*” and “*Manipulate Constructionmodel*” was already part of the initial MDE setting.

Excursus: evolution can also remove negative effects

The fact that the detailed version of the MDE setting BO is not the currently used version, allows the observation that occurrence of proto-anti-patterns can also be removed by evolution steps. The match of “*Generate Code & Tables*” and “*ModelS&AM*” to *subsequent adjustment*, which was introduced with evolution step S2, was removed later on in the clean-up evolution step S5. Functionality of multiple tools was integrated and the S&AM models were moved to an earlier position in the activity chain.

Theoretically, the introduction of *subsequent adjustment* can be caused by different changes. In all five examples where an introducing evolution step could be identified, this evolution step is a *substantial structural evolution step*. The match of *subsequent adjustment* was two times introduced by the addition of an automated activity that matched to *initial creation*. Once an automated activity was exchanged by another automated activity and a following manual activity (in order to produce the result in another format). In a fourth case both matched activities were introduced together. Finally, the match of “*Generate Code & Tables*” and “*ModelS&AM*” in BO was introduced by the introduction of the manual activity, only. 3 of the 5 occurrences of *subsequent adjustment* that were introduced by evolution steps have no mitigating factors (i.e. their negative effects manifest in practice).

Excursus: C6 and C8 are truly substantial even when they do not cause C9

This example, how the match of “*Generate Code & Tables*” and “*ModelS&AM*” to *subsequent adjust-*

Table 9.14.: Summary of identified evolution steps that introduced occurrences of proto-anti-patterns within MDE settings of the third study (see Chapter 4).

Case Study	Pattern Occurrence	Introducing Evolution Step
<i>creation dependence</i>		
BO	“ApplyChecks”	S3
Cap1	“Create Constructionmodel”	—
Cap2b	“Generate Result7” “Generate Result8” “Word Import Result2” “Word Import Result3”	S4 S4 S1 S1
<i>subsequent adjustment</i>		
BO	“Generate Code & Tables” and “Write Business Logic” “Generate Code & Tables” and “ModelS&AM” “Project to BO Model” and “Adapt BusinessObject Model”	S1 S2 S3
VCat	“acquire zip-file” and “update extension” “acquire zip-file” and “customize extension” “run installation script” and “customize extension”	— — S1
Cap1	“Create Constructionmodel” and “Manipulate Constructionmodel”	—
Cap2a	“Merge Parts of Model” and “Manipulate Result2”	S1

ment was introduced to the MDE setting, illustrates that an evolution step that does not include change type C9, but only C6 or C8, can cause effects on how hard changeability concerns are affected, too. In the given example, the order of manual and automated activities did not change, because “Generate Code & Tables” was already predecessor of another manual activity (“Write Business Logic”, which matches to subsequent adjustment in combination with “Generate Code & Tables”, too).

This supports the classification of C6 and C8 as substantial structural evolution steps, even in cases where C9 is not caused as a side effect.

Considering all 26 captured substantial structural evolution steps in these 6 case studies, 7 (more than one quarter) could be ascribed with the actual introduction of proto-anti-patterns.

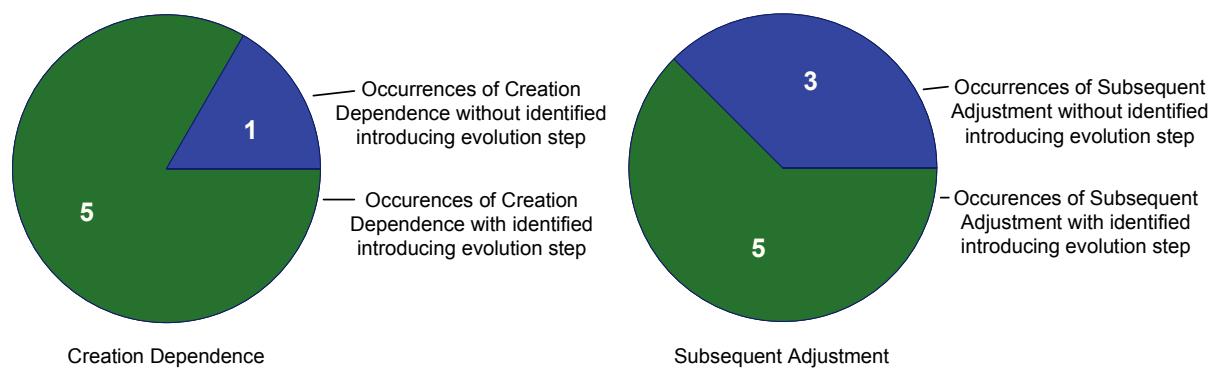


Figure 9.10.: Proportion of pattern occurrences that can be associated with an introducing evolution step: for 5 out of 6 occurrences of proto-anti-pattern creation dependence an introducing evolution step was identified, and for 5 out of 8 occurrences of proto-anti-pattern subsequent adjustment an introducing evolution step was identified.

9.4.3. Effects of Evolution on Process Interrelation

In the following, the impact of the evolution on the manifestations of MDE traits that were investigated in Section 9.3 is discussed.

Phases and Synchronization Points

As described above changes in the number of synchronization points and phases are more difficult to reconstruct than changes in pattern occurrences. However, for two of the case studies it is possible to associate evolution steps with probable changes to the number of phases. In the Carmeq the second phase (which includes the modeling activities) was added in the first evolution step (S1). Further, in Cap1 the evolution steps S5 and S7 led to the introduction of the first and second local focus, which both include an additional phase, respectively.

These two examples show that evolution affects the number of phases in practice, too, even if this effect seems to be rather seldom.

Complexity of Activity Chain

Similarly, it is difficult to identify changes in the complexity of activity chains. However, some evolution steps can be associated with tendencies. For example, while the first four evolution steps in BO (S1 - S4) rather increased the complexity, the later evolution steps (S5 - S7) reduced the complexity. Similarly, the evolution steps S1 and S2 introduced a supporting activity and checks into the MDE setting of case study Cap1 – the complexity of activity chain increased. The technological change in evolution step S3 of Cap1 made the supporting activity obsolete and thus decreased the complexity of activity chain. Later further evolution steps (S4 and S6) again increased the complexity of activity chain, by introducing other supporting activities and further checks.

Also in the case studies Carmeq and Cap2a the addition of checks and automated or semi-automated activities for the support of manual activities increased the complexity (Carmeque S2, S3 and Cap2a S4, S5). Finally, the introduction of the “*generation of figures and tables*” in S1 in Carmeq can be associated with an increasing complexity of activity chain.

Subsuming, evolution steps sometimes change the complexity of activity chains in practice (summarized in Figure 9.11). An increase of the complexity seems to occur more often than decrease of the complexity.

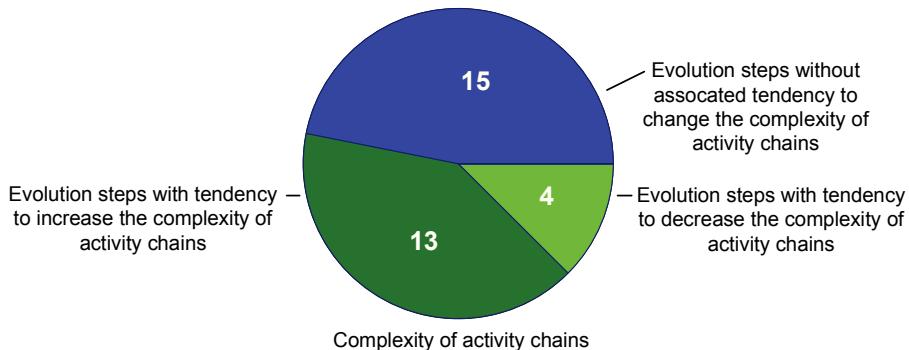


Figure 9.11.: Amount of evolution steps that can be associated with changes in the complexity of activity chains: 13 out of 32 evolution steps can be associated to increasing complexity of activity chains, while 4 out of 32 evolution steps can be associated to decreasing complexity of activity chains

Manual Information Propagation

Similar to the discussion of the proto-anti-patterns above, already identified manually transformed artifact pairs can be investigated in order to identify evolution steps that introduced these manual information propagations. As summarized in Table 9.15 and Figure 9.12, for 13 out of 31 manually

transformed artifact pairs the evolution step that introduced this manual information propagation into the corresponding MDE setting could be identified.

For example, in Cap1 the introduction of generation of the prototype in evolution step S5 added the manual information propagation between the artifacts “*UI-Specification*” and “*Code*”. Later, the situation that a second team started to manipulate copies of the “*Specification model*” added a task for merging different versions of the “*Specification model*”, which led to the introduction of the manually transformed artifact pair “*Specification model*” and “*Specification model to be merged*” in evolution step S6. Finally, the introduction of the task to create “*Specifications*” led to the introduction of three further manually transformed artifact pairs in evolution step S7.

The remaining 18 manually transformed artifact pairs were most probably already part of the initial MDE settings.

In addition, two evolution steps could be identified that led to a reduction of the number of manually transformed artifact pairs. In BO the fifth evolution step (S5) removed the manually transformed artifact pair “*ESR BO Model*” and “*BO Model*”, which was introduced in S4. Similarly, in VCat the first evolution step (S1) removed the manual task of copying configurations from an old framework. Although not identified here, there might be much more manually transformed artifact pairs that were removed during the evolution of the six case studies.

Table 9.15.: Summary of identified evolution steps that introduced manual information propagation into MDE settings from the third study (see Chapter 4).

Case Study	Manually Transformed Artifact Pair	Introducing Evolution Step
BO	“ <i>ESR BO Model</i> ” - “ <i>BO Model</i> ”	S4
Carmeq	“ <i>Basic SW Model</i> ” - “ <i>Graphics</i> ” “ <i>Basic SW Model</i> ” - “ <i>Concept Document</i> ” “ <i>Basic SW Model</i> ” - “ <i>Table</i> ” “ <i>Basic SW Model</i> ” - “ <i>Workingcopy of BSW Model</i> ” “ <i>Released Table HTML</i> ” - “ <i>Table HTML</i> ” “ <i>Figure PNG</i> ” - “ <i>Released Figure PNG</i> ”	S1 S1 S1 S1 S1 S1
Cap1	“ <i>Code</i> ” - “ <i>UI-Specification</i> ” “ <i>Specification model</i> ” - “ <i>Specification model to be merged</i> ” “ <i>Specification</i> ” - “ <i>Feature Lists</i> ” “ <i>Specification</i> ” - “ <i>Specification model</i> ” “ <i>Specification</i> ” - “ <i>GUI Widget Lists</i> ”	S5 S6 S7 S7 S7
Cap2a	“ <i>Partial Model</i> ” - “ <i>Mail</i> ”	S4

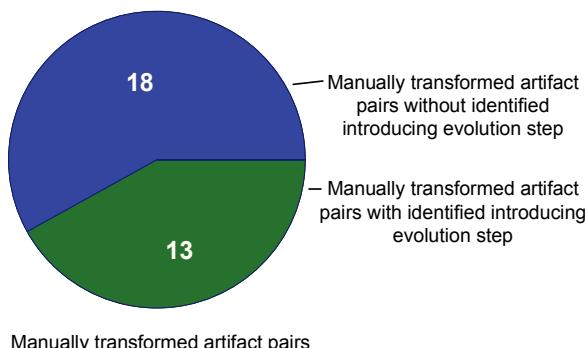


Figure 9.12.: 13 out of 31 manually transformed artifact pairs can be associated with an introducing evolution step.

9.4.4. Discussion of Evolution Effects

Following the results of this investigation are discussed. Based on the data from Section 9.4.2, it is not possible to identify a correlation between the actual number of structural or substantial structural evolution steps that occurred to an MDE setting and the number of matches of the proto-anti-patterns. However, it was possible to identify for 10 out of 14 occurrences of proto-anti-patterns introducing evolution steps. These results from practice support the theoretical consideration that substantial structural evolution steps can affect the changeability (see Section 3.4.2).

For changes in the manifestations of MDE traits the impact of evolution seems to be smaller. However, the investigation revealed examples that evolution actually led to changes in the manifestations of MDE traits in practice.

Subsuming, this final investigation revealed that characteristics of an MDE setting that concern the support of changeability and the interrelation to software development processes are actually affected by evolution in practice.

9.4.5. Threats to Validity

In the following, threats to conclusion validity are discussed for the presented results on evolution effects (for the discussion of other threats to validity see Section 4.4). The data set for this last examination consists of only 6 MDE settings. The number of pattern matches under study is with 6 and 8 relatively low. Consequently, it is not possible to use this data to draw conclusions on the general frequencies of proto-anti-pattern occurrences that were introduced by structural evolution steps, compared to proto-anti-pattern occurrences that were already part of the initial MDE setting. Similarly, data on more MDE settings and evolution histories would be necessary to make statements on the frequency of structural evolution steps that affect the manifestations of the different MDE traits.

However, evolution steps with effects on proto-anti-pattern occurrences or manifestations of the MDE traits were identified for multiple companies, respectively. This makes the data sufficient to conclude that changes in these characteristics of MDE settings are caused by evolution steps in practice.

10. Evaluation of Modeling and Analysis

So far the Software Manufacture Model language and analysis techniques have been introduced (in Chapters 6 and 7) and applied to MDE settings from practice (see Chapter 9). In this Chapter the Software Manufacture Model language and analysis techniques are evaluated.

10.1. Evaluation of Software Manufacture Model Language

The Software Manufacture Model language was designed to allow modeling of MDE settings from practice. One goal was to allow a comparison of the effects of activities. Further, the language aims at supporting analysis of an MDE setting's effects on changeability and the process interrelation. In this Section it is evaluated whether Software Manufacture Models are applicable in practice and allow a comparison of effects of different activities. Further, it is shortly discussed what concepts of the language design enabled the development of the analysis techniques that have been presented in Chapter 7.

10.1.1. Applicability

The Software Manufacture Model language was applied to model 11 case studies of MDE settings from practice. Together, 228 activities were modeled. Only, nine of these activities include more than 5 links. All activities could be expressed and visualized. Subsuming the language turns out to be applicable.

Although Software Manufacture Models are more complex than other process modeling languages, an experience when capturing the case studies was that the application of Software Manufacture Models is not more complex. In contrast, the application of the language supported the interviews and helped to ask questions in a more focused way. Actually, the effort that was put into the elicitation of an MDE setting decreased with the third study, where Software Manufacture Models were directly used during elicitation of the MDE settings. A main explanation for this improvement is the fact that the elicitation method changed. However, if Software Manufacture Models increase the complexity of eliciting an MDE setting, the effect is excelled by other influences on the elicitation effort (e.g. as the elicitation method).

10.1.2. Comparability

In the following it is examined whether the Software Manufacture Model language helps to better compare activities in MDE. Therefore, it is considered how classical model operations look like in Software Manufacture Model notation. In the area of software development with MDE several standard model operations are applied, such as model merge, model comparison, transformation, synchronization, or model check [138, 48], but also model creation or deletion. Taking a closer look at these model operations reveals that they are very variable in most parts of the activity, due to a high variability in the used definitions and understandings:

Example 16

For example, when trying to capture a model merge as a Software Manufacture Model activity, two characteristic aspects can be identified: a) input artifacts are not in a containment relation to each other (i.e. the only allowed artifact relations are references and referenced by), and b) the content of an input artifact overlaps or is equal to the content of the output artifact. Figure 10.1 shows an example of a merge activity that fulfills this characteristic. However, it is possible that the output artifact of a merge is identical to one of the input artifacts, while other merge implementations always produce a new artifact. This can make a relevant difference for the MDE setting, since the affected input artifact is available for further treatment in the later case, while it is not in the first case. Further, not each merge technology is able to implement both variants. Subsuming, each merge fulfills at least the listed properties, but can be refined further.

This uncertainty about what a merge actually means is even more drastic when transformations are considered. The term transformation is used in quite different ways. The only thing that is not variable

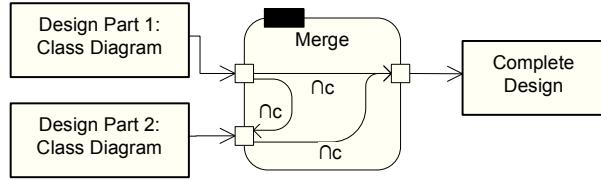


Figure 10.1.: Example for a model merge with two inputs

in the characteristic of a transformation activity is the fact that input and output artifacts have at least a content overlap (i.e. some information of the input is taken and moved to the output, where information might be added by the transformation) or are even equal in content. Figure 10.2 shows an example transformation activity, which translates one artifact to an artifact of another language automatically.

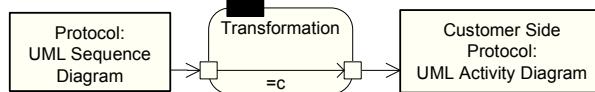


Figure 10.2.: Example for a transformation with input and output of different languages

Other characteristics differ from transformation to transformation: In an in-place transformation the output artifact is one of the input artifacts. The number of input and output artifacts can differ (and is for some transformations allowed to be variable). A transformation can be performed manually or automated. There are transformations that require a transformation definition (e.g. TGG rules) while other transformations are hard-coded. These examples show that the term transformation, too, is not sufficient to make a clear statement about the effects and role of an activity within an MDE setting.

Besides, the aspect that instances of standard model operations might have very different characteristics it can be observed that a differentiation of activities on the basis of these terms can be misleading. On the one hand, it is not always clear what the difference between two types of model operations is. For example, a merge is sometimes seen as an own form of model operation and sometimes understood as a special case of transformation.

On the other hand, one of our case studies revealed that it seems to be difficult for developers or designers of an MDE setting to transfer and apply knowledge about one type of model operation to other types of model operations.

Excursus: terminology can prevent the application of solutions

For example, the case study Cap1 includes an automated “model transformation” (activity “Create Specification Model”) as well as an automated “code generation” (activity “Generate Code”). In both cases the results need to be further manipulated and enriched. This is implemented for the model transformation by directly manipulated the resulting model – including the risk that a new execution of the model transformation will lead to loss of this manually added content. However, the adaption of the results of the “code generation” is done, by applying the generation gap pattern [78] – which aims at preventing exactly the same risk that can be associated with the “model transformation”. Obviously, the same problem was not identified or addressed for a transformation activity between two models. When calling attention to this situation, the interviewees admitted that the loss of manually created content exists and that developers use workarounds like “copy and paste”. The question for the reason why they choose to apply a technique for prevention of this effect to the code generation, but not to the model transformation, resulted in a discussion about the terminology. The application of the generation gap pattern happened on an activity which includes the words “Generate Code” in its name. Here the naming helped the developers to recognize the possible application point for solutions to prevent unexpected loss of code. In contrast to the “code generate”, our interviewees referred to the “model transformation” as “automated support” and hence just did not attach so much importance to this activity. This example provides an idea how differentiations in terminology can mislead developers when creating an MDE setting.

However, both activities turned out to be comparable when the Software Manufacture Model of the MDE setting was created during the interviews. Figure 10.3 shows both activities. As marked red, both activities include content flow from one of the input artifacts to one of the resulting artifacts. Both activities do not consume this resulting artifact as input. Thus content of this artifact is not preserved when the activity is executed a second time (which is the characteristic that leads to the potential loss of content). Both activities are also different. For example, how the transformation is executed is influenced by two scripts. During the transformation, the reference to the “UI-Specification” is copied from the “Specification Model” to the “Structural Models”. Finally, for tracing reasons a second model is created, which stores references to both “Specification Model” and “Structural Models”.

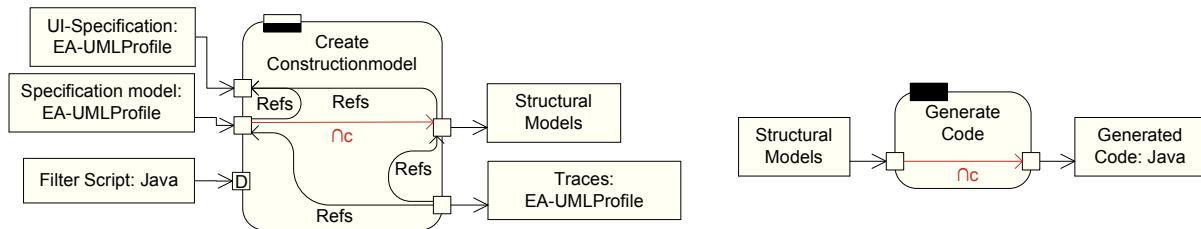


Figure 10.3.: Transformation (left) and generation (right) from case study Cap1.

Subsuming, this examination of standard model operation terminology reveals that the Software Manufacture Model language in contrast to normally used model operation terminology

- a) helps to identify relevant differences between activities of the same type of model operation and
- b) can help to identify potentials for improvements and support decisions on solutions independent of the actually used model operation terminology.

10.1.3. Analysis Support

One main aim of the Software Manufacture Model language is to support analysis of an MDE settings’ influence on hard changeability concerns and assessment of the manifestation of MDE traits. In the following, it is shortly evaluated whether these goals are reached.

First, two of the main concepts in Software Manufacture Models are the differentiation between manual and automated activities and the expression of changes in artifact relations in terms of pre- and post conditions of activities. These two modeling concepts enable the straight forward identification of manual information propagation using a simple Software Manufacture Model pattern. Further, the analysis of an MDE setting’s effect on hard changeability concerns is enabled, since changes in artifact relations (concerning references to other artifacts and content overlaps) are expressed in Software Manufacture Models. Due to the additional ability to express artifact hierarchies (i.e. to model partial artifacts and their changing relations) it is possible to express that different parts of an artifacts are affected differently. This enabled the formulation of proto-patterns on changeability concerns.

Finally, the expression of object flow and activity pre and post conditions, is used as a basis to automatically calculate the set of technically possible predecessor sets (and with it predecessors and successors) of activities within a Software Manufacture Model. This information about the order of activities enables the identification of lengths of activity chains, phases, and synchronization points.

Targeted improvements to the languages’ expressiveness (e.g. concerning the differentiation of activities for creation, recreation, and manipulation) can be applied in future to decrease the number of necessary modeling conventions. Nonetheless, the reached degree of automation of the analysis techniques is already very high. The main manual effort is caused by the need to create Software Manufacture Models that conform to the modeling conventions. For the analysis only some additional decisions are required. To sum up, the Software Manufacture Model language enabled the development of intuitive analysis techniques.

10.1.4. Discussion

Subsuming, it was shown that the Software Manufacture Model language is applicable to capture MDE settings from practice. Further, the examination of comparability revealed anecdotal evidences that Software Manufacture Models can help to get another focus when reasoning about MDE activities, compared to usually used model operation terminology. The final discussion showed that the different concepts in the Software Manufacture Model language indeed supported the design of the analysis techniques for changeability and process interrelation.

Apart from the evaluation of applicability, comparability and analysis support, the application of Software Manufacture Models to the case studies revealed that additional hidden aspects of an MDE setting might be identified:

Example 17

For example, in the interviews, an implicit dependency between automation steps in the case studies Cap2a and Cap2b could be identified on the basis of the Software Manufacture Models. One of the results from Cap2b includes hyper links to corresponding parts of a generation product of Cap2a. This implicit relationship might one day become relevant when the link addresses that are created in Cap2a are changed and the results created with the MDE setting in Cap2b become invalid. Interestingly, a comparable hidden dependency could also be identified between two automated activities in case study Carmeq.

Thus, modeling how artifact relations change within activities, helped to reveal situations, where the creation of required artifact relations is done implicit (i.e. by cloning the code for the creation of links or storage locations in multiple automated activities).

10.2. Evaluation of Analysis Techniques

In this section it is evaluated whether the analysis techniques presented in Chapter 7 are applicable to MDE settings from practice and whether there is an actual need for these analysis techniques. Further the overall analysis approach from Section 7.3 is discussed.

10.2.1. Applicability of Analysis

All analysis techniques that have been presented in Chapter 7 have been applied to 11 MDE settings from practice in Chapter 9. This shows that the developed analysis techniques are actually applicable.

Special cases are the proto-patterns. First, the application of the analysis shows that the pattern matching is applicable. However, there is also the question whether the proto-patterns that were introduced in Section 7.1 have the potential to be actual patterns.

A proto-pattern has the potential to become an actual pattern, when more than three occurrences in different contexts (or companies) in practice are documented. For the two proto-anti-patterns this rule-of-thumb can be evaluated as fulfilled when taking the eleven case studies and the open source emf case study together. Together, 20 occurrences of proto-anti-patterns *subsequent adjustment* are identified within 8 MDE settings in 3 companies as well as in the open sources case study. Similarly, 8 occurrences of *creation dependence* are identified within 5 MDE settings in two companies as well as in the open sources case study.

Only some matches for the proto-patterns with positive effects on changeability could be identified, while the case studies revealed the application of further structures for the support of changeability. The first conclusion is that the data that was so far collected is not sufficient to proof that *anchor* is more than a proto-pattern. A second conclusion of this observation is, that the two proto-patterns (although technically valid) are not sufficient to capture all changeability improving strategies that are relevant and applied in practice. In future work, the collection of additional data on MDE settings might lead to the formulation of additional Software Manufacture Model patterns to capture these solutions.

Finally, for *split manufacture* 5 occurrences are identified within 3 MDE settings in two companies as well as in the open sources case study. Thus, the proto-pattern has not only been matched in the

MDE setting, where it was initially identified, but also in other MDE settings. Consequently, for this proto-pattern the rule-of-thumb can just be evaluated as fulfilled.

Subsuming, due to the occurrences in multiple MDE settings from different companies, three of the four presented proto-(anti-)patterns seem to actually capture reoccurring situations that are relevant for MDE settings in practice.

10.2.2. Need of Analysis Techniques

In Chapter 3 it was examined how MDE settings theoretically can affect changeability and that the manifestation of the three presented MDE traits might be process-relevant. However, in the beginning it was not clear whether there are MDE settings in practice that include process-relevant manifestations of the three MDE traits or affect hard changeability concerns. Consequently, it was not clear whether analysis techniques are actually needed.

Thus, in order to exclude or confirm the need for the analysis techniques, it was first necessary to analyze the MDE settings from practice. Therefore, first the required analysis techniques have been introduced in Chapter 7 and applied to the case studies from practice in Chapter 9. As a result, it is now possible to evaluate the need for the analysis techniques:

For the MDE traits the analysis of the MDE settings from the first and third study in Chapter 9 led to a confirmation of the hypothesis H_{traits} , including the hypotheses H_{phases} , $H_{manualInformationPropagation}$, and $H_{complexActivityChains}$ (see Section 9.3). This means that for each of the three MDE traits there are in practice MDE settings, where the MDE trait is manifested process-relevant, and MDE settings, where the MDE trait is manifested process-neutral. The process relevance of certain manifestations of the MDE traits was theoretically discussed and some anecdotal evidence is provided in Chapter 9. Consequently, the difference in practice confirms the need for analysis techniques that allow assessing the manifestation of these MDE traits.

For the hard changeability concern *unexpected loss of content*, the analysis of the MDE settings from the first and third study in Chapter 9 led to a confirmation of the hypothesis $H_{changeability}$ (see Section 9.2). This means that there are in practice MDE settings that affect this hard changeability concern, while there are also seem to be MDE settings that do not affect this hard changeability concern. This diversity in MDE settings from practice confirms the need for analysis techniques for the hard changeability concern *unexpected loss of content*.

10.2.3. Discussion

In Section 7.3 an approach for analysis in context of the practical application of MDE settings was presented. As shown in Figure 10.4 the different use cases for analysis are supported by the presented analysis techniques. Unfortunately, it was not possible to actually apply the analysis techniques during projects in practice in order to choose or tailor processes, refactor MDE settings, or plan evolution. Thus, it remains subject to future work to evaluate the benefit of adapting MDE settings, their context, or use according to the analysis' results.

However, as indicated in Figure 10.4, we applied the analysis techniques as a *post mortem analysis* to eleven MDE settings from practice in Chapter 9. Thus, although the MDE settings are partly still in use in practice, the analysis results are not used to modify the MDE settings or the way they are applied in practice (i.e. how they are combined with software development processes and how developers use the MDE settings). As discussed above, this confirmed the applicability of the analysis techniques and also the need for the analysis.

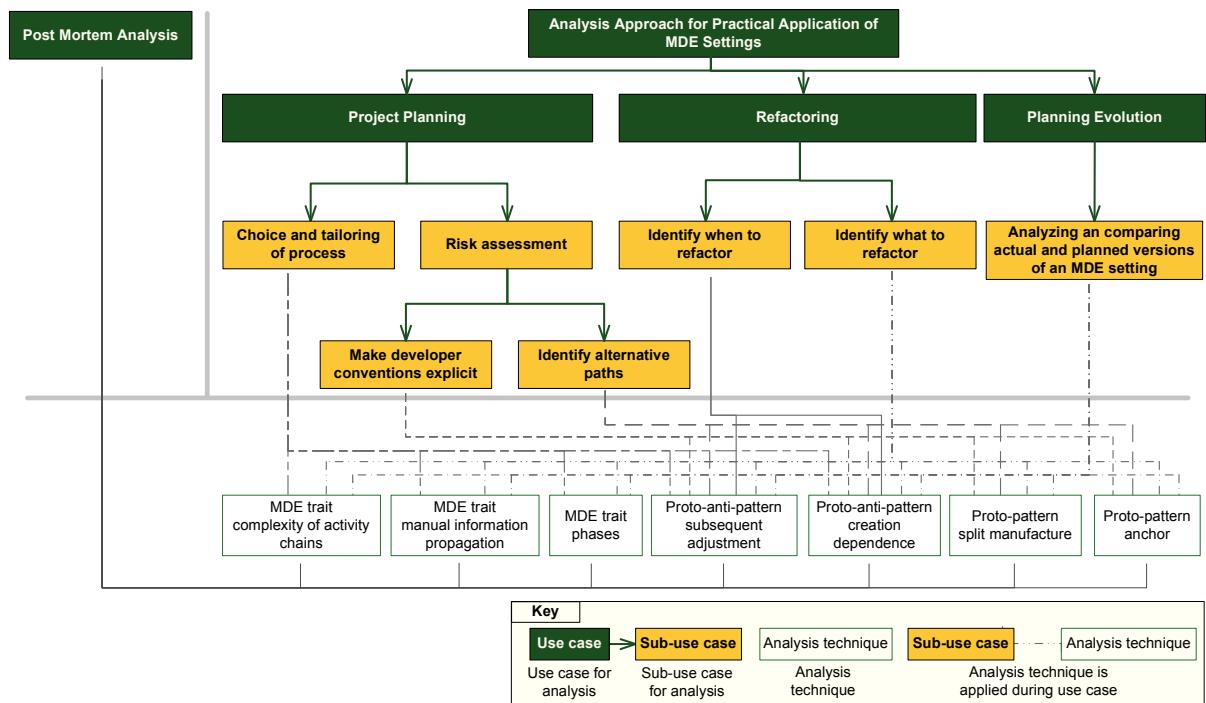


Figure 10.4.: Overview of the use cases for analysis.

Part IV.

Roundup

11. Related Work

Within this thesis, different aspects of MDE are studied and analysis techniques for MDE settings are provided. In this chapter, related work on the different contributions of this thesis is presented and discussed. In this context, several MDE settings from practice were captured and discussed in the Chapters 4 and 5. Therefore, related work on studies about the state of the art of MDE in practice is discussed in Section 11.1. In order to further investigate the different MDE settings the Software Manufacture Model language for representation of MDE settings was introduced in Chapter 6. Related work on modeling and analysis approaches for MDE settings is discussed in Section 11.2.

Both, capturing MDE settings from practice and introducing a modeling language for MDE settings was done in order to support the investigation of the three objects of this thesis: MDE's influence on changeability (Figure 11.1a), the interrelation of MDE and software development processes (Figure 11.1b), and the evolution of MDE settings (Figure 11.1c). Firstly, how MDE influences changeability was theoretically discussed in Chapter 3. Therefore, related work on theoretical explanations for the influence of MDE on changeability and related work on empirical studies about the actual degree of success and impact of MDE is discussed in Section 11.3. Second, the interrelation of MDE and software development processes was theoretically discussed in Chapter 3 and analysis techniques on relevant MDE traits are introduced in Chapter 7. Correspondingly, related work on process interrelations is discussed in Section 11.4. Finally, the evolution of MDE was theoretically discussed in Chapter 3 and studied in Chapter 5. Related work on the topic of evolution shall be discussed in Section 11.5.

This chapter is partially based on [P3, P2, P6], and [P4].

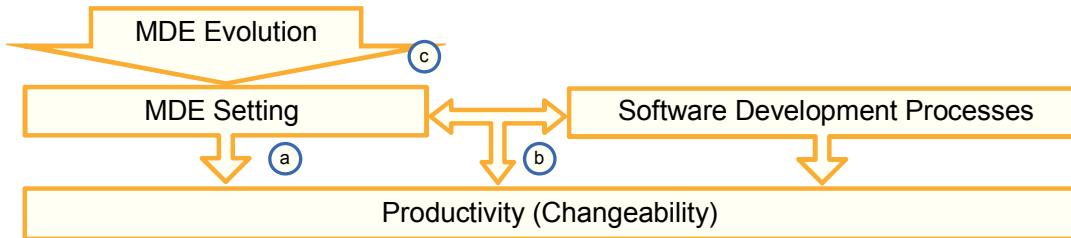


Figure 11.1.: Overview of investigated aspects of MDE settings

11.1. Related Work on Studies about MDE in Practice

There is much literature about the manifestation of MDE in practice that focuses on capturing and describing case studies in detail. For example, Guttman et al. collected in [94] 6 case studies about the introduction of MDA in different companies. Also Brown describes two case studies on MDA in [37]. In addition, this paper includes a list of state of the art MDA tools. Backer et al. list languages and automation steps that are used within a case study of Motorola [15]. Although automation is used for simulation as well as generation of code and test cases, Backer et al. subsume that manual coding still plays an important role in this case study. Heijstek et al. present a case study from a Dutch IT service provider in [95]. They list the frequency of different UML diagram types in use. They observe that in this case study activity diagrams and class diagrams are used in most cases. Also Mohagheghi et al. present in [146] three case studies on MDE and Mellegård et al. present a detailed case study on MDE in automotive domain [135]. Mellegård et al. also examined what effort was put in the different documents.

All of these case studies are presented with textual descriptions of the used languages and automations. Mohagheghi et al. and Mellegård et al. provide even simple models or illustrations of these MDE settings in [146] and [135]. However, while all works focus on automation and used artifacts, explicit descriptions of the required manual tasks are rare.

11. Related Work

Besides such case studies, some work in literature aims at capturing a broader picture of MDE. First of all, in 2008 Mohagheghi and Dehlen collected in their survey paper information from different case studies [143]. They found that automation was (at this point in time) most often used for code generation. Further, they identified a lack of executable models and summarized that analysis was most often performed on code level. Other works base on surveys and interviews among practitioners. For example, Kirstan et al. performed in 2009 twelve interviews in car industry [114]. As a result, they identified that techniques like prototyping and model in the loop are extensively used. Fieber et al. examine software development at Siemens [71]. One insight of their interviews was that models are mostly used for generation within Siemens.

Finally, some studies focus on questionnaires to collect data that is not specific to a single company. In their survey with more than 100 software practitioners Forward et al. came to the result that code generation is used seldom, while modeling is most often applied to plan and discuss the design of the software. Hutchinson et al. found in their survey, that UML class diagrams and activity diagrams are most commonly used [99]. Further, they made the observation that the use of MDE can be rather pragmatic. To illustrate this insight, they provide a transcription of an interview. This transcription describes how complex different languages are combined in a practical example. Finally, Torchiano and Tomassetti et al. examined in 2011 the use of models in the Italian software industry [196, 195]. 13% or the responding practitioners stated that they always used models, while 55% only sometimes do. UML is most often used (70%) and code generation is used in 44% of the cases, while model interpretation and transformations are not so often in use.

Summing up, while these broader studies can provide information about the frequency of model use, they cannot provide a detailed picture how models and automation steps are actually composed. Therefore, the publication of case studies remains important for research on MDE characteristics.

In contrast to related work, the case studies captured in context of this thesis have a main focus on the structure of MDE settings, including the structure of manual and automated activities, in practice.

11.2. Related Work on Modeling and Analysis of MDE Settings

In this Chapter, related work on modeling and analysis approaches for MDE settings is discussed.

11.2.1. Modeling of MDE Development

The Software Manufacture Model language that is presented in this thesis is designed, such that activities and their impact on the interrelation of artifacts can be expressed. In the following, related work on modeling approaches for MDE as well as modeling approaches for processes is discussed.

Modeling MDE

Approaches to model MDE have different perspectives. First of all, there are approaches that focus on the comparison of MDE tools. For example, Atkinson and Kühne discussed concepts for modeling and comparing architectures of modeling tools in [11]. They focused on the question how different modeling concepts (such as the relation between different meta-model layers) are implemented. Another example is the modeling language TIL that enables the comparison and analysis of whole tool chains in MDE [28]. TIL focuses on the question how different languages and formats are integrated.

Apart from the tool perspective, megamodel approaches focus on the question how different artifacts are interrelated (as summarized in Section 2.1.2). As indicated in the survey paper in [P5] the execution of model operations, if captured, is expressed in form of an own relation by megamodel approaches. No information how “static”-relations and “overlap”-relations between artifacts change is provided. This observation still holds for more recent megamodel approaches, such as in [65].

In contrast, research approaches for the characterization of activities or transformations often focus on the question what artifacts are consumed and produced. For example, Vignaga et al. present an approach for typing relations in model management [204]. They focus on the problem that transformations might be input of (higher order) transformations and introduce an approach that allows hierarchies in the types. Favre and NGuyen go a step further and define a set of basic relations that can occur between models or systems, e.g. “DecomposedIn”, “ConformsTo”, or “IsTransformedIn”. They use these basic relations to formulate patterns that capture parts of the software evolution process [66]. In contrast to the approach

presented in this thesis, “overlap”-relations or explicit references between artifacts are not discussed. Similarly it is not differentiated between manual and automated transformations. Nonetheless, Favre and NGuyen present with this approach a possibility to capture, formalize, and differentiate software evolution activities, such as reverse engineering or round-trip evolution.

Subsuming, similar to Software Manufacture Models the discussed approaches aim at differentiating between MDE activities. However, in contrast to the discussed approaches, the Software Manufacture Model language provides concepts to express how manual and automated modeling or transformation activities affect changes in artifact relations.

Modeling Processes

The main focus of most process modeling languages is the illustration of control flow or object flow. Activities are characterized by names as well as the artifacts that are consumed and produced. Examples are SPEM [2] and languages that base on SPEM [24, 25]. Engels and Sauer present with MetaAME an alternative approach to model software engineering methods [58]. In addition, textual annotations might be used, such as in [40] or [183].

Approaches, like the six UML-based languages for software process modeling surveyed in [26], can be used to capture activities on a more detailed level. These approaches characterize activities by input and output artifacts. MODAL [117] also focuses on this more detailed view. The language SPEM is extended with concepts to deal with model based engineering. Like the approaches surveyed in [26], the authors of MODAL consider the involved artifacts within the specification of activities. In addition, they use further activity diagrams to specify activities more in detail.

The SimSWE approach of Birkhölzer [30] introduces a library for activities to enable reuse for the simulation of processes. The activities are mainly characterized by aspects that are important for the simulation, like task size, productivity, or communication overhead. However, there is no consideration of artifacts that are produced or consumed. Esfahani et al. present in [60] a repository for agile method fragments, where a method fragment is considered at the high abstraction layer of pair programming or daily scrum meeting. Here artifacts are not taken into account. Jacobson et al. describe in [101] the vision of the SEMAT project, which is to provide a common theory and terminology to describe methods in software development. Due to this comprehensive goal, the key concepts (“kernel elements”) are very high level (e.g. software system, team, work, or requirements) and (so far) have no specific focus on MDE development.

Only some process modeling languages are directly designed to support MDE concepts. For example, Diaw et al. (SPEM4MDE [51]), Porres and Valiente ([164]), and Maciel et al. ([129]) present extensions of the process modeling language SPEM. Goal of these extensions is the creation of process modeling languages that allow the automated execution of code generators or transformations.

Subsuming, in order to guide developers or automatically execute automated activities, the discussed approaches aim at capturing activities, similar to Software Manufacture Models. However, in contrast to these process modeling languages, the Software Manufacture Model language allows to capture and reason about changes in the relations between artifacts, which is crucial for analyzing correctness and consistency.

11.2.2. Analysis Techniques for MDE Settings

Models of software development can be used for an analysis how productivity and quality will be affected. For example, Förster et al. present in [75] and [76] and pattern-based approach to model and verify, to what extent quality requirements from ISO standard 9001 are fulfilled by a process.

This section deals with related work on the question how an MDE setting’s impact on productivity and quality can be analyzed. First of all it is necessary to distinguish between the analysis of *model quality* and the analysis of the quality of an MDE approach. As summarized by Mohagheghi et al. [144], the term quality is in context of MDE often used to refer to the quality of the models that represent the software system. For example, Monperrus et al. present a framework that can be used to declaratively specify metrics on models. The framework then automatically generates corresponding measurement software [147]. This means that the approach supports the development of metrics to evaluate software at the stage of modeling, already.

11. Related Work

However, this thesis focuses on the quality of an MDE approach. Fieber et al. introduce a corresponding taxonomy for model quality in [69], with a main focus of software - and documentation models. Although criteria for the quality of a whole MDE approach are not discussed systematically, some aspects of the quality of a modeling notation are mentioned, e.g. the *degree of formalization* and the *adequacy for the application domain*. However, research that actually deals with the question how MDE approaches can be evaluated is rare.

An approach for the estimation of costs that are to be expected when implementing a project with a specific MDE setting is presented by Sunkle and Kulkarni. This approach is a proposal how the COCOMO II model can be used for cost estimation of model-driven development tool sets [193]. The main idea of the approach is to encrypt the impact of MDE characteristics in terms of their effect on COCOMO Scale Factors and Effort Multipliers. The rating levels for these cost drivers are determined with the help of a questionnaire that is answered by developers that have years of experiences with the MDE setting under study. Further, the estimation of system size is adapted by introducing individual adjustment factors for code that is generated from different languages. The approach is presented on the example of a specific MDE setting. MDE characteristics such as abstraction (e.g. separation of functionality or component abstraction), automation (e.g. documentation generation, automated consistency validation, or configuration management support), and maturity of tooling are considered.

Subsuming, Sunkle and Kulkarni present the probably first applicable approach for a quantitative cost estimation that is individual for an MDE setting. Naturally, the approach comes with some constraints. First, the evaluation what MDE characteristics are relevant for what cost driver, remains a research challenge. Here experience is required to apply the cost estimation. Second, the proposed assignment of the rating levels bases on experience of the developers. Thus, the approach, as is presented, is not applicable to new or adapted MDE settings. Finally, Sunkle and Kulkarni make a pragmatic decision for size estimation on the basis of the resulting lines of code, which is probably applicable for the most MDE settings. However, the approach cannot be applied when MDE settings include interpretation of models (i.e. when no code is generated). Another approach to apply COCOMO is presented by Biehl and Törngren, who estimate costs savings when tool chains are used. They consider cost drivers, such as reuse of tool adapters, when using automated tool chains [29].

A qualitative approach that aims at the goal to evaluate changeability or maintainability of an MDE approach is presented by Domínguez-Mayo et al. They introduce in [53] a framework to study maintainability of model-driven web methodologies. For different maintainability aspects, such as changeability, it is listed what MDE characteristics have an influence. Relevant MDE characteristics are levels of abstraction, standard definition, transformations, traces, and model based testing. For the evaluation it is rated which of the relevant MDE characteristics are supported by a model-driven web methodology.

The framework is very simple and specific to model-driven web methodologies. For example, it is not considered that certain characteristics of MDE can have positive as well as negative influences on productivity.

In Section 3.2.4 it was discussed that an approach for the overall evaluation of changeability is difficult to achieve. This is the reason why it is not a part of this thesis to aim at the same high goals as Domínguez-Mayo et al. or Sunkle and Kulkarni. Instead this thesis focuses on the examination of the impact of single MDE characteristics on hard changeability concerns.

Apart from these two approaches that aim at a direct evaluation of properties such as costs or changeability, there is a small amount of related work that aim to classify modeling approaches concerning their maturity. These maturity models define levels that modeling approaches might fulfill. A well known approach is Brown's modeling spectrum [37], which was already introduced in Section 2.1 (Figure 2.1). Comparable to Brown's modeling spectrum, Warmer and Kleppe described six Modeling Maturity Levels in [209]. While the modeling stages of Brown focus on the interrelations between code and models, Warmer and Kleppe mainly focus on the quality and level of detail within models.

While these first two approaches describe rather theoretical views, Backer et al. present in [15] the modeling challenge levels (MCLs) framework that captures experiences from different developers and teams within Motorola. The goal of the MCL framework is to communicate experience and to provide a road map for the introduction of modeling. The six presented MCLs reflect stages of the use of models that were traversed by different teams, starting from *no modeling*, via *informal and formal modeling*, up to *optimized model-driven engineering*. Unfortunately, Backer et al. present no details about how different modeling approaches look like during these stages. A final approach towards an MDD maturity model was presented in 2006 by Rios et al. [168]. In contrast to Backer et al., Rios et al. present

detailed information on the structure and differences between the different maturity levels. Although the idea of maturity models for MDE is promising, all these approaches suffer from missing description of risks and benefits that are associated to the different stages. Thus, they cannot be used to evaluate the changeability or maintainability of an MDE approach or MDE setting.

Subsuming, only few research work focuses on the evaluation or classification of the quality of a given MDE setting. Furthermore, the available approaches suffer from the complexity of this challenge. For this reason this thesis focuses on a qualitative and partial approach (i.e. only specific changeability concerns are addressed).

11.3. Related Work on MDE's Influence of Changeability and Productivity

This section focuses on related work on MDE and productivity (especially changeability). This thesis approached the question how structures in MDE settings affect changeability in two steps. First, possible influences have been examined theoretically (see Section 3.2). The second step was to empirically investigate how MDE settings from practice differ in some of these relevant structures and with it in their influence on changeability. Therefore, related work on theoretical explanation for MDE's influence on changeability is discussed in Section 11.3.1. Afterwards, empirical studies on the impact of MDE on productivity are examined.

11.3.1. Theoretical Explanations on MDE's Changeability Influence

In Section 2.2 it was already summarized how the effect of MDE on different productivity aspects, such as reusability, portability, interoperability, complexity, and quality, is explained in literature. In the following, related work on theoretical explanations for MDE's influence on changeability is discussed.

Stahl et al. [188] argue that *changeability* and *Maintainability* is supported in MDE by separation of concerns. Separation of concerns ensures that changes do not have to be applied in all models of the system. As Kelly et al. [110] argues, a setting where platform aspects are separated and introduced by a fully automated transformation allows reaction to changes in platform requirements, since only the transformation has to be changed and reapplied to all build systems. Further, Kleppe et al. [115] and Gruhn et al. [92] argue that abstract models that are automatically transformed to code solve the problem of keeping a consistent high level documentation. A good documentation is necessary to ensure maintainability and changeability.

Mohagheghi et al. discuss in [144] that the development process and modeling conventions can influence the changeability. Further, they argue that generating models from models can also affect changeability positively (due to the automation). Some works focus on how generated code can be embedded with manual code, such that handwritten code is preserved in case the generation has to be performed again, e.g. generation gap pattern presented by Fowler [78] or protected regions as discussed by Stahl et al. [188]. Almeida et al. [5] theoretically discuss, that a higher number of intermediate models (such as PSM in MDA) and corresponding transformations, enables a better reuse of transformations, while the efficiency of the design process might be decrease with an increasing number of intermediate models. Finally, Hutchinson et al. present a table that lists for different productivity and maintenance aspects possible positive as well as negative impacts that might be caused by MDE [99]. For example, the time to develop code might be reduced by automated code generation, while it might be increased by the required additional time for the implementation of model transformations.

It can be seen that there is a difference in the discussions of the impact of MDE. While most works discuss how MDE can lead to benefits, some researchers also focus on possible disadvantages of MDE (such as Fowler [78], Stahl et al. [188], Almeida et al. [5], and Hutchinson et al. [99]). Apart from single arguments, the variety of changeability concerns that might be affected by MDE is not explored to the same extent in the discussed related work as in this thesis. In contrast, this thesis approached a systematic discussion on MDE's impact on changeability in Section 3.2. Section 3.2 has a focus on the creation of the software product and leaves impacts on the productivity of development of the MDE setting out. Within this focus, the discussion in Section 3.2 provides a frame for the single arguments that are discussed in related work. Moreover, only one of the here discussed works ([99]) combines the theoretical consideration with an empirical investigation of the occurrence of the effects in practice. However, while Hutchinson et al. [99] focus on the effort of single development tasks, this thesis focuses on effects that are caused by the structure of an MDE setting.

11.3.2. Empirical View on MDE Productivity

There are several studies, dealing with the question how model-driven engineering or model-driven development influences software development. The results of most of them until 2008 are summarized in the review paper “Where is the proof?” of Mohaghegi et al. [143]. This review shows that there is no final proof for most of the common assumptions about MDE’s influence. For example, there are studies supporting the assumption that MDE enhances productivity, while other studies state that MDE reduces productivity. Also a positive influence on software quality is not empirically proven. Since then, several studies on the impact and success of MDE were published.

Studies that Focus on Productivity and Quality Gains

In order to study the benefits of MDE on productivity and quality, different studies were performed and published. A main part of these papers deals with single case studies or the comparison of two or three case studies. Further, some researchers performed experiments to study the productivity and quality gains. Finally, three questionnaire based studies collected insights from more than 100 software practitioners, each.

Literature that presents case study based research often focus on qualitative aspects and experiences of developers. For example, Kapteijns et al. discussed different factors, such as the amount of non-generated features that influence the productivity increase [107]. For the considered case study they measured a performance increase of 2.6 times compared to a legacy project. Also Weigert et al. report on an improvement of productivity within in Motorola [211]. Backer et al., who considered several projects with MDE at Motorola, even report on detailed numbers such as up to 33% effort reduction, 65% code generation, or a 30 to 70 times reduction required time for bug fixing [15]. Vogel performed three different studies (including a case study) during the introduction of MDE in a small to medium-sized enterprise. He reports on a productivity gain from up to 57% for maintenance projects [205], but concluded also, that for some development project there might be no productivity gain at all. Thus, not all studies provide proves for an increased productivity. For example, Kirstan et al. – although aware of the potentials – came to no clear result about actual productivity improvements in [114]. Also the report of Backer et al. include information about one pilot project that failed, which is explained with the use of manual transitions and informal models [15]. Also possible quality gain is in focus of research. Here, Weigert et al. identified a higher fault discovery rate in early phases of development (compared to the legacy base line) [210]. Similarly, within an industrial case study Nugroho et al. found a reduced fault rate in classes that were modeled with UML compared to classes that were not modeled [152]. Heijstek et al. studied the influence of model size and model complexity on effort during development. In addition, software quality is examined on the basis of a case study. They found no correlation between the fault density and the complexity and size of the models.

The few experiments, that were performed, have a different focus. Mellegård et al. studied the benefit of the use of graphical modeling in requirements engineering. The experiment shows, that the impact of changes can better be predicted on a graphical model and that less time is required for the application of changes [136]. Similarly, Martínez et al. found in an experiment that the time required for maintenance tasks was more predictable, with a modeling approach [133]. However, they also found that developers had a greater trust in stability of changes that they performed on code level.

Finally, some studies aim at collecting a boarder view on MDE in practice. Therefore Forward and Lethbridge [77], Hutchinson et al. [99], and Torchiano et al. [195, 196] used questionnaires to collect experiences form software practitioners. Forward and Lethbridge published their results on the opinions of 113 software practitioners in 2008. They observed the tendency that a better validity of the software can be reached, when models are used.

In 2011 Hutchinson et al. published results from a questionnaire with 250 responses and additional interviews [99]. They retrieved detailed responses on the experiences with productivity and maintainability increase associated to differed uses of models. For example, more than 70% of the responses confirmed an increase in productive, when models are used for team communication and problem understanding. Further more than 60% of the responses confirmed that productivity increase when models are used for code generation. However, less than 50% of the responses agreed with a productivity or maintainability increase when DSLs are used. In addition, Hutchinson et al. studied the balance between specific disadvantages and advantages of MDE techniques, such as code generation.

Finally, Torchiano and Tomassetti et al. published a survey about the state of the art in the Italian industry in 2012 ([195]) and 2013 ([196]). They collected responses from 155 software professionals. One observation was that there is a statistically significant difference in the productivity gain and reached platform independencies, when comparing the use of models, only and the use of models in combination with modeling technologies such as code generation. Further, they collected the experience that company specific tooling leads to a higher flexibility and productivity.

Studies that Search for Factors that Support MDE Success

Besides the focus on the productivity gain, many studies focused collected factors that influence the success of an adoption of MDE in practice. For example, Staron compares in [190] two case studies – one with a successful adoption of MDE and one where the company stopped the adoption of MDE. Based on data from interviews and a questionnaire, motivations for the adoption, such as the ability to estimate costs or improve quality, are collected. Finally Staron compares both cases and identifies certain factors that play a role in the adoption. These factors include the maturity of tools and techniques, the compatibility to the software processes, the skills (especially language engineering expertise) as well as the way MDE is adopted. Also Weigert et al. and Hutchinson et al. identified modeling skills as a relevant factor ([210],[99]).

Mohagheghi et al. examined different case studies with the focus on factors that influence the adoption of MDE ([146], [145]). Amongst other challenges, they identified a need for user friendly tools and a good language design. That the used tools are an important factor is also supported by the results of Backer et al. [15] and Chaudron et al. [41].

Another factor that was observed in a case study by Kuhn et al. is that short cycles are required, especially when the built software has an experimental character [119]. To provide a frame for different studied factors, Whittle et al. provide a taxonomy of factors that capture impact of MDE tools on MDE adoption. This taxonomy is a result data from almost 40 interviews with practitioners [213]. Finally, Torchiano and Tomassetti et al., collected potential problems for the adoption of MDE, *including missing competencies* (developer skills) and *the fear of a vendor lock-in* ([195], [196]).

Discussion

Many of the findings regarding the increase of productivity and quality are positive, while there are still results that are ambiguous. This observation that was already made by Mohagheghi et al. [143] fits to this results of this thesis. From the viewpoint of the observation that MDE settings are diverse in practice (Chapters 5 and 9) it would rather be surprising to find similar impacts on productivity in all applications of MDE.

Further, most of the studies on relevant factors in MDE adoption focuses on maturity and usability of tooling as well as social or organizational aspects like the willingness of developers and management to adopt a new development approach. Most existing studies or experience reports do not focus on the design of MDE settings (i.e. on the interplay of different MDE technologies). Thus, only some of the studies consider the structure of the MDE setting as a possible influence on the success of MDE adoption or productivity improvement.

A first example for a discussion at the level of MDE technologies is the work of Kuhn et al. [119]. They discuss the need of short cycles for the examined case study. This, finding can be considered as an example how the MDE trait *complexity of activity chain* (see Section 3.3.3) can lead to problems in practice. The discussion of the failed pilot study by Backer et al. is even more interesting. The authors explicitly search for reasons within characteristics of the used MDE setting: the use of manual transitions (considered as MDE trait *manual information propagation* in this thesis (Section 3.3.3)) and the use of informal models [15].

Finally, only Staron explicitly discusses the compatibility to the process as an important factor in the introduction of MDE. Staron argues that a need to adapt the process increases the effort of the MDE adoption [190]. This observation supports the need for the discussion of process interrelation in this thesis. All, these related studies use the empirical data to identify potential factors that might influence the success of MDE.

In contrast to these studies, the factors that are studied in this thesis have been identified in a systematic theoretical consideration, while the empirical studies are used to explore the diversity of

manifestations of these factors in practice. A similar approach can be found by Hutchinson et al., where it is considered that MDE technologies, such as code generation, have potential positive as well as negative impacts [99]. Similar to this thesis, Hutchinson et al. use an empirical study to investigate the manifestation of the theoretically discussed effects in practice. However, in contrast to this thesis, Hutchinson et al. focus on different factors than the structure of MDE settings (as discussed above in Section 11.3.1).

11.4. Related Work on Process Interrelation

The question how MDE and software development processes are combined is addressed by some works on the general question how combinations can look like, only. Similarly, systematic research on the impact that MDE can have on a process is seldom. In this section it is first discussed how related work on combinations fit into the *reference model* presented in Section 3.3.1. Afterwards, related work on the mutual impact of MDE and software development processes is discussed and compared to the MDE traits presented in this paper.

11.4.1. Combination of MDE and Software Development Processes

In the following, related work on the combination of MDE and software development processes is discussed. How MDE generally can be combined with or embedded in software development processes is rarely addressed in literature. For example, Kent discusses in his seminal work that MDE is accompanied with a micro process (capturing the manipulation of single models) and a macro process (capturing the order of model manipulations) [111]. However, there is no discussion how macro and micro processes are interrelated with software development processes. Also the more actual research road map from France and Rumpe includes no discussion of this aspect [79].

After all, Stahl et al. discuss that MDSD (model-driven software development) can in principle be combined with every process or method that bases on a iterative approach [188]. The basic assumption of the combination is that modeling activities are applied during implementation phases of the respective process. Whether a modeling activity can be used to support other phases, too, is not discussed in [188]. Kleppe et al. describe a similar understanding in [115]. Also Engels et al. assume that all processes can be combined with a model based development approach [59].

However, not all literature assumes that MDE or MDA will be combined with existing standard processes. For example, Asadi and Ramsin surveyed several proposals for methodologies that are specifically defined for MDA [9].

To summarize, only Stahl et al. discuss a general view on the combination of MDE and processes that can be compared to the *reference model* from Section 3.3.1. However, the possibility that MDE artifacts are mapped to additional process documents and MDE activities are used in multiple phases of the process is not discussed in [188]. In addition, the described MDSD approach bases on a relatively fixed set of activities (modeling, transformation or generation, and manual implementation), while the *reference model* presented in this thesis bases on no assumptions on the concrete structure of MDE activities.

11.4.2. Impacts between MDE and Software Development Processes

In the following, related work with a direct focus on the impact of MDE on software development processes (and the other way around) is discussed. For example, Engels et al. expect that the applied process has impact on the quality of the different development artifacts [59].

Heijstek and Chaudron examined in [96] a case study with focus on the impact of MDE on software development processes. They identified 14 factors that impact the architectural process. The concrete structure of the studied MDE setting is not described in detail, but seems to include a model-to-code generation. In addition, the authors make the assumption that it is easier to implement changes with the studied MDE setting than with code, only. The identified factors mainly concern aspects like impacts on communication, required skills, and tooling. One of the most interesting factors is the identified increased of the need for collective code ownership. All in all, the results base on a single case study and thus on a single MDE setting.

Besides this targeted study, also some qualitative studies on MDE led to the conclusion that the interrelation to the process is important. For example, Staron identified the importance of compatibility between processes and MDE, too [190]. He comes to the conclusion that an introduced MDD approach should not lead to the redefinition of the applied process. Aranda et al. [8] performed interviews in a company that changed their development to use MDE. They found that division of labor changed within the company. The identified reason for that result is that models enabled non-software engineers to take over parts of the work that was formerly performed by software engineers.

Finally, Kalus and Kuhrmann surveyed literature to get a collection of criteria for software process tailoring in [106]. Identified criteria are for example, the used programming language, tool infrastructure, and complexity. However, the MDE traits identified in this thesis do not yet appear within this summary of criteria for process tailoring in literature.

In [96] Heijstek and Chaudron make a first important step towards the examination of the interrelation of MDE and software development processes. In contrast to their case study based research approach, this thesis contributes with a theoretical consideration of the relationship between MDE setting and processes. Further, this thesis provides the insight that MDE settings differ strongly in their MDE traits and with it in their influence on the processes.

11.5. Related Work on MDE Evolution

This section focuses on related work on evolution in context of MDE. Therefore, first categorizations and surveys on evolution are discussed. Afterwards, related work on the introduction of MDE and on schema evolution are shortly discussed.

There are different surveys and research agendas that classify types of changes in model-driven software evolution or in software evolution in general. For example, Stammel et al. provide an extensive survey on techniques and approaches in software engineering that enable flexibility and evolution of software [189]. The whole spectrum of approaches reaching from agile methods via model-driven software development to architectural patterns is discussed on a high level of abstraction.

A collection of possibilities to support software evolution with models is presented by Karanam and Akepogu in [109]. However, within MDE approaches models are also part of the implementation of the software and need to be evolved when the software evolves. For example, Engels et al. provide support for the preservation of consistency among development artifacts by describing model evolution in terms of transformations [57]. Khalil and Dingel present in [112] a survey on techniques that support the evolution of UML models. They consider model evolution as a subset of software evolution. Thus, the discussed approaches deal with evolution of the different models that are used for the specification of a software system. Discussed evolution tasks concern for example the change propagation between different models.

That evolution in MDE can be more than evolution of the software is summarized by van Deursen et al. in [201]. Here, challenges in supporting the different forms of evolution are discussed. Considered forms of evolution are “regular evolution”, which is software evolution, “meta-model evolution”, which is a form of language evolution (discussed as change type *C2* in this thesis), “platform evolution”, which means the evolution of code generators and tools (discussed as change types *C1* and *C3* in this thesis), and “abstraction evolution”, which means the introduction of an additional modeling language to the set of languages (discussed as change type *C5* in this thesis). “Abstraction evolution” as is introduced by van Deursen et al. is a special case of structural evolution with the potential to affect change types *C4* and *C6*, when the introduced additional language is used to express additional artifacts roles (and does not only substitutes the language that is used for an already existing artifact role). In [137] Mens et al. supplement the challenges listed by van Deursen et al. with a stronger focus on the evolution of models.

Finally, Corrêa et al. classified change types for changes that are associated to software product lines [47]. The categories include the change of the meta-model (summarized as language evolution *C2* in this thesis), changes of features and changes of models, which are both forms of software evolution, and changes within a transformation (discussed as change type *C1* in this thesis).

Subsuming, most categorizations have a central view on software evolution, which is as such not part of the categorization in this thesis. Further, non-structural changes are considered in most categories, while only in [201] a very specific form of structural changes is considered.

11. Related Work

An area of research that should be mentioned in that context is research on the introduction of MDE into existing projects or to maintain already existing software. This goal was introduced as one of the architecture-driven modernization scenarios in context of the architecture modernization initiative (ADM) of the OMG [158]. Mansurov and Campara present in [131] a concrete approach to introduce MDA into an already running project to maintain the created software. This introduction might be seen as a special form of evolution.

Finally, evolution has a long history in data base research. Here schema evolution and the co-evolution of data were studied. As summarized by Roddick in 1992 in an annotated bibliography, much research was done on schema evolution [170]. Schema evolution can be compared to language evolution or even to co-evolution of meta-models and models. For example, Lerner presents a framework for the automated creation of transformations for migrating data from one schema to a new one [125]. Noy and Klein discuss in [151] the difference between ontologies and database schemata in order to identify additional challenges for the evolution of ontologies and corresponding co-evolution of data.

From the viewpoint of the types of MDE evolution that are discussed in this dissertation, research on schema evolution focuses on the non-structural change type *C2*. The schema can be seen as the language, while the data can be compared to instances of this language. To sum up, related work on evolution mainly focuses on non-structural forms of evolution. In contrast, this thesis introduced a classification of structural evolution.

12. Conclusion

In this chapter the results of this thesis are summarized. Subsequently, it is discussed how the goals of this thesis are fulfilled and what implications arise from the gained insights and observations.

12.1. Summary

In this thesis, characteristics of MDE settings in practice were studied. This includes the influence of an MDE setting on changeability, the interrelation to software development processes, and evolution.

First, these topics were approached on a theoretical level by analyzing the problem domains. It was discussed how changeability can be affected by MDE. In this context the notion of *changeability concerns* and *hard changeability concerns* was introduced. The main insight of this consideration is that MDE settings can affect additional changeability concerns, compared to single languages or to a system's architecture.

Further, it was discussed how MDE settings and processes are interrelated. In this context a generic reference model was introduced to refer to possible combinations of MDE settings and software development processes. Additionally, the notion of *MDE traits* was introduced in order to refer to an MDE setting's characteristics that can, depending on their manifestation, lead to constraints on software development processes. As a result, three MDE traits were selected for further consideration: *phases*, *manual information propagation*, and *complex activity chains*.

As a final topic the evolution of MDE settings is examined. This includes the presentation of a classification of possible change types. Further, it was discussed how the characteristics of an MDE setting (i.e. the influence on changeability and other productivity dimensions as well as the manifestation of the different MDE traits) are affected when different change types are applied on or happen to an MDE setting. The notions of *non-structural changes*, *structural changes* and *substantial structural changes* were introduced. The main insight of this discussion is the different consequences that structural changes and non-structural changes can have on how an MDE setting affects productivity dimensions. An MDE setting's effects can be altered for more productivity dimensions if structural changes are applied compared to non-structural changes. Similarly, the extent to which an MDE setting's effect is altered for a specific productivity dimension is potentially stronger for structural changes compared to non-structural changes.

Subsequent to these discussions, three hypotheses were formulated on the diversity of manifestations of MDE traits in practice (H_{traits} with sub-hypotheses H_{phases} , $H_{manualInformationPropagation}$, and $H_{complexActivityChains}$), on the diversity of the influence of MDE settings on changeability in practice ($H_{changeability}$), and on the existence and commonness of structural evolution in practice ($H_{existence}$ and H_{common}).

In order to approach an evaluation of these hypotheses, three studies were performed to collect data about MDE settings and their evolution in practice. Together, these three studies provided models of 11 MDE settings from 4 companies, evolution histories from seven MDE settings from five companies (spanning 33 evolution steps), several records on experiences with the MDE settings and motivations for evolution steps, as well as 7 literature reports on MDE settings from practice. Further, the Software Manufacture Model language is introduced as a modeling language that allows the capture of MDE settings. To provide the possibility for documenting and communicating experiences with structures in MDE settings, the Software Manufacture Model pattern language is also introduced.

On the basis of these two languages, novel techniques for the analysis of MDE settings are provided. This includes 2 proto-patterns and 2 proto-anti-patterns for the identification of structures in an MDE setting that affect changeability concerns positively or negatively. Further, simple techniques for assessing the manifestations of the three identified MDE traits are introduced. While the technique for *manual information propagation* bases on a simple Software Manufacture Model pattern, the techniques for the MDE traits *phases* and *complex activity chains* are based on an identification of *predecessor sets* of activities and artifacts.

12. Conclusion

To complement the modeling languages and analyses techniques, some tools are provided. These are editors for Software Manufacture Models and Software Manufacture Model pattern, as well as a pattern matcher which automatically identify matches of Software Manufacture Model patterns within a given Software Manufacture Model. Further, an automated identification of predecessor sets was implemented. On that basis, semi-automated support for the identification of (local) *phases* and the *length of activity chains* is provided.

The applicability of the Software Manufacture Model language and the analysis techniques was evaluated by applying them on the 11 captured MDE settings. This evaluation revealed several aspects. Firstly, the Software Manufacture Model language is applicable to capture MDE settings from practice. Further, the characterization of activities in Software Manufacture Models can support recognition of similar situations (with perspective on changeability concerns), even when they are differentiated in classical terminology. For example, the hard changeability concern *unexpected loss of content* might be affected by both, code generations and model transformations. Thirdly, the evaluation revealed that the analysis techniques are applicable to MDE settings from practice. Finally, the search for pattern matches revealed that the proto-patterns *subsequent adjustment*, *creation dependence*, and *split manufacture* occur several times in different companies. Therefore, these proto-patterns have the potential to become actual patterns.

By examining the data from the studies and applying the analysis methods on these data, the three hypotheses were evaluated. First of all, the hypotheses on existence and commonness of structural evolution in practice ($H_{existence}$ and H_{common}) are supported. Thus, it was concluded that structural evolution of MDE settings exists and is common in practice. Second, the hypothesis on the manifestation of MDE traits H_{traits} is supported by the results of the analysis of the 11 MDE settings from practice. Thus, *in practice the identified MDE traits occur within a spectrum that includes process-relevant manifestations as well as process-neutral manifestations*. Also the hypothesis on the impact of MDE settings on the hard changeability concern *unexpected loss of content* $H_{changeability}$ is supported by the results of the analysis of the 11 MDE settings from practice. Thus, *MDE settings in practice vary considerably in their influence on changeability*.

In addition to the evaluation of these hypotheses, the records and data from the interviews on evolution histories were used to collect 11 observations on the combination of changes during structural evolution, on trade-offs, and on motivations for structural evolution. These observations illustrate that structural evolution occurs for various reasons, which may apply for most MDE settings in practice.

This thesis has shown that there is the theoretical possibility that structural evolution has negative impacts and that structural evolution is common in practice. In addition, a final examination of the captured structural evolution steps revealed actual examples of structural evolution steps that introduced or removed proto-anti-patterns to MDE settings as well as structural evolution steps that changed the manifestation of MDE traits of MDE settings.

Finally, the discussion of related work illustrates the novelty of the modeling and analysis approaches and further evaluates the basic assumptions of this thesis. There is no modeling approach that focuses on similar aspects to Software Manufacture Models. Approaches to analyzing properties of an MDE setting in general or changeability in particular are rare. The consideration of related work on the combination of MDE with software development processes and actual approaches for such combinations (as shown in Section 3.3), shows that the *generic reference model* presented in this thesis captures all proposed forms of combinations. Further, the examination of this related work on process interrelation shows that all proposed approaches base on strong assumptions about the structure of the MDE setting or change the software development process (if a standard process is applied at all). Nonetheless, the absence of literature on process tailoring that considers MDE as relevant criterion emphasizes the novelty of the examination of process interrelation in this thesis. Finally, the examination of related work on evolution in context of MDE reveals that there is a lack of research that focuses on structural evolution in MDE.

12.2. Thesis' Goals

In the following, a short summary is given of how the goals of this thesis have been achieved (summarized in Figure 12.1). First **Thesis' Goal 1** (to provide a modeling approach that enables capturing of MDE settings, such that the effects of activities on artifacts and their interrelations can be documented) is addressed by the introduction of the Software Manufacture Model language.

Thesis' Goal 2 (to enhance the knowledge about MDE's influence on changeability) is addressed by the theoretical examination of changeability concerns and the examination and evaluation of hypothesis $H_{changeability}$, which revealed how MDE settings differ in their impact on changeability. Similarly, **Thesis' Goal 3** (to enhance the knowledge about MDE's interrelation with software development processes) is addressed by a theoretical examination of MDE traits as well as the examination and evaluation of hypothesis H_{traits} , which revealed the diversity of manifestations of different MDE traits in practice.

On this basis, **Thesis' Goal 4** (to provide analysis techniques for MDE settings that support the identification of risks for changeability as well as constraints on software development processes) was achieved for the hard changeability concern *unexpected loss of content* and the three identified MDE traits. The identification of the diversity of MDE settings concerning their influence on changeability and the manifestations of MDE traits fulfills **Thesis' Goal 5** (to identify candidate factors to explain the diversity in the degree of success of MDE in practice). The results show that differences of MDE settings in these characteristics are candidates for explanations of differences in MDE success or failure in practice. Consequently, these candidates should be considered in future research on the success of MDE.

Thesis' Goal 6 (to understand whether evolution in practice might change an MDE setting's impact on changeability and software development processes) was achieved by the examination of structural evolution. This includes the theoretical discussion of the potential effects of structural evolution, the empirical examination of the commonness of structural evolution in practice, and the identification of examples of structural evolution steps that resulted in the introduction of proto-anti-patterns and in negative effects on the manifestations of MDE traits.

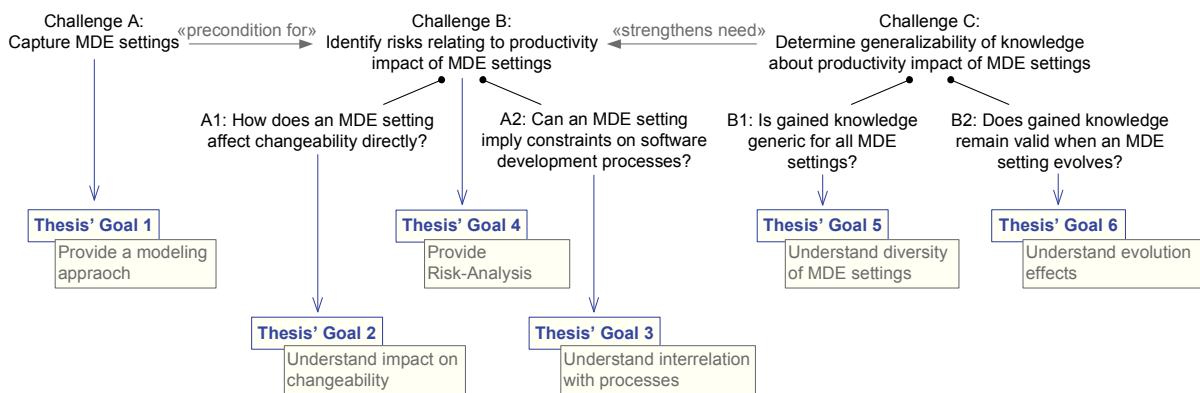


Figure 12.1.: Overview of addressed thesis' goals.

12.3. Implications

There is still much work to be done in understanding the impacts of MDE as well as in proving the benefits of MDE. The insights of this thesis have some implications on what future work would be beneficial, but also more generally on the focus of future research in MDE. These implications are discussed in the following.

12.3.1. Changeability & Process Interrelation

The influence on changeability and the process interrelation of an MDE setting have been identified in this thesis as candidates for factors that can explain the diversity of MDE success. In the following, implications of this result are discussed.

Change Research on MDE Success and Benefits!

First, research efforts to prove the benefits of MDE are affected. The observed diversity in the characteristics of MDE settings shows that it is impossible to certify that the concept MDE is beneficial in each manifestation. With the knowledge collected here, it is easily possible to artificially construct an MDE setting that affects hard changeability concerns (e.g. includes occurrences of the subsequent adjustment patterns) and that has process-relevant manifestations of the three MDE traits that are identified in this thesis. Such an MDE setting could be designed as counterexample for the theory that MDE is always more productive than non-MDE approaches – even if the applied modeling languages are very intuitive. However, the context of the application of the MDE setting is also relevant. An MDE setting with process-relevant manifestations of MDE traits might be applied successfully, e.g. in combination with the V-Modell, but fail when another process is used.

The main implication for research on MDE success is, that the question “Is MDE beneficial?” needs to be substituted by the question “When and why is an MDE setting beneficial?”. Fortunately, some researchers have already started to reformulate this question and to search for factors that influence MDE’s success.

A further implication concerns the comparability of case studies on MDE success or benefits. Currently, such studies mainly focus on social and organizational factors as well as on tools. To make such studies more comparable, it would be beneficial if researchers start to document the structure of MDE settings explicitly. This would help to exclude the possibility that certain negative effects are actually caused by the technical structure of the MDE setting. Especially, effects that stem from the interrelation of MDE settings with different aspects (such as the process, but also for example with organizational factors) cannot be identified without such a holistic view on the studied MDE approaches.

A final implication is that results from older case studies should not be transferred or generalized without discussion. Such discussion should capture potential impact of the changeability influence of the concretely studied MDE setting or effects that stem from the interrelation with the process used.

Provide Guidance for Project Preparation!

Second, there is a need for support during project preparation. Also these implications arise from the observed diversity of MDE settings and the possibility that a given MDE setting affects changeability negatively or that a given MDE setting implies constraints on a process. The main implication is that research is required to provide guidance for project preparation in order to prevent mismatches of MDE settings and processes as well as to support a safe use of risky MDE settings.

Although this thesis provides first techniques for the analysis of hard changeability concerns and the assessment of the manifestation of MDE traits, further research is still required. For example, while the identification of a process-relevant manifestation of an MDE trait helps to identify the need for process tailoring, it is still not clear what tailoring actions provide the best solution. Thus, research is required on best practices and guidelines for MDE-based tailoring of processes.

Further, under financial and time constraints it is often necessary to work with the existing MDE setting. Analyzing an MDE setting can nonetheless be useful in order to agree on conventions about how to use the MDE setting safely, as proposed above. Project preparation might in future be supported by research on whether and how such conventions can be automatically retrieved.

For the companies in future the question arises whether management of MDE settings need to be embedded in the management and tailoring of the software development processes.

Future Work on MDE Effects and Boundaries

Further, the thesis had a focused view, only. On the one hand, a focus was set on an MDE setting’s impact on changeability. As discussed the provided analysis techniques focus mainly on hard changeability concerns, especially on “*unexpected loss of content*”. Future work might lead to analysis techniques for additional changeability concerns. Still, it was not considered how MDE settings can affect other internal software quality attributes, such as reusability. This remains a subject for future work, too.

On the other hand, a focus was set on the interrelation of MDE and software development processes. It was possible to theoretically argue how the studied MDE traits can affect software development processes. While the case studies included some hints how the processes actually were adapted, best practices for process tailoring according to MDE traits is still missing. Further, while three promising

MDE traits were studied in detail, future work might reveal additional relevant MDE traits. However, MDE has boundaries to other techniques, too. For example, the question how MDE and architectural strategies are interrelated and can influence each other should be studied in future work.

Summing up, research on both internal software quality attributes and the interrelation to other techniques is promising and could lead to the identification of further candidates for the explanation of the variations in the degree of success of MDE in practice.

12.3.2. Structural Evolution

This thesis provides the insight that structural evolution of MDE settings is part of the practice. This opens up a discussion on evolution that is capable of changing the characteristics of an MDE setting dramatically. In the following, implications of the existence of structural evolution are discussed.

Predicting Consequences of Evolution Decisions

Research often focuses on designing and introducing innovative and specialized MDE settings or single MDE techniques. These approaches are sometimes evaluated empirically by reporting on success or by experiments (e.g. as in [133]). However, when MDE settings are the result of structural evolution, such empirical evaluations of the initial MDE setting are no longer helpful to predict the quality or risks of the new MDE setting. Thus, techniques to evaluate the quality of an MDE setting need to change. An empirical evaluation is expensive and will mostly not be applicable after an evolution step.

Consequently, well-founded knowledge is required about the causality of an MDE setting's structure and the benefits. Based on such knowledge, techniques to statically analyze and predict benefits of an MDE setting would become possible. Such techniques can then be used by practitioners to balance trade-offs when planning the next evolution step.

The analysis techniques introduced in this thesis are a first step in this direction. However, they focus mainly on the identification of risks and constraints. How benefits can be predicted reliably is still an open question. Apart from such overall analysis techniques, support for evolution decisions might also be provided with respect to the concrete evolution steps. Best practices of evolution steps might be collected and shared.

Shifting Trade-Offs

Trade-off plays an important role during evolution decisions. In combination with fact that different forms of evolution lead to different changes in the characteristics of an MDE setting, this leads to the implication that trade-offs need to be better understood.

It is possible to reach the same goals with different MDE settings and thus also by different evolution steps on MDE settings. Therefore, it is necessary to show when and why practitioners choose specific forms of evolution to reach a given goal. Researchers might provide techniques and tools that shift the trade-off, such that less risky (structural) evolution steps become advantageous. For example, frameworks for combining and extending DSLs, like the one presented in [104] might be a first step in this direction. As summarized in Table 12.1 (explained in detail in Appendix H), literature on support for evolution in context of MDE most often focuses on non-structural changes, only. However, it seems that structural evolution is sometimes required. Therefore, more research on guidance and support for structural evolution is required.

In this context, the question arises about how much effort is required to apply an evolution step to an MDE setting (i.e. its *evolvability*). To estimate the costs that are associated with the evolution step itself, techniques to assess and evaluate the evolvability of MDE settings are required. It was an incidental observation of the studies in this thesis that Software Manufacture Models can also help to reveal risks to the evolvability of an MDE setting (e.g. as in Example 17 in Section 10.1.4).

Evolution Strategy and MDE Maturity

Beyond single evolution steps, there is the question of the extent to which a strategic planning of evolution can be used to reach long term goals. This includes the option and task to plan refactoring of MDE settings. During periods when requirements on the created software are stable and resources are available, such a refactoring can be used to correct negative side effects of former evolution steps. For

12. Conclusion

Table 12.1.: Change types considered in literature on evolution support (details can be found in Appendix H) (= specific changes covered; \circ = approach provides solution with assumptions on the language or implementation to be changed; \bullet = approach with a general coverage of the change type)

Kind of Changes\Approaches	[146]	[121]	[87]	[150]	[154]	[97]	[203]	[141]	[104]	[61]
	[120]	[216]	[12]	[43]	[72]	[126]			[140]	
					[142]	[199]				
					[208]	[93]				
<i>Non-Structural Changes</i>										
<i>C1</i> exchange automated activity	-	\circ			-	-	-	\circ	\circ	\circ
<i>C2</i> exchange language			\circ		-	\circ	\bullet	\circ		\circ
<i>C3</i> exchange tool										
<i>Structural Changes</i>										
<i>C4</i> change number of artifacts							-	-		
<i>C5</i> change number of languages							-	-		
<i>C6</i> change number of manual activities							-	-		
<i>C7</i> change number of tools										
<i>C8</i> change number of automated activities										
<i>C9</i> change order of manual / automated activities										

example, a goal might be to improve the integration of different tools or to decrease complexity for the developers.

In the course of this, research on capability maturity models for MDE might be resumed. As discussed in the related work section, existing proposals for such CMMs are more focused on specific forms of MDE rather than of focusing on characteristics of these MDE settings. However, the insights about MDE characteristics and structural evolution gained in this thesis can be a good starting point, to create a CMM that reflects differences in the properties of MDE settings, such as their changeability or evolvability. Thus, the maturity levels of a CMM for MDE – similar to the levels of the CMM for software development processes [159] – can become actual indicators for the predictability of a project.

Change Management

Finally, the observations on structural evolution not only have implications for research, but also for practice. For example, the observations indicate that sometimes developers trigger and implement evolution steps on MDE settings on their own initiative. Considering the risks and potential that are associated with structural evolution, it might be a meaningful step to establish a management of change requests for MDE settings within a company. Such a change management would allow developers to contribute to improvement of the MDE settings, while the risk of uncoordinated and inefficient evolution of MDE settings can be reduced.

Bibliography

- [1] ISO/IEC TR 9126: Software engineering – Product quality. International Organization for standardization, 2001.
- [2] Software and Systems Process Engineering Metamodel specification (SPEM) Version 2.0. <http://www.omg.org/spec/SPEM/2.0/>, April 2008.
- [3] Systems Modeling Language v. 1.1, November 2008.
- [4] The Process Enactment Tool Framework - Transformation of software process models to prepare enactment. *Science of Computer Programming* 79, 0 (2014), 172 – 188.
- [5] ALMEIDA, J. P. A., PIRES, L., AND SINDEREN, M. Costs and benefits of multiple levels of models in MDA development. In *Second European Workshop on Model Driven Architecture with an Emphasis on Methodologies and Transformations, Canterbury, UK* (Canterbury, August 2004), D. H. Akehurst, Ed., no. TR-17-04, Computing Laboratory, University of Kent, pp. 12–20.
- [6] AMBLER, S. W. *The Object Primer*, 3 ed. Cambridge University Press, 2004.
- [7] APPLETON, B. Patterns and Software: Essential Concepts and Terminology. *Object Magazine Online* 3, 9 (May 1997).
- [8] ARANDA, J., DAMIAN, D., AND BORICI, A. Transition to Model-Driven Engineering. In *Model Driven Engineering Languages and Systems*, R. France, J. Kazmeier, R. Breu, and C. Atkinson, Eds., vol. 7590 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 692–708.
- [9] ASADI, M., AND RAMSIN, R. MDA-based methodologies: an analytical survey. In *Model Driven Architecture—Foundations and Applications* (2008), Springer, pp. 419–431.
- [10] ASCHAUER, T., DAUENHAUER, G., AND PREE, W. A modeling language's evolution driven by tight interaction between academia and industry. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2* (New York, NY, USA, 2010), ICSE '10, ACM, pp. 49–58.
- [11] ATKINSON, C., AND KÜHNE, T. Concepts for comparing modeling tool architectures. In *Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2005), MoDELS'05, Springer-Verlag, pp. 398–413.
- [12] BABAU, J.-P., AND KERBOEUF, M. Domain Specific Language Modeling Facilities. In *Proceedings of the 5th MoDELS workshop on Models and Evolution* (Wellington, Nouvelle-Zélande, October 2011), pp. 1–6.
- [13] BACKUS, J. The syntax and Semantics of the Proposed International Algebraic Language. Information Processing. In *International Conference on Information Processing, Paris, France* (1959).
- [14] BAILLIEZ, STEPHANE AND BAROZZI, NICOLA KEN AND BERGERON, JACQUES AND BODEWIG, STEFAN AND CHANEZON, PATRICK AND DAVIDSON, JAMES DUNCAN AND DIMOCK, TOM AND DONALD, PETER AND HATCHER, ERIK AND HOLT, DIANE. *Apache Ant User Manual*. The Apache Software Foundation, 2003.
- [15] BAKER, P., LOH, S., AND WEIL, F. Model-Driven engineering in a large industrial context – motorola case study. In *Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2005), MoDELS'05, Springer-Verlag, pp. 476–491.
- [16] BALZERT, H. *Lehrbuch der Software-Technik: Software-Entwicklung*. Spektrum, 1996.

Bibliography

- [17] BALZERT, H. *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum, Heidelberg, 1998.
- [18] BARBERO, M., FABRO, M. D., AND BÉZIVIN, J. Traceability and Provenance Issues in Global Model Management. In *ECMDA-TW'07: Proceedings of 3rd Workshop on Traceability, Haifa, Israel* (Haifa, Israel, June 2007), J. Oldevik, G. K. Olsen, and T. Neple, Eds., SINTEF, pp. 47–55.
- [19] BARBERO, M., JOUAULT, F., AND BÉZIVIN, J. Model Driven Management of Complex Systems: Implementing the Macroscope's Vision. In *ECBS '08: Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 277–286.
- [20] BASILI, V. R. The role of experimentation in software engineering: past, current, and future. In *Proceedings of the 18th international conference on Software engineering* (Washington, DC, USA, 1996), ICSE '96, IEEE Computer Society, pp. 442–449.
- [21] BASILI, V. R., AND ROMBACH, H. D. Tailoring the software process to project goals and environments. In *Proceedings of the 9th international conference on Software Engineering* (Los Alamitos, CA, USA, 1987), ICSE '87, IEEE Computer Society Press, pp. 345–357.
- [22] BASILI, V. R., SELBY, R., AND HUTCHENS, D. Experimentation in software engineering. *IEEE Transactions on Software Engineering SE-12*, 7 (1986), 733–743.
- [23] BECK, K., AND ANDRES, C. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.
- [24] BENDRAOU, R., COMBEMALE, B., CREGUT, X., AND GERVAIS, M.-P. Definition of an Executable SPEM 2.0. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference* (Washington, DC, USA, 2007), APSEC '07, IEEE Computer Society, pp. 390–397.
- [25] BENDRAOU, R., GERVAIS, M.-P., AND BLANC, X. UML4SPM: a UML2.0-Based metamodel for software process modelling. In *Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2005), L. Briand and C. Williams, Eds., MoDELS'05, Springer-Verlag, pp. 17–38.
- [26] BENDRAOU, R., JÉZÉQUEL, J.-M., GERVAIS, M.-P., AND BLANC, X. A Comparison of Six UML-Based Languages for Software Process Modeling. *IEEE Transactions on Software Engineering* (2010).
- [27] BÉZIVIN, J., GERARD, S., MULLER, P.-A., AND RIOUX, L. MDA components: Challenges and Opportunities. In *Metamodelling for MDA First International Workshop* (York, UK, November 2003), A. Evans, P. Sammut, and J. S. Willans, Eds., pp. 23 – 41.
- [28] BIEHL, M. *A Modeling Language for the Description and Development of Tool Chains for Embedded Systems*. PhD thesis, KTH Industrial Engineering and Management, 2013.
- [29] BIEHL, M., AND TORNGREN, M. A Cost-Efficiency Model for Tool Chains. In *Global Software Engineering Workshops (ICGSEW), 2012 IEEE Seventh International Conference on* (2012), IEEE, pp. 6–11.
- [30] BIRKHÖLZER, T., MADACHY, R., PFAHL, D., PORT, D., BEITINGER, H., SCHUSTER, M., AND OLKOV, A. SimSWE - a library of reusable components for software process simulation. In *New modeling concepts for today's software processes: Proceedings of the 2010 international conference on software process* (2010), J. Münch, Y. Yang, and W. Schäfer, Eds., vol. 6195 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg.
- [31] BOEHM, B. W. Improving Software Productivity. *Computer 20* (September 1987), 43–57.
- [32] BOEHM, B. W. A Spiral Model of Software Development and Enhancement. *IEEE Computer 21*, 5 (1988), 61–72.

- [33] BÖNNEN, C., AND HERGER, M. *SAP NetWeaver Visual Composer*. SAP PRESS, 2007.
- [34] BORISON, E. A model of software manufacture. In *An international workshop on Advanced programming environments* (London, UK, UK, 1986), Springer-Verlag, pp. 197–220.
- [35] BOWMAN, J., EMERSON, S., AND DARNOVSKY, M. *The Practical SQL Handbook - Using Structured Query Language*. Addison-Wesley, 1994.
- [36] BOX, D., EHNEBUSKE, D., KAKIVAYA, G., LAYMAN, A., MENDELSON, N., NIELSEN, H. F., THATTE, S., AND WINE, D. Simple Object Access Protocol (SOAP) 1.1. Tech. rep., World Wide Web Consortium (W3C), May 2000.
- [37] BROWN, A. W. Model driven architecture: Principles and practice. *Software and Systems Modeling* 3, 4 (December 2004), 314–327.
- [38] BROWN, W. H., MALVEAU, R. C., AND MOWBRAY, T. J. *AntiPatterns: refactoring software, architectures, and projects in crisis*. Wiley, 1998.
- [39] BUCKLEY, J., MENS, T., ZENGER, M., RASHID, A., AND KNIESEL, G. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice* 17, 5 (2005), 309–332.
- [40] CASS, A. G., OSTERWEIL, L., AND WISE, A. A Pattern for Modeling Rework in Software Development Processes. In *International Conference on Software Process, ICSP 2009* (Vancouver, Canada, 16-17 May 2009), Q. Wang, V. Garousi, R. Madachy, and D. Pfahl, Eds.
- [41] CHAUDRON, M. R., HEIJSTEK, W., AND NUGROHO, A. How effective is UML modeling? *Softw. Syst. Model.* 11, 4 (October 2012), 571–580.
- [42] CHRISTENSEN, E., CURBERA, F., MEREDITH, G., AND WEERAWARANA, S. Web Services Description Language (WSDL) 1.1. Tech. rep., World Wide Web Consortium (W3C), 2001.
- [43] CICCHETTI, A., RUSCIO, D. D., ERAМО, R., AND PIERANTONIO, A. Automating Co-evolution in Model-Driven Engineering. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 222–231.
- [44] CLARK, J. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, November 1999.
- [45] CLOCKSIN, W., AND MELLISH, C. *Programming in Prolog*, 4th ed. Springer Verlag, 1994.
- [46] CORBIN, J. M., AND STRAUSS, A. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology* 13 (1990), 3–21.
- [47] CORRÊA, C. K. F., OLIVEIRA, T. C., AND WERNER, C. M. L. An analysis of change operations to achieve consistency in model-driven software product lines. In *Proceedings of the 15th International Software Product Line Conference, Volume 2* (New York, NY, USA, 2011), SPLC ’11, ACM, pp. 24:1–24:4.
- [48] CZARNECKI, K., AND HELSEN, S. Classification of Model Transformation Approaches. In *OOPSLA 2003 Workshop on Generative Techniques in the Context of Model-Driven Architecture, Anaheim, CA, USA* (2003).
- [49] DEMUTH, A., LOPEZ-HERREJON, R. E., AND EGYED, A. Supporting the Co-Evolution of Metamodels and Constraints through Incremental Constraint Management. In *Model Driven Engineering Languages and Systems, 16th International Conference, MODELS 2013* (Miami, USA, 20 September - 4 October 2013), A. Moreira and B. Schaetz, Eds., LNCS, Springer.
- [50] DENG, G., LU, T., TURKAY, E., GOKHALE, A., SCHMIDT, D. C., AND NECHYPURENKO, A. Model Driven Development of Inventory Tracking System. In *Proceedings of the ACM OOPSLA 2003 Workshop on Domain-Specific Modeling Languages* (Anaheim, CA, October 2003).

Bibliography

- [51] DIAW, S., LBATH, R., THÁI, L. V., AND COULETTE, B. SPEM4MDE: a Metamodel for MDE Software Processes Modeling and Enactment. In *Proceedings of the 3rd Workshop on Model-Driven Tool and Process Integration* (2010).
- [52] DIDONET DEL FABRO, M., ALBERT, P., BÉZIVIN, J., AND JOUAULT, F. Industrial-strength Rule Interoperability using Model Driven Engineering. Research Report RR-6747, INRIA, 2008.
- [53] DOMÍNGUEZ-MAYO, F. J., ESCALONA, M. J., MEJÍAS, M., AND TORRES, J. Studying Maintainability on Model-Driven Web Methodologies. In *Information Systems Development*, J. Pokorný, V. Repa, K. Richta, W. Wojtkowski, H. Linger, C. Barry, and M. Lang, Eds. Springer New York, 2011, pp. 195–206.
- [54] DU BOIS, B., DEMEYER, S., VERELST, J., MENS, T., AND TEMMERMAN, M. Does god class decomposition affect comprehensibility? In *IASTED Conf. on Software Engineering* (2006), pp. 346–355.
- [55] EHRIG, H., EHRIG, K., PRANGE, U., AND TAENTZER, G. *Fundamentals of Algebraic Graph Transformation*. Springer, 2006.
- [56] ENDRES, A., AND ROMBACH, D. *A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories*, 1 ed. The Fraunhofer IESE series on software engineering. Addison-Wesley, 2003.
- [57] ENGELS, G., HECKEL, R., KÜSTER, J., AND GROENEWEGEN, L. Consistency-Preserving Model Evolution through Transformations. In *Proceedings of the 5th International Conference on The Unified Modeling Language (UML 2002), Dresden (Germany)* (Berlin/Heidelberg, 2002), J.-M. Jézéquel, H. Hussmann, and S. Cook, Eds., Springer, pp. 212–226.
- [58] ENGELS, G., AND SAUER, S. A meta-method for defining software engineering methods. In *Graph transformations and model-driven engineering*, G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, and B. Westfechtel, Eds. Springer-Verlag, Berlin, Heidelberg, 2010, pp. 411–440.
- [59] ENGELS, G., SAUER, S., AND SOLTENBORN, C. Unternehmensweit verstehen–unternehmensweit entwickeln: Von der Modellierungssprache zur Softwareentwicklungsmethode. *Informatik-Spektrum* 31, 5 (2008), 451–459.
- [60] ESFAHANI, H. C., AND YU, E. A Repository of Agile Method Fragments. In *New Modeling Concepts for Today's Software Processes: Proceedings of the 2010 international conference on software process* (2010), J. Münch, Y. Yang, and W. Schäfer, Eds., vol. 6195 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg.
- [61] ESTUBLIER, J., VEGA, G., AND IONITA, A. D. Composing domain-specific languages for wide-scope software engineering applications. In *Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2005), MoDELS'05, Springer-Verlag, pp. 69–83.
- [62] FAVRE, J.-M. Foundations of Model (Driven) (Reverse) Engineering – Episode I: Story of The Fidus Papirus and the Solarus. In *Post-proceedings of Dagstuhl seminar on Model-Driven Reverse Engineering* (2004).
- [63] FAVRE, J.-M. Languages evolve too! Changing the Software Time Scale. In *Proceedings of the Eighth International Workshop on Principles of Software Evolution* (Washington, DC, USA, 2005), IWPSE '05, IEEE Computer Society, pp. 33–44.
- [64] FAVRE, J.-M. Megamodelling and Etymology. In *Transformation Techniques in Software Engineering* (2005), J. Cordy, R. Lämmel, and A. Winter, Eds., vol. 05161 of *Dagstuhl Seminar Proceedings*, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

-
- [65] FAVRE, J.-M., LÄMMEL, R., AND VARANOVICH, A. Modeling the linguistic architecture of software products. In *Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2012), MODELS'12, Springer-Verlag, pp. 151–167.
- [66] FAVRE, J.-M., AND NGUYEN, T. Towards a Megamodel to Model Software Evolution Through Transformations. *Electr. Notes Theor. Comput. Sci.* 127, 3 (2005), 59–74.
- [67] FELDMAN, S. I. Make - a program for maintaining computer programs. *Software: Practice and Experience* 9, 4 (1979), 255–265.
- [68] FERNANDEZ, A., INSFRAN, E., AND ABRAHÃO, S. Integrating a Usability Model into Model-Driven Web Development Processes. In *Proceedings of the 10th International Conference on Web Information Systems Engineering* (Berlin, Heidelberg, 2009), WISE '09, Springer-Verlag, pp. 497–510.
- [69] FIEBER, F., HUHN, M., AND RUMPE, B. Modellqualität als Indikator für Softwarequalität: eine Taxonomie. *Informatik-Spektrum* 31, 5 (2008), 408–424.
- [70] FIEBER, F., AND PETRASCH, R. Erweiterung des V-Modell XT—Eine Projektdurchführungsstrategie für die modellgetriebene Software-Entwicklung mit der MDA. *GI Softwaretechnik-Trends* 25, 3 (2005).
- [71] FIEBER, F., REGNAT, N., AND RUMPE, B. Assessing usability of model driven development in industrial projects. In *4th European Workshop “From code centric to model centric software engineering: Practices, Implications and ROI (C2M)”(Proceedings)* (2009), Citeseer, pp. 1–9.
- [72] FLEUREY, F., BRETON, E., BAUDRY, B., NICOLAS, A., AND JÉZÉQUEL, J.-M. Model-driven engineering for software migration in a large industrial context, booktitle = Proceedings of the 10th international conference on Model Driven Engineering Languages and Systems. MODELS'07, Springer-Verlag, pp. 482–497.
- [73] FLEUREY, F., AND SOLBERG, A. A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems. In *MODELS '09: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2009), A. Schürr and B. Selic, Eds., vol. 5795 of *LNCS*, Springer-Verlag, pp. 606–621.
- [74] FONDEMENT, F., AND SILAGHI, R. Defining Model Driven Engineering Processes. In *in Proceedings of WISME* (2004).
- [75] FÖRSTER, A., ENGELS, G., AND SCHATTKOWSKY, T. Activity diagram patterns for modeling quality constraints in business processes. In *Model Driven Engineering Languages and Systems*. Springer, 2005, pp. 2–16.
- [76] FÖRSTER, A., ENGELS, G., SCHATTKOWSKY, T., AND VAN DER STRAETEN, R. Verification of business process quality constraints based on visual process patterns. In *Theoretical Aspects of Software Engineering, 2007. TASE'07. First Joint IEEE/IFIP Symposium on* (2007), IEEE, pp. 197–208.
- [77] FORWARD, A., AND LETHBRIDGE, T. C. Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals. In *Proceedings of the 2008 international workshop on Models in software engineering* (New York, NY, USA, 2008), MiSE '08, ACM, pp. 27–32.
- [78] FOWLER, M. *Domain-Specific Languages*. Addison-Wesley, October 2010.
- [79] FRANCE, R., AND RUMPE, B. Model-driven Development of Complex Software: A Research Roadmap. In *FOSE '07: 2007 Future of Software Engineering* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 37–54.
- [80] FRASER, G., WOTAWA, F., AND AMMANN, P. E. Testing with model checkers: a survey. *Softw. Test. Verif. Reliab.* 19 (September 2009), 215–261.

Bibliography

- [81] FRICKE, E., AND SCHULZ, A. P. Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle. *Syst. Eng.* 8 (November 2005), 308.1–308.2.
- [82] FRIEDRICH, J., HAMMERSCHALL, U., KUHRMANN, M., AND SIHLING, M. Das V-Modell XT. In *Das V-Modell® XT, Informatik im Fokus*. Springer Berlin Heidelberg, 2009, pp. 1–32.
- [83] FRITZSCHE, M., AND JOHANNES, J. Models in software engineering. Springer-Verlag, Berlin, Heidelberg, 2008, ch. Putting Performance Engineering into Model-Driven Engineering: Model-Driven Performance Engineering, pp. 164–175.
- [84] FUGGETTA, A. Software process: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering* (New York, NY, USA, 2000), ACM, pp. 25–34.
- [85] GALL, H. C., FLURI, B., AND PINZGER, M. Change Analysis with Evolizer and ChangeDistiller. *IEEE Softw.* 26 (January 2009), 26–33.
- [86] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns - Elements of Reusable Object-Oriented Software*, 34 ed. Ad, 2007.
- [87] GARCÉS, K., JOUAULT, F., COINTE, P., AND BÉZIVIN, J. Managing Model Adaptation by Precise Detection of Metamodel Changes. In *ECMDA-FA '09: Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 34–49.
- [88] GARZAS, J., AND PIATTINI, M. Analyzability and Changeability in Design Patterns. In *The Second Latin American Conference on Pattern Languages of Programming (SugarLoafPLoP)* (2002).
- [89] GIESE, H., AND WAGNER, R. Incremental Model Synchronization with Triple Graph Grammars. In *Proc. of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS), Genova, Italy* (October 2006), O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds., vol. 4199 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, pp. 543–557.
- [90] GOSLING, J., JOY, B., AND STEELE, G. *The Java Language Specification*. Addison-Wesley, Reading, MA, 1996.
- [91] GRADY, R. B. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, Englewood Cliffs, NJ, 1992.
- [92] GRUHN, V., PIEPER, D., AND RÖTTGERS, C. *MDA: Effektives Softwareengineering mit UML2 und Eclipse*, 1 ed. Springer, Berlin, 2006.
- [93] GRUSCHKO, B., KOLOVOS, D., AND PAIGE, R. Towards synchronizing models with evolving metamodels. In *Proceedings of the International Workshop on Model-Driven Software Evolution* (2007).
- [94] GUTTMAN, M. *Real-Life MDA*. Morgan Kaufmann Publishers, 2005.
- [95] HEIJSTEK, W., AND CHAUDRON, M. R. Empirical Investigations of Model Size, Complexity and Effort in a Large Scale, Distributed Model Driven Development Process. In *Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications* (Washington, DC, USA, 2009), SEAA '09, IEEE Computer Society, pp. 113–120.
- [96] HEIJSTEK, W., AND CHAUDRON, M. R. The Impact of Model Driven Development on the Software Architecture Process. *Software Engineering and Advanced Applications, Euromicro Conference* (2010), 333–341.
- [97] HERRMANNSDOERFER, M., RATIU, D., AND WACHSMUTH, G. Language evolution in practice: the history of GMF. In *Proceedings of the Second international conference on Software Language Engineering* (Berlin, Heidelberg, 2010), M. Van den Brand, D. Gasevic, and J. Gray, Eds., SLE'09, Springer-Verlag, pp. 3–22.

-
- [98] HUTCHINSON, J., ROUNCEFIELD, M., AND WHITTLE, J. Model-driven engineering practices in industry. In *Proceeding of the 33rd international conference on Software engineering* (Waikiki, Honolulu, HI, USA, 2011), ICSE '11, ACM, pp. 633–642.
- [99] HUTCHINSON, J., WHITTLE, J., ROUNCEFIELD, M., AND KRISTOFFERSEN, S. Empirical assessment of MDE in industry. In *Proceeding of the 33rd international conference on Software engineering* (New York, NY, USA, 2011), ICSE '11, ACM, pp. 471–480.
- [100] JACOBSON, I., BOOCH, G., AND RUMBAUGH, J. *The Unified Software Development Process*. The Addison-Wesley Object Technology Series. Addison-Wesley, January 1999.
- [101] JACOBSON, I., HUANG, S., KAJKO-MATTSSON, M., MCMAHON, P., AND SEYMOUR, E. Semat-Three Year Vision. *Programming and computer software* 38, 1 (2012), 1–12.
- [102] JÉZÉQUEL, J.-M., COMBEMALE, B., DERRIEN, S., GUY, C., AND RAJOPADHYE, S. Bridging the chasm between MDE and the world of compilation. *Software and Systems Modeling* 11, 4 (October 2012), 581–597.
- [103] JOHANNES, J., AND ASSMANN, U. Concern-based (de)composition of model-driven software development processes. In *Proceedings of the 13th international conference on Model driven engineering languages and systems: Part II* (Berlin, Heidelberg, 2010), D. Petriu, N. Rouquette, and Ø. Haugen, Eds., MODELS'10, Springer-Verlag, pp. 47–62.
- [104] JOHANNES, J., AND FERNÁNDEZ, M. A. Adding abstraction and reuse to a network modelling tool using the reuseware composition framework. In *Proceedings of the 6th European conference on Modelling Foundations and Applications* (Berlin, Heidelberg, 2010), T. Kühne, B. Selic, M.-P. Gervais, and F. Terrier, Eds., ECMFA'10, Springer-Verlag, pp. 132–143.
- [105] JOUAULT, F., ALLILAIRE, F., BÉZIVIN, J., AND KURTEV, I. ATL: A model transformation tool. *Science of Computer Programming* 72, 1-2 (2008), 31–39.
- [106] KALUS, G., AND KUHRMANN, M. Criteria for software process tailoring: a systematic review. In *Proceedings of the 2013 International Conference on Software and System Process* (New York, NY, USA, 2013), ICSSP 2013, ACM, pp. 171–180.
- [107] KAPTEIJNS, T., JANSEN, S., BRINKKEMPER, S., HOUET, H., AND BARENDE, R. A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare. *From code centric to model centric software engineering: Practices, Implications and ROI* (2009), 22.
- [108] KARAILA, M. Evolution of a Domain Specific Language and its engineering environment – Lehman's laws revisited. In *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling* (2009).
- [109] KARANAM, M., AND AKEPOGU, A. Model-Driven Software Evolution: The Multiple Views. In *Proceedings of the International MultiConference of Engineers and Computer Scientists* (2009), vol. 1.
- [110] KELLY, S., AND TOLVANEN, J.-P. *Domain-Specific Modeling: Enabling Full Code Generation*. John Wiley & Sons, 2008.
- [111] KENT, S. Model Driven Engineering. In *Proceedings of the Third International Conference on Integrated Formal Methods (IFM 2002)*, Turku, Finland (May 2002), M. Butler, L. Petre, and K. Sere, Eds., vol. 2335 of *Lecture Notes in Computer Science (LNCS)*, Springer Verlag, pp. 286 – 298.
- [112] KHALIL, A., AND DINGEL, J. Supporting the Evolution of UML Models in Model Driven Software Development: A Survey. Tech. rep., School of Computing, Queen's University Kingston, Ontario, Canada, 2013.

Bibliography

- [113] KINDLER, E., RUBIN, V., AND WAGNER, R. An Adaptable TGG Interpreter for In-Memory Model Transformation. In *Proc. of the 2nd International Fujaba Days 2004, Darmstadt, Germany* (2004), A. Schürr and A. Zündorf, Eds., vol. tr-ri-04-253 of *Technical Report*, University of Paderborn, pp. 35–38.
- [114] KIRSTAN, S., AND ZIMMERMANN, J. Evaluating costs and benefits of model-based development of embedded software systems in the car industry - Results of a qualitative Case Study. In *The Fifth Workshop From code centric to model centric: Evaluating the effectiveness of MDD (C2M:EEMDD)* (2010).
- [115] KLEPPE, A. G., WARMER, J., AND BAST, W. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley, Boston, MA, USA, 2003.
- [116] KÖHLER, H. J., NICKEL, U. A., NIERE, J., AND ZÜNDORF, A. Integrating UML Diagrams for Production Control Systems. In *Proc. of the 22nd International Conference on Software Engineering (ICSE), Limerick, Ireland* (2000), ACM Press, pp. 241–251.
- [117] KOUDRI, A., AND CHAMPEAU, J. MODAL: a SPEM extension to improve co-design process models. In *New Modeling Concepts for Today's Software Processes: Proceedings of the 2010 international conference on software process* (2010), J. Münch, Y. Yang, and W. Schäfer, Eds., vol. 6195 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg.
- [118] KRUCHTEN, P. *Rational Unified Process 3rd Edition: An Introduction*. Object Technology Series. Addison-Wesley Longman, Amsterdam, 2003.
- [119] KUHN, A., MURPHY, G., AND THOMPSON, C. An Exploratory Study of Forces and Frictions Affecting Large-Scale Model-Driven Development. In *Model Driven Engineering Languages and Systems*, R. France, J. Kazmeier, R. Breu, and C. Atkinson, Eds., vol. 7590 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 352–367.
- [120] KULKARNI, V., BARAT, S., AND RAMTEERTHKAR, U. Early experience with agile methodology in a model-driven approach. In *Proceedings of the 14th international conference on Model driven engineering languages and systems* (Berlin, Heidelberg, 2011), J. Whittle, T. Clark, and T. Kühne, Eds., MODELS'11, Springer-Verlag, pp. 578–590.
- [121] KÜSTER, J., GSCHWIND, T., AND ZIMMERMANN, O. Incremental Development of Model Transformation Chains Using Automated Testing. In *Model Driven Engineering Languages and Systems: 12th International Conference, MODELS 2009, Denver, CO, USA*, (2009), vol. 5795/2009 of *Lecture Notes in Computer Science (LNCS)*, Springer Berlin / Heidelberg, pp. 733–747.
- [122] KWIATKOWSKA, M. Z., NORMAN, G., AND PARKER, D. PRISM: Probabilistic Symbolic Model Checker. In *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools* (London, UK, 2002), TOOLS '02, Springer-Verlag, pp. 200–204.
- [123] LAMB, D. A. Relations in Software Manufacture. Tech. Rep. 1990-292, Queen's University School of Computing, 1991.
- [124] LEHMAN, M. M., RAMIL, J. F., WERNICK, P. D., PERRY, D., AND TURSKI, W. M. Metrics and Laws of Software Evolution - The Nineties View. In *METRICS '97: Proceedings of the 4th International Symposium on Software Metrics* (Washington, DC, USA, 1997), IEEE Computer Society, p. 20.
- [125] LERNER, B. S. A model for compound type changes encountered in schema evolution. *ACM Transactions on Database Systems (TODS)* 25, 1 (2000), 83–127.
- [126] LEVENDOVSKY, T., BALASUBRAMANIAN, D., NARAYANAN, A., AND KARSAI, G. A novel approach to semi-automated evolution of DSML model transformation. In *Proceedings of the Second international conference on Software Language Engineering* (Berlin, Heidelberg, 2010), M. Van den Brand, D. Gasevic, and J. Gray, Eds., SLE'09, Springer-Verlag, pp. 23–41.

-
- [127] LEWIS, G., SMITH, D. B., AND KONTOGIANNIS, K. A Research Agenda for Service-Oriented Architecture (SOA): Maintenance and Evolution of Service-Oriented Systems. Tech. Rep. CMU/SEI-2010-TN-003, Carnegie Mellon University, Software Engineering Institute, 2010.
- [128] LINDHOLM, T., YELLIN, F., BRACHA, G., AND BUCKLEY, A. *The JavaTM Virtual Machine Specification, Java SE 7 Edition*. Oracle America, Inc., 500 Oracle Parkway M/S 5op7, California 94065, U.S.A., July 2011.
- [129] MACIEL, R. S. P., DA SILVA, B. C., MAGALHÃES, A. P. F., AND ROSA, N. S. An Integrated Approach for Model Driven Process Modeling and Enactment. In *Proceedings of the 2009 XXIII Brazilian Symposium on Software Engineering* (Washington, DC, USA, 2009), SBES '09, IEEE Computer Society, pp. 104–114.
- [130] MAHÉ, V., COMBEMALE, B., AND CADAVID, J. Crossing Model Driven Engineering and Agility: Preliminary Thought on Benefits and Challenges. In *3rd Workshop on Model-Driven Tool & Process Integration, in conjunction with ECMFA 2010* (Paris, France, June 2010).
- [131] MANSUROV, N., AND CAMPARA, D. UML Modeling Languages and Applications. Springer-Verlag, Berlin, Heidelberg, 2005, ch. Managed architecture of existing code as a practical transition towards MDA, pp. 219–233.
- [132] MARCZYK, G. R., DEMATTEO, D., AND FESTINGER, D. *Essentials of research design and methodology*, vol. 2. Wiley, 2010.
- [133] MARTÍNEZ, Y., CACHERO, C., MATERA, M., ABRAHAO, S., AND LUJÁN, S. Impact of MDE approaches on the maintainability of web applications: an experimental evaluation. In *Proceedings of the 30th international conference on Conceptual modeling* (Berlin, Heidelberg, 2011), M. Jeusfeld, L. Delcambre, and T.-W. Ling, Eds., ER'11, Springer-Verlag, pp. 233–246.
- [134] MAXWELL, K. D., VAN WASSENHOVE, L., AND DUTTA, S. Software Development Productivity of European Space, Military, and Industrial Applications. *IEEE Trans. Softw. Eng.* 22 (October 1996), 706–718.
- [135] MELLEGÅRD, N., AND STARON, M. Characterizing model usage in embedded software engineering: a case study. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume* (New York, NY, USA, 2010), ECSA '10, ACM, pp. 245–252.
- [136] MELLEGÅRD, N., AND STARON, M. Improving efficiency of change impact assessment using graphical requirement specifications: an experiment. In *Proceedings of the 11th international conference on Product-Focused Software Process Improvement* (Berlin, Heidelberg, 2010), PROFES'10, Springer-Verlag, pp. 336–350.
- [137] MENS, T., BLANC, X., AND MENS, K. Model-Driven Software Evolution: An alternative Research Agenda. In *The 6th BELgian-NETHERlands software eVOLUTION workshop (BENEVOL 2007)* (2007).
- [138] MENS, T., CZARNECKI, K., AND GORP, P. V. 04101 Discussion – A Taxonomy of Model Transformations. In *Language Engineering for Model-Driven Software Development* (Dagstuhl, Germany, 2005), J. Bézivin and R. Heckel, Eds., no. 04101 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
- [139] MENS, T., AND DEMEYER, S. *Software Evolution*. Springer, 2008.
- [140] MEYERS, B., MANNADIAR, R., AND VANGHELUWE, H. Evolution of Modelling Languages. In *8th BELgian-NETHERlands software eVOLUTION seminar (BENEVOL)* (2009).
- [141] MEYERS, B., AND VANGHELUWE, H. A framework for evolution of modelling languages. *Science of Computer Programming* 76, 12 (2011), 1223 – 1246.
- [142] MEYERS, B., WIMMER, M., CICCHETTI, A., AND SPRINKLE, J. A generic in-place transformation-based approach to structured model co-evolution. In *Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM'10) @ MoDELs'10* (2010), Electronic Communications of the EASST.

Bibliography

- [143] MOHAGHEGHI, P., AND DEHLEN, V. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In *Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications* (Berlin, Heidelberg, 2008), ECMDA-FA '08, Springer-Verlag, pp. 432–443.
- [144] MOHAGHEGHI, P., DEHLEN, V., AND NEPLE, T. Definitions and Approaches to Model Quality in Model-Based Software Development - A Review of Literature. *Journal of Information and Software Technology - Quality of UML Models* 51, 12 (December 2009), 1646–1669.
- [145] MOHAGHEGHI, P., FERNANDEZ, M., MARTELL, J. A., FRITZSCHE, M., AND GILANI, W. Models in Software Engineering. Springer-Verlag, Berlin, Heidelberg, 2009, ch. MDE Adoption in Industry: Challenges and Success Criteria, pp. 54–59.
- [146] MOHAGHEGHI, P., GILANI, W., STEFANESCU, A., FERNANDEZ, M., NORDMOEN, B., AND FRITZSCHE, M. Where does model-driven engineering help? Experiences from three industrial cases. *Software and Systems Modeling* (2011), 1–21.
- [147] MONPERRUS, M., JÉZÉQUEL, J.-M., CHAMPEAU, J., AND HOELTZENER, B. A Model-Driven Measurement Approach. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2008), K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds., MoDELS '08, Springer-Verlag, pp. 505–519.
- [148] MÜNCH, J., ARMBRUST, O., KOWALCZYK, M., AND SOTO, M. *Software Process Definition and Management*. Springer Publishing Company, Incorporated, 2012.
- [149] MURATA, M., LEE, D., MANI, M., AND KAWAGUCHI, K. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Technol.* 5, 4 (November 2005), 660–704.
- [150] NARAYANAN, A., LEVENDOVSZKY, T., BALASUBRAMANIAN, D., AND KARSAI, G. Automatic Domain Model Migration to Manage Metamodel Evolution. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2009), MODELS '09, Springer-Verlag, pp. 706–711.
- [151] NOY, N. F., AND KLEIN, M. Ontology evolution: Not the same as schema evolution. *Knowledge and information systems* 6, 4 (2004), 428–440.
- [152] NUGROHO, A., AND CHAUDRON, M. R. Evaluating the Impact of UML Modeling on Software Quality: An Industrial Case Study. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2009), A. Schürr and B. Selic, Eds., MODELS '09, Springer-Verlag, pp. 181–195.
- [153] OBJECT MANAGEMENT GROUP. *Meta Object Facility (MOF) Specification*, September 1997.
- [154] OBJECT MANAGEMENT GROUP. *MDA Guide Version 1.0.1*, June 2003.
- [155] OBJECT MANAGEMENT GROUP. *MOF 2.0 QVT 1.0 Specification*, 2008.
- [156] OMG. UML 2.0 Superstructure Specification, Object Management Group, Version 2.0, formal/05-07-04, 2005.
- [157] OMG. Business Process Model and Notation (BPMN), 2011.
- [158] OMG, ARCHITECTURE-DRIVEN MODERNIZATION TASK FORCE. *Architecture-Driven Modernization Scenarios*, January 2006.
- [159] PAULK, M. C., CURTIS, B., CHRISSIS, M. B., AND WEBER, C. V. Capability Maturity Model, Version 1.1. *IEEE Softw.* 10, 4 (July 1993), 18–27.
- [160] PEROVICH, D., BASTARRICA, M., AND ROJAS, C. Model-Driven approach to Software Architecture design. In *SHARK '09: Proceedings of the 2009 ICSE Workshop on Sharing and Reusing Architectural Knowledge* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 1–8.

-
- [161] PERRY, D. E., PORTER, A. A., AND VOTTA, L. G. Empirical studies of software engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (New York, NY, USA, 2000), ICSE '00, ACM, pp. 345–355.
- [162] PFEIFFER, R.-H., AND WASOWSKI, A. Cross-language support mechanisms significantly aid software development. In *Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2012), MODELS'12, Springer-Verlag, pp. 168–184.
- [163] POHL, K., BÖCKL, G., AND VAN DER LINDEN, F. *Software Product Line Engineering. Foundations, Principles, and Techniques*. Springer, Berlin Heidelberg New York, 2005.
- [164] PORRES, I., AND VALIENTE, M. C. Process Definition and Project Tracking in Model Driven Engineering. In *Product-Focused Software Process Improvement* (2006), vol. 4034 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 127–141.
- [165] PYTHON SOFTWARE FOUNDATION. *The Python Language Reference, Release 2.7.4*, van rossum, guido and drake, fred l. ed., April 2013.
- [166] RAUSCH, A., BARTELT, C., TERNITÉ, T., AND KUHRMANN, M. The V-Modell XT Applied - Model-Driven and Document-Centric Development. In *Proceedings 3rd World Congress for Software Quality* (September 2005), vol. 3.
- [167] RICH, C., AND WATERS, R. C. Automatic Programming: Myths and Prospects. *Computer* 21, 8 (August 1988), 40–51.
- [168] RIOS, E., BOZHEVA, T., BEDIAGA, A., AND GUILLOREAU, N. MDD maturity model: a roadmap for introducing model-driven development. In *Proceedings of the Second European conference on Model Driven Architecture: foundations and Applications* (Berlin, Heidelberg, 2006), ECMDA-FA'06, Springer-Verlag, pp. 78–89.
- [169] RISING, L. *The Pattern Almanac 2000*. Addison-Wesley, 2000.
- [170] RODDICK, J. F. Schema Evolution in Database Systems - An Annotated Bibliography. *SIGMOD record* 21, 4 (1992), 35–40.
- [171] ROUSSEV, B., AND WU, J. Transforming use case models to class models and OCL-specifications. *Int. J. Comput. Appl.* 29, 1 (January 2007), 59–69.
- [172] SADOVYKH, A., VIGIER, L., GOMEZ, E., HOFFMANN, A., GROSSMANN, J., AND ESTEKHIN, O. On Study Results: Round Trip Engineering of Space Systems. In *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications* (2009), ECMDA-FA '09, Springer-Verlag, pp. 265–276.
- [173] SALAY, R. Towards a Formal Framework for Multimodeling in Software Engineering. In *Proceedings of the Doctoral Symposium at the ACM/IEEE 10th International Conference on Model-Driven Engineering Languages and Systems*, (Nashville (TN), USA, October 2007), C. Pons, Ed., vol. Vol-26.
- [174] SALAY, R., MYLOPOULOS, J., AND EASTERBROOK, S. Using Macromodels to Manage Collections of Related Models. In *21st International Conference, CAiSE 2009, Amsterdam, The Netherlands* (8-12 June 2009), vol. 5565/2009 of *Lecture Notes in Computer Science (LNCS)*, Springer-Verlag, pp. 141–155.
- [175] SCHÄTZ, B., AND SPIES, K. Model-based software engineering: Linking more and less formal product-models to a structured process. In *Proceedings of International Conference Software and Systems Engineering and their Applications (ICSSEA 2002)* (December 2002).
- [176] SCHWABER, K., AND SUTHERLAND, J. *The Scrum Guide*, 1991.
- [177] SEAMAN, C. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 25, 4 (1999), 557–572.

Bibliography

- [178] SEIBEL, A., NEUMANN, S., AND GIESE, H. Dynamic hierarchical mega models: comprehensive traceability and its efficient maintenance. *Software and Systems Modeling* 9, 4 (2010), 493–528.
- [179] SELIC, B. The Pragmatics of Model-Driven Development. *IEEE Software* 20, 5 (September 2003), 19–25.
- [180] SELIC, B. What will it take? A view on adoption of model-based methods in practice. *Software and Systems Modeling* 11, 4 (October 2012), 513–526.
- [181] SHAW, M. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer (STTT)* 4, 1 (2002), 1–7.
- [182] SHIRTZ, D., KAZAKOV, M., AND SHAHAM-GAFNI, Y. Adopting model driven development in a large financial organization. In *Proceedings of the 3rd European conference on Model driven architecture-foundations and applications* (Berlin, Heidelberg, 2007), ECMDA-FA'07, Springer-Verlag, pp. 172–183.
- [183] SIMIDCHIEVA, B. I., OSTERWEIL, L., AND WISE, A. Structural Considerations in Defining Executable Process Models. In *International Conference on Software Process, ICSP 2009* (Vancouver, Canada, 16-17 May 2009), Q. Wang, V. Garousi, R. Madachy, and D. Pfahl, Eds.
- [184] SINDICO, A., NATALE, M., AND SANGIOVANNI-VINCENTELLI, A. An Industrial System Engineering Process Integrating Model Driven Architecture and Model Based Design. In *Model Driven Engineering Languages and Systems*, R. France, J. Kazmeier, R. Breu, and C. Atkinson, Eds., vol. 7590 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 810–826.
- [185] SNEED, H. M. *Softwaremanagement*. Verlagsgesellschaft Rudolf Müller, 1987.
- [186] SOMMERVILLE, I. Software process models. *ACM Comput. Surv.* 28, 1 (March 1996), 269–271.
- [187] SOMMERVILLE, I. *Software Engineering*, 8 ed. Addison Wesley, 2007.
- [188] STAHL, T., VÖLTER, M., EFFTINGE, S., AND HAASE, A. *Modellgetriebene Softwareentwicklung - Techniken, Engineering, Management*, 2 ed. dpunkt Verlag, 2007.
- [189] STAMMEL, J., DURDIK, Z., KROGMANN, K., WEISS, R., AND KOZIOLEK, H. Software Evolution for Industrial Automation Systems: Literature Overview. Tech. rep., KIT, Fakultät für Informatik, 2011.
- [190] STARON, M. Adopting model driven software development in industry: a case study at two companies. In *Proceedings of the 9th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2006), O. Nierstrasz, J. Whittle, D. Harel, and G. Reggio, Eds., MoDELS'06, Springer-Verlag, pp. 57–72.
- [191] STEVENS, W. P., MYERS, G. J., AND CONSTANTINE, L. Structured design. *IBM Syst. J.* 13, 2 (1974), 115–139.
- [192] STRAETEN, R., MENS, T., AND BAELEN, S. Models in Software Engineering. Springer-Verlag, Berlin, Heidelberg, 2009, ch. Challenges in Model-Driven Software Engineering, pp. 35–47.
- [193] SUNKLE, S., AND KULKARNI, V. Cost estimation for model-driven engineering. In *Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2012), MODELS'12, Springer-Verlag, pp. 659–675.
- [194] TARR, P., OSSHER, H., HARRISON, W., AND SUTTON, S. M. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 1999 international conference on Software engineering May 16 - 22, 1999, Los Angeles, CA USA* (1999), pp. 107–119.
- [195] TOMASSETTI, F., TORCHIANO, M., TISO, A., RICCA, F., AND REGGIO, G. Maturity of software modelling and model driven engineering: A survey in the Italian industry. In *Evaluation Assessment in Software Engineering (EASE 2012), 16th International Conference on* (2012), pp. 91–100.

-
- [196] TORCHIANO, M., TOMASSETTI, F., RICCA, F., TISO, A., AND REGGIO, G. Relevance, benefits, and problems of software modelling and model driven techniques – A survey in the Italian industry. *Journal of Systems and Software* 86, 8 (2013), 2110 – 2126.
- [197] TROMPETER, J., PIETREK, G., BELTRAN, J. C. F., HOLZER, B., KAMANN, T., KLOSS, M., MORK, S., NIEHUES, B., AND THOMS, K. *Modellgetriebene Softwareentwicklung - MDA und MDSD in der Praxis*. entwickler.press, 2007.
- [198] TRUJILLO, S., BATÓRY, D., AND DIAZ, O. Feature Oriented Model Driven Development: A Case Study for Portlets. In *Proceedings of the 29th international conference on Software Engineering* (Washington, DC, USA, 2007), ICSE '07, IEEE Computer Society, pp. 44–53.
- [199] VAN DEN BRAND, M., PROTIĆ, Z., AND VERHOEFF, T. A generic solution for syntax-driven model co-evolution. In *Proceedings of the 49th international conference on Objects, models, components, patterns* (Berlin, Heidelberg, 2011), J. Bishop and A. Vallecillo, Eds., TOOLS'11, Springer-Verlag, pp. 36–51.
- [200] VAN DEURSEN, A., KLINT, P., AND VISSER, J. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.* 35, 6 (June 2000), 26–36.
- [201] VAN DEURSEN, A., VISSER, E., AND WARMER, J. Model-Driven Software Evolution: A Research Agenda. In *Proceedings 1st International Workshop on Model-Driven Software Evolution (MoDSE)* (2007), D. Tamzalit, Ed., University of Nantes, pp. 41–49.
- [202] VANHOOFF, B., AYED, D., VAN BAELEN, S., JOOSEN, W., AND BERBERS, Y. UniTI: A Unified Transformation Infrastructure. In *Model Driven Engineering Languages and Systems: 10th International Conference, MODELS 2007, Nashville, USA* (2007), Lecture Notes in Computer Science (LNCS), Springer Berlin / Heidelberg, pp. 31–45.
- [203] VERMOLEN, S., AND VISSER, E. Heterogeneous Coupled Evolution of Software Languages. In *Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2008), K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, Eds., MoDELS '08, Springer-Verlag, pp. 630–644.
- [204] VIGNAGA, A., JOUAULT, F., BASTARRICA, M. C., AND BRUNELIÈRE, H. Typing in Model Management. In *ICMT '09: Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 197–212.
- [205] VOGEL, R. Practical case study of MDD infusion in a SME: Final Results. In *Models and Evolution Joint MODELS'09 Workshop on Model-Driven Software Evolution (MoDSE) and Model Co-Evolution and Consistency Management (MCCM)* (2009), D. Tamzalit, D. Deridder, and B. Schätz, Eds., pp. 68–78.
- [206] VÖLTER, M., AND STAHL, T. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.
- [207] W3C. *Cascading Style Sheets*, Bert Bos and Håkon Wium Lie and Chris Lilley and Ian Jacobs ed., May 1998.
- [208] WACHSMUTH, G. Metamodel adaptation and model co-adaptation. In *ECOOP 2007—Object-Oriented Programming* (2007), Springer, pp. 600–624.
- [209] WARMER, J., AND KLEPPE, A. Getting Started with Modeling Maturity Levels. <http://www.devx.com/enterprise/Article/26664>, December 2004.
- [210] WEIGERT, T., AND WEIL, F. Practical experiences in using model-driven engineering to develop trustworthy computing systems. In *Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006. IEEE International Conference on* (2006), vol. 1, pp. 8 pp.–.

Bibliography

- [211] WEIGERT, T., WEIL, F., MARTH, K., BAKER, P., JERVIS, C., DIETZ, P., GUI, Y., VAN DEN BERG, A., FLEER, K., NELSON, D., WELLS, M., AND MASTENBROOK, B. Experiences in deploying model-driven engineering. In *Proceedings of the 13th international SDL Forum conference on Design for dependable systems* (Berlin, Heidelberg, 2007), SDL'07, Springer-Verlag, pp. 35–53.
- [212] WENZ, C. *JavaScript und AJAX-Das umfassende Handbuch*. Galileo Press, 2007.
- [213] WHITTLE, J., HUTCHINSON, J., ROUNCEFIELD, M., BURDEN, H., AND HELDAL, R. Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In *Model-Driven Engineering Languages and Systems*. Springer, 2013, pp. 1–17.
- [214] WOHLIN, C., RUNESON, P., HöST, M., OHLSSON, M. C., REGNELL, B., AND WESSLÉN, A. *Experimentation in Software Engineering : An Introduction*. Kluwer Academic Publishers, November 1999.
- [215] WORLD WIDE WEB CONSORTIUM (W3C), XML WORKING GROUP. *Extensible Markup Language (XML) 1.0*, 1998.
- [216] YIE, A., CASALLAS, R., WAGELAAR, D., AND DERIDDER, D. An Approach for Evolving Transformation Chains. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems* (Berlin, Heidelberg, 2009), MODELS '09, Springer-Verlag, pp. 551–555.
- [217] YIN, R. K. *Case study research: Design and methods*, vol. 5. Sage, 2009.
- [218] YOURDON, E. N., AND CONSTANTINE, L. *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*. Prentice Hall, 1979.
- [219] ZELKOWITZ, M. V., AND WALLACE, D. Experimental validation in software engineering. *Information and Software Technology* 39, 11 (1997), 735 – 743.
- [220] ZEND TECHNOLOGIES LTD. *Taking the Pulse of the Developer Community*, 2012.
- [221] ZHANG, Y. Test-Driven Modeling for Model-Driven Development. *IEEE Software* 21, 5 (September 2004), 80–86.
- [222] ZHANG, Y., AND PATEL, S. Agile Model-Driven Development in Practice. *IEEE Software* 28, 2 (2011), 84–91.
- [223] ZIMMERMANN, J., STARK, C., AND RIECK, J. *Projektplanung: Modelle, Methoden, Management*. Springer DE, 2005.

Selected Publications

- [P1] HEBIG, R. An Approach to Integrating Model Management and Software Development Processes. In *Doctoral Symposium at MODELS 2011* (2011).
- [P2] HEBIG, R., GABRYSIAK, G., AND GIESE, H. Towards Patterns for MDE-Related Processes to Detect and Handle Changeability Risks. In *2012 International Conference on Software and Systems Process* (2012).
- [P3] HEBIG, R., AND GIESE, H. MDE Settings in SAP. A Descriptive Field Study. Tech. Rep. 58, Hasso-Plattner Institut at University of Potsdam, 2012.
- [P4] HEBIG, R., GIESE, H., STALLMANN, F., AND SEIBEL, A. On the Complex Nature of MDE Evolution. In *Model Driven Engineering Languages and Systems, 16th International Conference, MODELS 2013* (Miami, USA, 20 September - 4 October 2013), A. Moreira and B. Schaetz, Eds., LNCS, Springer.
- [P5] HEBIG, R., SEIBEL, A., AND GIESE, H. On the Unification of Megamodels. In *Proceedings of the 4th International Workshop on Multi-Paradigm Modeling (MPM 2010)* (2011), V. Amaral, H. Vangheluwe, C. Hardebolle, L. Lengyel, T. Magaria, J. Padberg, and G. Taentzer, Eds., vol. 42 of *Electronic Communications of the EASST*.
- [P6] HEBIG, R., SEIBEL, A., AND GIESE, H. Toward a Comparable Characterization for Software Development Activities in Context of MDE. In *Proceedings of the 2011 International Conference on Software and Systems Process* (New York, NY, USA, 2011), ICSSP '11, ACM, pp. 33–42.

Additional Publications

- [A1] BEYHL, T., HEBIG, R., AND GIESE, H. A Model Management Framework for Maintaining Traceability Links. In *Software Engineering 2013 Workshopband* (Aachen, February 2013), S. Wagner and H. Licher, Eds., vol. P-215 of *Lecture Notes in Informatics (LNI)*, Gesellschaft für Informatik (GI), pp. 453–457.
- [A2] BINKLEY, D., LAWRIE, D., HILL, E., BURGE, J., HARRIS, I., HEBIG, R., KESZÖCZE, O., REED, K., AND SLANKAS, J. Task Driven Software Summarization. In *ERA Track at 29th IEEE International Conference on Software Maintenance (ICSM)* (Eindhoven, The Netherlands, 2013).
- [A3] GABRYSIAK, G., EICHLER, D., HEBIG, R., AND GIESE, H. Enabling Domain Experts to Modify Formal Models via a Natural Language Representation Consistently. In *Proc. of the First ICSE 2013 Workshop on Natural Language Analysis in Software Engineering* (25 May 2013), NatuRaLiSE'13.
- [A4] GABRYSIAK, G., GUENTERT, M., HEBIG, R., AND GIESE, H. Teaching Requirements Engineering with Authentic Stakeholders: Towards a Scalable Course Setting. In *Proc. of ICSE 2012 Workshop on Software Engineering Education based on Real-World Experiences* (Zurich, Switzerland, June 2012).
- [A5] GABRYSIAK, G., HEBIG, R., AND GIESE, H. Decoupled Model-Based Elicitation of Stakeholder Scenarios. In *Proc. of the Seventh International Conference on Software Engineering Advances* (Lisbon, Portugal, November 2012), IARIA.
- [A6] GABRYSIAK, G., HEBIG, R., AND GIESE, H. Simulation-Assisted Elicitation and Validation of Behavioral Specifications for Multiple Stakeholders. In *21st IEEE International WETICE conference (WETICE-2012)* (Toulouse, France, June 2012), S. Reddy and K. Drira, Eds., IEEE, pp. 220–225.
- [A7] GABRYSIAK, G., PIRL, L., HEBIG, R., AND GIESE, H. Cooperating with a Non-governmental Organization to Teach Gathering and Implementation of Requirements. In *Proc. of the 26th Conference on Software Engineering Education and Training* (May 2013), CSEE&T'13.
- [A8] HEBIG, R., GIESE, H., AND BECKER, B. Making Control Loops Explicit When Architecting Self-Adaptive Systems. In *SOAR '10: Proceedings of the second international workshop on Self-Organizing Architectures* (Washington, DC, USA, June 2010), ACM, pp. 21–28.
- [A9] HEBIG, R., MEINEL, C., MENZEL, M., THOMAS, I., AND WARSCHOFSKY, R. A Web Service Architecture for Decentralised Identity- and Attribute-Based Access Control. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services* (Los Angeles, CA, USA, 2009), IEEE Computer Society, pp. 551–558.
- [A10] SEIBEL, A., HEBIG, R., AND GIESE, H. *Software and Systems Traceability*. Springer London, 2012, ch. Traceability in Model-Driven Engineering: Efficient and Scalable Traceability Maintenance, pp. 215–240.
- [A11] SEIBEL, A., HEBIG, R., NEUMANN, S., AND GIESE, H. A Dedicated Language for Context Composition and Execution of True Black-Box Model Transformations. In *4th International Conference on Software Language Engineering (SLE 2011), Braga, Portugal* (July 2011).
- [A12] STEUDEL, H., HEBIG, R., AND GIESE, H. A Build Server for Model-Driven Engineering. In *6th International Workshop on Multi-Paradigm Modeling (MPM 2012)* (2012), ACM.

Part V.

Appendix

Appendix A.

Languages used in Captured Case Studies

All in all 56 different languages were identified for the 11 captured MDE settings (as summarized in Table A.1).

The captured MDE settings include five general purpose languages or programming languages: ABAP, Excel (which was in one case rated as GPL and in one case not), Java, Java Script, and PHP.

Further, 21 domain specific languages (DSLs) can be identified: HTML, status and action models (S&A Models), UI Component Definition Language, MDRS language for meta data, Business Object Definition Language (BODL), TYPO-Script, MySQL, CSS, Adobe Photoshop PSD, Adobe Illustrator Artwork (AI), Powerpoint Documents (ppt), XSD, Latex tex-files, Apache string templates, emf-files, Diff-files, SQL, Export configuration, DDL script, DB script, and Korn shell script.

In addition, the captured MDE settings include 14 modeling languages: BPMN, Generic Modeling Language (GML), data-flow diagrams, UML Plus, ESR specific Business Object Model, EA-UML, EA-UML (AUTOSAR Profile), Ecore UML, AUTOSAR XML, a further EA-UML Profile, UML Profile and extension, internal meta model, internal document meta model, and internal format.

Finally, 15 further artifact types can be identified: WSDL artifacts (web service description language), context variables with DDIC-Reference (Data-Dictionary), Info Objects (defined in ABAP Dictionary), rulesets (Regelsprache und Modellierungssprache), ZIP-files, natural language text, URL, Word templates, Word documents, PDF, Bugzilla Tasks, e-mail, PNG, XML, and RTF.

Table A.1.: Used languages in the case studies

Object of Study	GPL / Programming language	DSL	Modeling language	No category assigned
BO	• ABAP • Excel files	• S&A	• UML Plus • ESR specific meta model for business objects models	
SIW	• ABAP			• Context variables with DDIC-Reference • WSDL
VC			• GML • BPMN	• Excel files
Oberon		• UI Component Definition Language • MDRS • BODL		
BRF	• ABAP		• BPMN	• Regelsprache und Modellierungssprache
BW	• ABAP	• HTML	• data-flow diagrams	• Info Objects (defined in ABAP Dictionary)
VCat	• PHP • JavaScript	• TYPO-Script • MySQL • HTML • CSS • Adobe Photoshop PSD • Adobe Illustrator Artwork (AI)		• ZIP-Files • Natural language text • URL
CarmeQ		• Powerpoint Document (ppt) • HTML • XSD • Tex-File (Latex) • Apache String Template • EMF-File • Diff-File	• EA-UML • EA-UML (AUTOSAR Profile) • Ecore UML Profile • Eclipse UML • AUTOSAR XML (arxml)	• Word Template • Word Document • PDF • Bugzilla Tasks • e-mail • PNG
Cap1	• Java		• EA-UML (a Profile)	• XML • RTF
Cap2a		• Export Configuration • HTML	• UML Profile and extension • Internal Meta Model • Internal Document Meta Model	• e-mail • Word document (docx)
Cap2b		• DDL Script • DB Script • Korn Shell Script • HTML • CSS	• UML Profile and extension • Internal Format	• Word document (docx)

Appendix B.

Case Studies on Evolution of MDE Settings

This appendix includes data and references to reports about evolution histories of the case studies collected in the third study (as described in Section 4.3). Table B.1 summarizes the change types that are documented for the different captured evolution steps.

B.1. BO Evolution History

This appendix section includes the description of the evolution history of BO and stems from [P4].

SAP Business ByDesign¹ is a hosted ERP solution for small and medium enterprises. It was built on top of a newly designed platform that has introduced numerous new architecture and modeling concepts into the development process. We focus on a very specific aspect, namely the design and implementation of *business objects*, the main building blocks of the system. Business ByDesign is primarily built using a proprietary programming language and runtime (ABAP), with a tool chain that is necessarily also largely proprietary. The ABAP infrastructure has been a major success factor for SAP, as it enables customers to modify and extend SAP's software. The origin of object modeling at SAP lies in the Data Modeler, a graphical design tool for creating entity-relationship diagrams using the SAP SERM notation. It has no generation capabilities, but as it is part of the ABAP infrastructure, it is possible to navigate directly from an entity or relationship to the implementing table (provided that the link has been manually maintained in the model).

A new architecture. Business ByDesign is based on a modular service-oriented architecture (SOA). In this context, the Data Modeler was used as a conceptual modeling tool for designing the structural aspects of business objects. An important design goal was to provide a set of consistently designed services with harmonized signatures. The chosen solution was to make the object models available in a service repository, which was done by manually reentering them in a different tool. From there, skeletons for the business objects were generated and subsequently fleshed out manually.

S1: Code generation (C4 – C9). To improve development efficiency, architects in different teams began to develop frameworks that automated and standardized the generic parts of the object implementation. This typically covered the generation of elementary services and table structures during development, but also extended to runtime libraries. The generation process used the generated skeletons as input, but required specification of additional model parameters. To increase homogeneity in the platform, one of the frameworks was ultimately selected over the others as the mandated standard (C5, C7).

S2: Behavioral modeling (C4 – C8). Business ByDesign introduced a concept called Status and Action Management (SAM) for constraining when and in which sequence basic services can be invoked on an object. The constraints were evaluated at runtime by a dedicated engine and models were created in Maestro, a proprietary, standalone (non-ABAP) tool providing a graphical editor with simulation capabilities. The goal was to make object behavior more transparent and to ensure correctness of the implementation by eliminating the need to manually write checks for preconditions.

S3: Model integration (C2, C5 – C9). Conceptual SOA modeling was done in ARIS, a commercial business modeling tool, using a custom visual DSL. As many of these models contained references to business objects, it seemed advantageous to consolidate the conceptual models and move the detailed design of object structure and data types into ARIS, eliminating the potential for inconsistencies. While this move further severed the link between model and implementation, it enabled additional validation activities in ARIS.

S4: Model quality (C6, C7, C8). To meet external quality standards, it became necessary to demonstrably *prove* that models in ARIS and the service repository were consistent. Therefore, an

¹SAP Business ByDesign <http://www.sap.com/solutions/technology/cloud/business-by-design/highlights/index.epx>

infrastructure for replicating models from ARIS into the system hosting the service repository was created. This allowed cross-checking manually created content against the replicated conceptual models. The introduction of the checks revealed, however, how strongly conceptual modeling and implementation frameworks had evolved in different directions.

S5: A new infrastructure (*C4 – C9*). Several releases later, development efficiency and total cost of ownership became a major focus while the importance of conceptual modeling declined. In a bold move, ARIS, Maestro, and the service repository were eliminated and replaced by a new metadata repository that was built using the business object runtime infrastructure itself. The new repository was closer to the implementation and provided a single source of truth by consolidating multiple tools and databases. However, this also came at a cost. There initially were no graphical modeling capabilities, the ability to simulate status models was lost, and the modeling of design alternatives or future evolutions of existing objects was not supported.

S6: A simpler alternative (*C1 – C9*). In parallel with the new repository, a new Visual Studio-based tool targeting third-party developers was developed, allowing them to define and program business objects using a script language. As its focus is simplicity, the supported feature set is reduced. In return, the editor acts as a facade that completely hides the underlying tools from the user, thus allowing for very efficient development – within the set limits.

S7: Optimization (*C4 – C8*). Finally, the latest release has brought a redesign of the underlying frameworks. The motivation for this was primarily runtime efficiency, as the existing set of independently developed frameworks proved to generate a significant overhead in their interactions. It was therefore decided to merge features such as Status and Action Management directly into the business object runtime, which was in turn made an integral part of the basic service infrastructure. For the first time, all modeling activities for business objects were gathered in a single tool.

The history of business object development provides multiple examples for structural changes to the development process. The case study illustrates how the weight given to different productivity dimension changes. While initially automation was the key driver of the change (*S1*), the reduction of cost of ownership became more important later on (*S5*). Finally, this case study shows that a sequence of structural evolution steps with changing priorities can transform even a code-centric development approach into a complex MDE setting.

B.2. Ableton Evolution History

The major product of Ableton is a software called Live, which provides artists and musicians with an environment for musical compositions and productions. An important part of the business of Ableton is the development of libraries, which provide users with a collection of presets for instruments included in Live. Technically, the modeled presets are XML documents that conform to a schema definition. This DSL allows non-software engineers (i.e. musicians) to create these libraries. Building a library is a complex process of gathering information, validation of presets, packing libraries into installable binaries and deployment. These steps are automated as an incremental build process that is triggered regularly by cronjobs on a build server.

Ableton continually has to deal with evolution of this process. The evolution can be classified into different categories, which have different reasons. Constant extensions and improvements of the capability of the software imply evolution of schema definitions. This language evolution requires migration of legacy models as well as depending technologies (conforming to the change types *C1* and *C2* in Table B.1). Certain parts of the automation need to be migrated manually (e.g. automatically applied fixes to presets have to be adapted).

Not only the schema definitions evolve, but also the process of building libraries. One reason is the need for refactoring to increase understandability, maintainability, flexibility, etc. Currently, the build process is re-factored by spinning off certain aspects of the build process, such as validation, into separate processes. Thus, the validation of presets can now also be run locally by sound designers during implementation, which enables them to directly apply fixes, which implements the change types *C8* and *C9* in Table B.1. This is an important improvement, as in the previous version of the process automated fixes were implicitly applied on a temporary basis during the build process, which caused issues to remain in the original models. Thus, a small reduction of the degree of automation (manually triggering the fix process) was accepted to reach consistency. In addition, the validation rules became explicit input of the

verification and are configured/manipulated by developers (manifesting changes C_4 , and C_5). Although this increases the number of languages the developers have to deal with, the flexibility of the automated verification was enhanced. This improves the ability to adapt verification to language evolution of the DSL. A further reason for evolution is organizational change. There is a long-term plan to migrate the build process of libraries closer into the build process of the actual software development, which is done based on a publicly available build server (Jenkins²). Goal is to have a central build system to achieve a central visibility of the build process. Further, it should decrease the maintenance efforts that are currently required to maintain both systems. Thus, splitting up different aspects of the build process prepares the whole development for future evolution.

Despite the applied and planned evolution, some wishes are not fulfilled, since they are currently not rated as profitable. For example, the migration of models and generation implementation to a new version of the DSL is still a manual task.

B.3. Cap1 Evolution History

Figures B.1 and B.2 summarized the evolution history of case study Cap1.

B.4. Cap2a and Cap2b Evolution Histories

Figures B.3, B.4, and B.5 summarized the evolution history of the case studies Cap2a and Cap2b.

B.5. VCat Evolution History

Figure B.6 summarized the evolution history of case study VCat.

B.6. Carmeq Evolution History

Figures B.7 and B.8 summarized the evolution history of case study Carmeq.

²<http://jenkins-ci.org/>

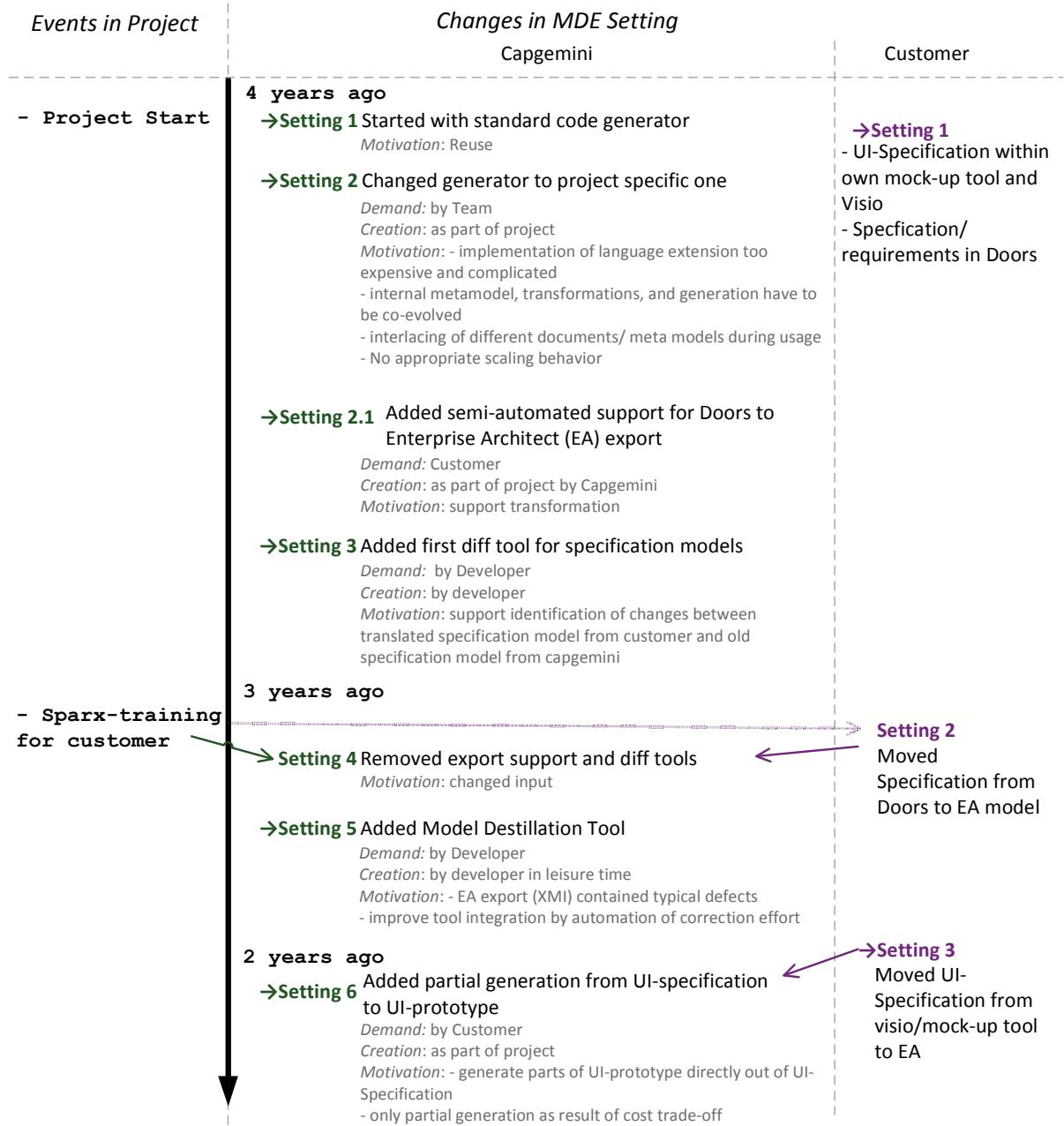


Figure B.1.: Summary of evolution history of case study Cap1 (part 1)

Table B.1.: Summary and classification of captured structural evolution steps (• = documented change)

Case Studies	Evolution step	C4 change number of artifacts	C5 change number of languages	C6 change number of manual activities	C7 change number of tools	C8 change number of automated activities	C9 change order of manual / automated activities
Cap1	S1				•	•	•
Cap1	S2	•	•		•	•	•
Cap1	S3	•	•		•	•	•
Cap1	S4			•			•
Cap1	S5	•	•	•	•	•	•
Cap1	S6	•			•	•	•
Cap1	S7				•	•	•
Cap2a	S1		•	•	•		
Cap2a	S2	•	•				
Cap2a	S3						
Cap2a	S4						
Cap2a	S5	•	•			•	•
Cap2b	S1	•	•	•	•		
Cap2b	S2	•	•			•	
Cap2b	S3	•	•			•	•
Cap2b	S4	•	•			•	
Cap2b	S5						
Cap2b	S6	•	•	•		•	
BO	S1	•	•	•	•	•	•
BO	S2	•	•	•	•	•	
BO	S3		•	•	•	•	•
BO	S4			•	•	•	
BO	S5	•	•	•	•	•	•
BO	S6	•	•	•	•	•	•
BO	S7	•	•	•	•	•	
Ableton	S1	•	•			•	•
VCat	S1	•		•	•	•	•
VCat	S2	•			•	•	
Carmeq	S1	•	•		•	•	•
Carmeq	S2			•	•	•	•
Carmeq	S3	•	•		•	•	•
Carmeq	S4				•		
Carmeq	S5				•		

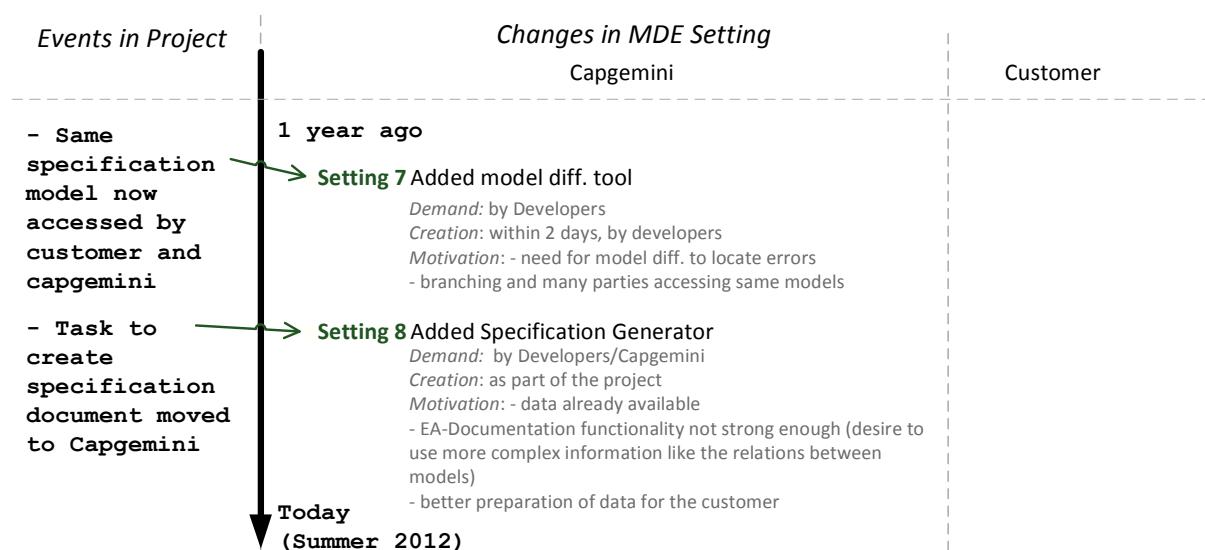


Figure B.2.: Summary of evolution history of case study Cap1 (part 2)

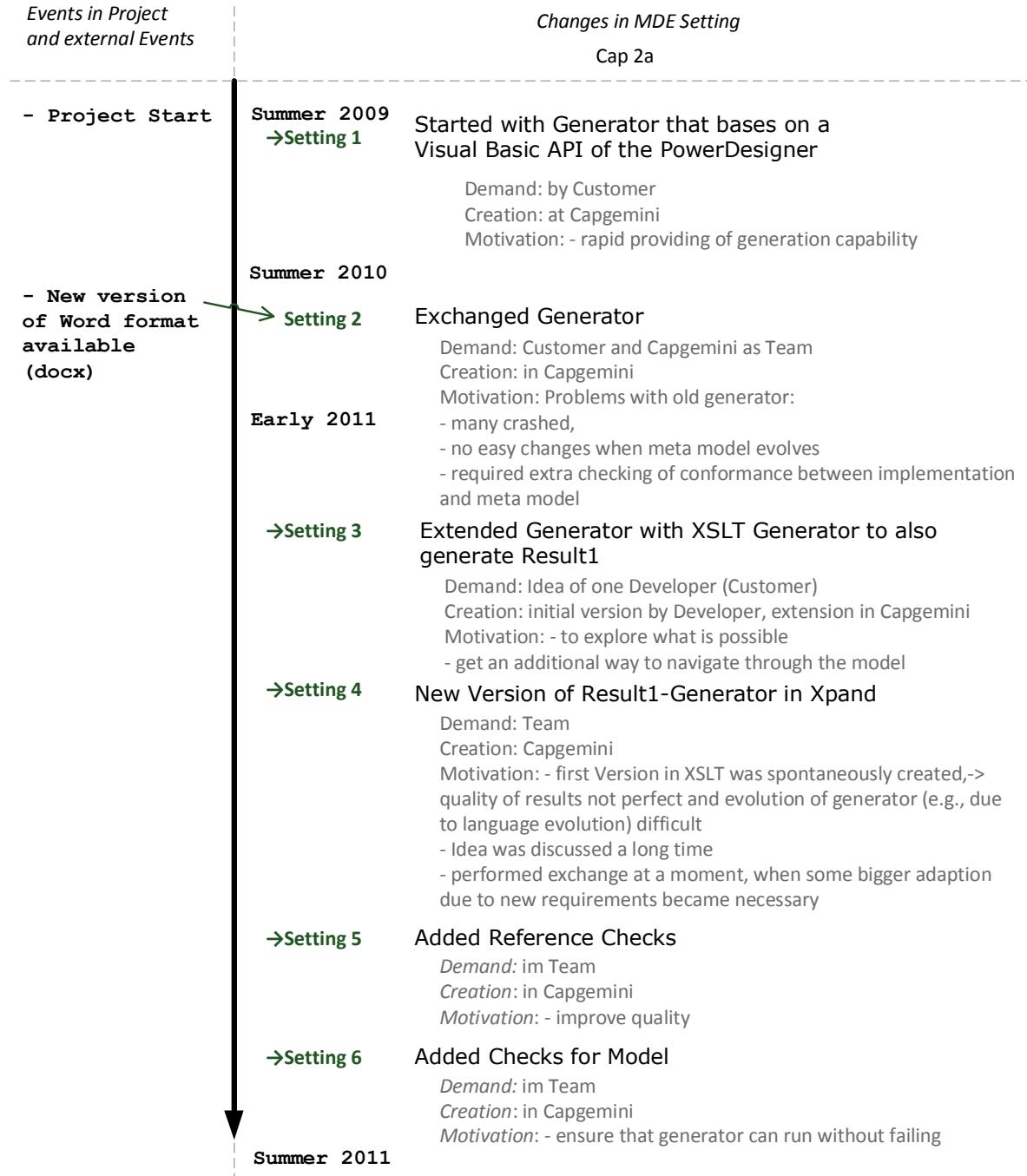


Figure B.3.: Summary of evolution history of case studies Cap2a

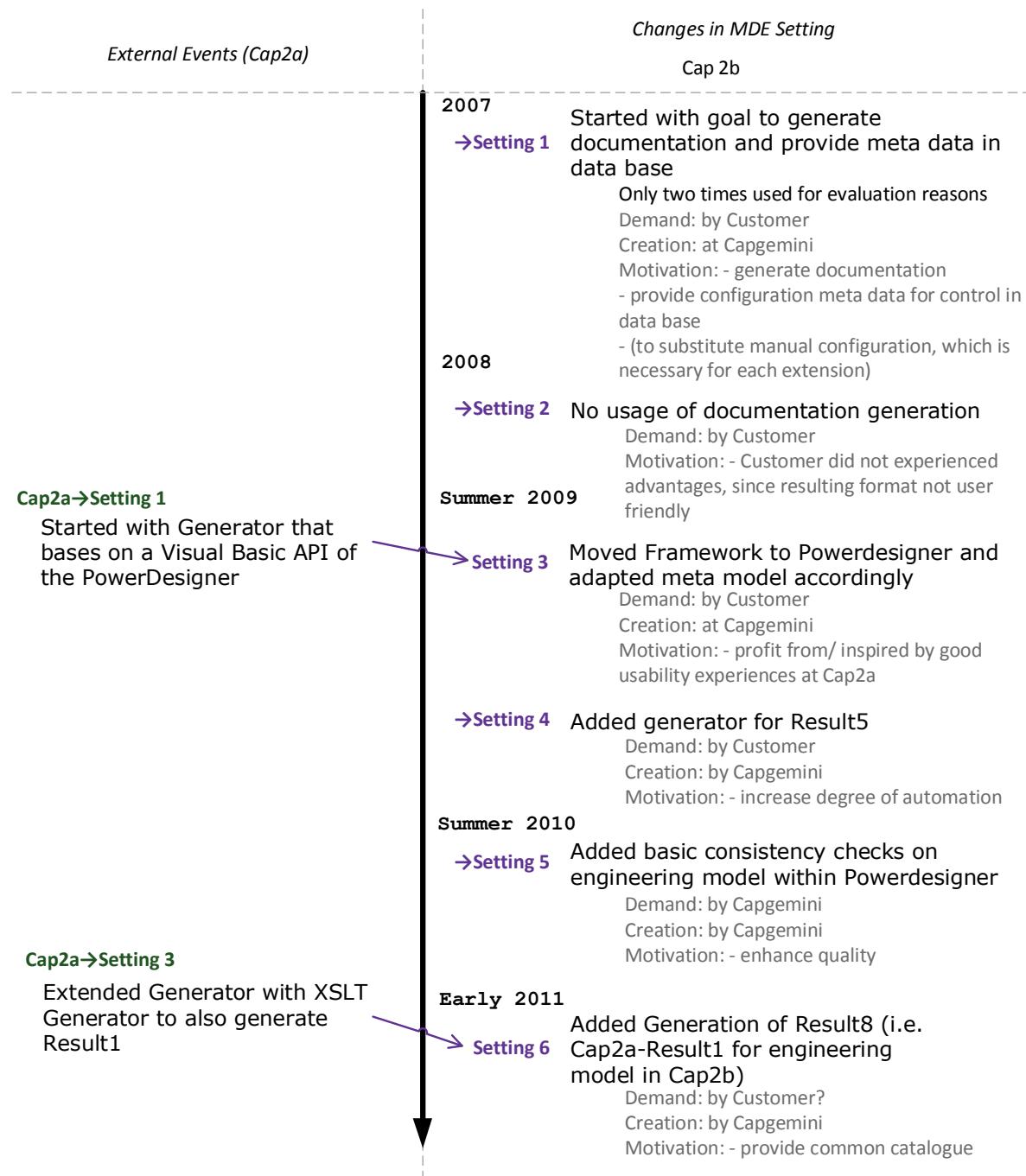


Figure B.4.: Summary of evolution history of case studies Cap2b (part 1)

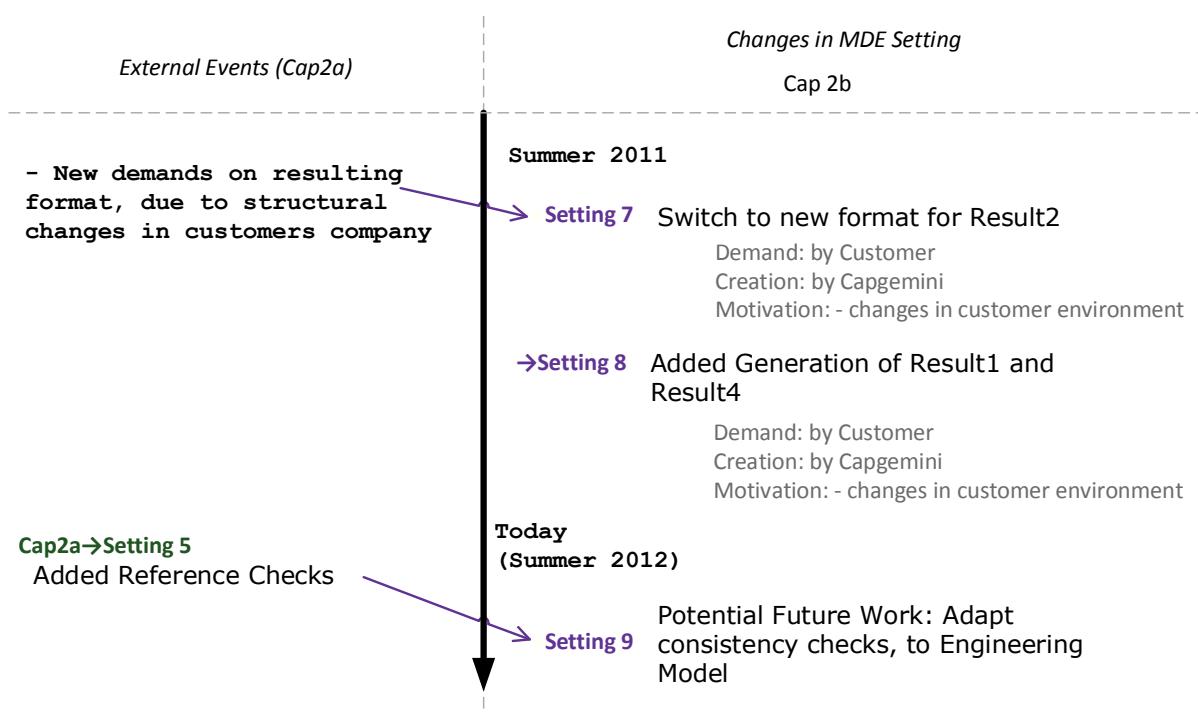


Figure B.5.: Summary of evolution history of case studies Cap2b (part2)

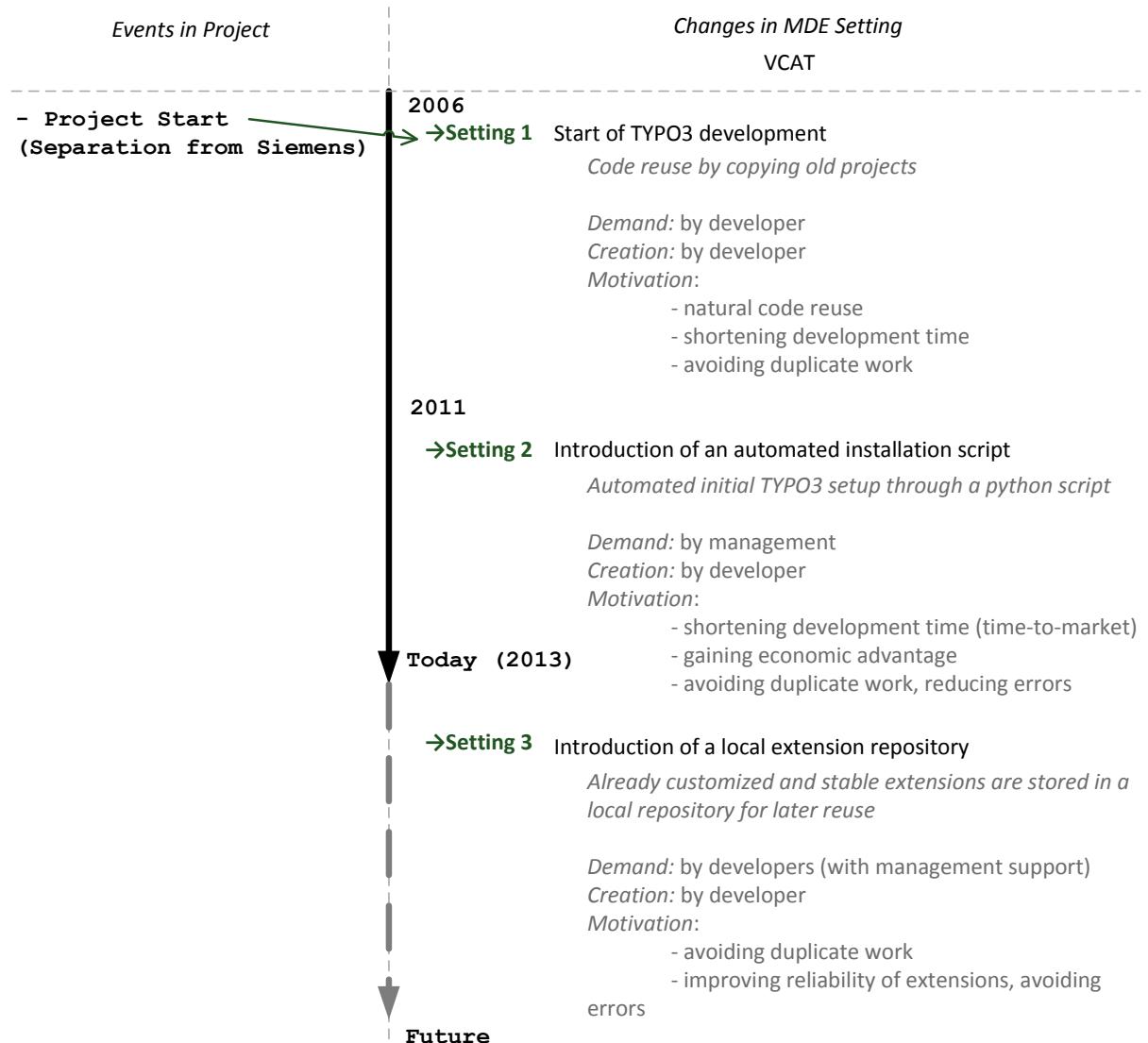


Figure B.6.: Summary of evolution history of case study VCat

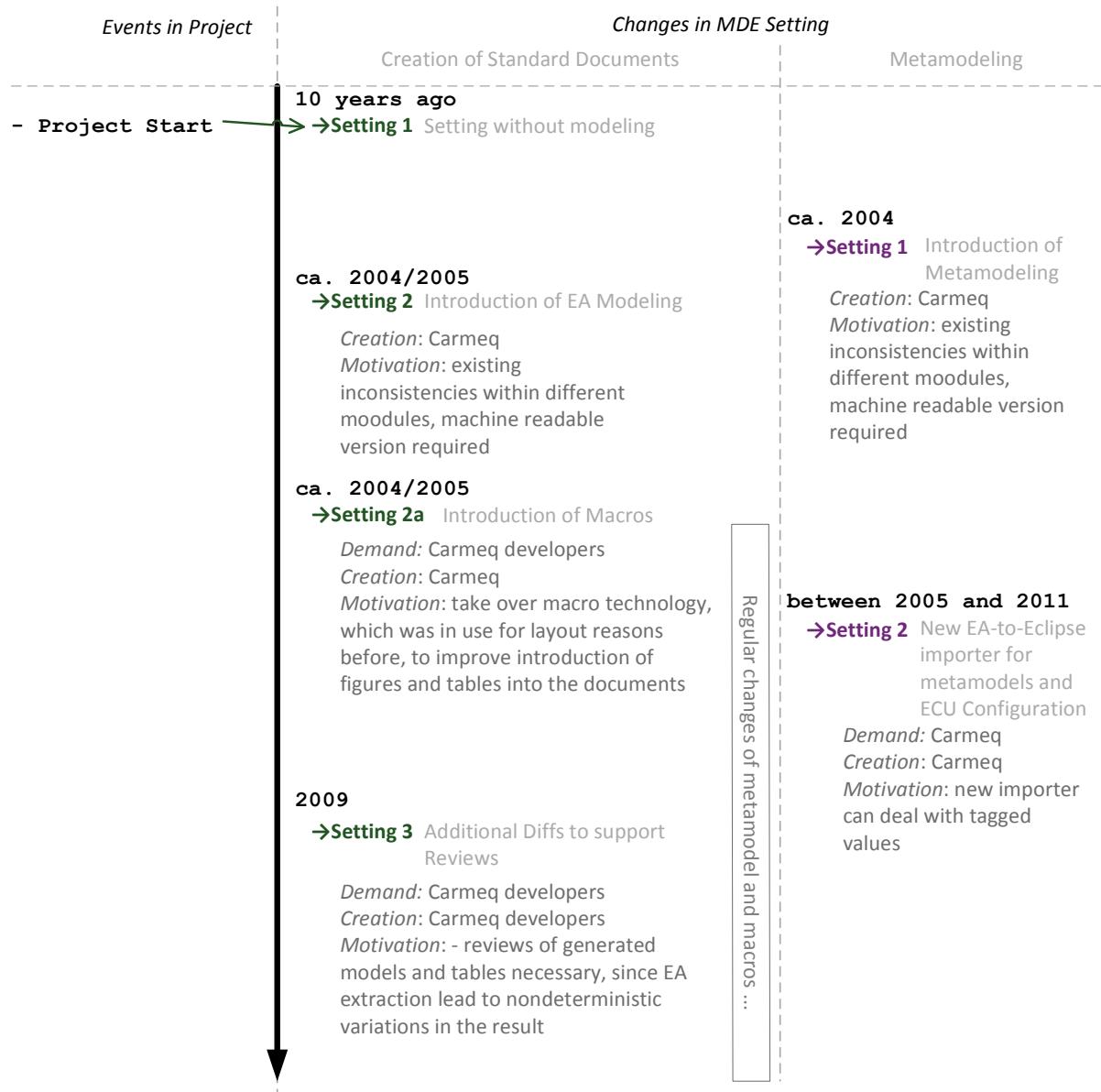


Figure B.7.: Summary of evolution history of case study Carmeq (part 1)

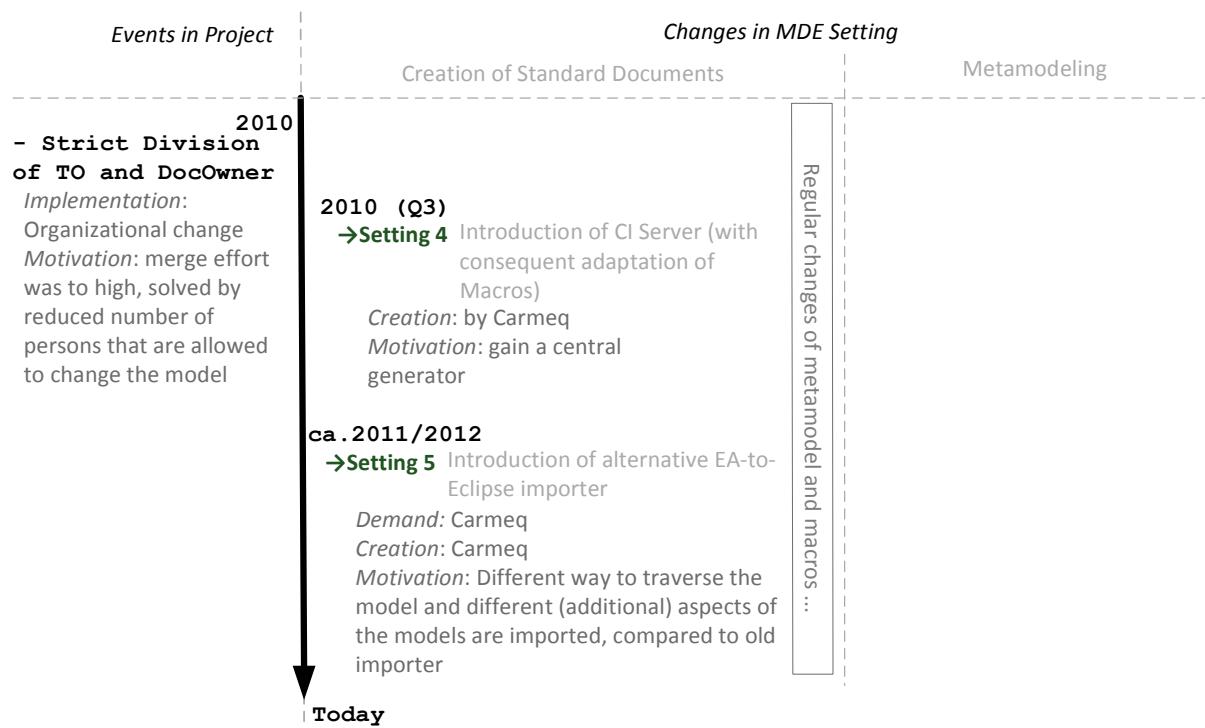


Figure B.8.: Summary of evolution history of case study Carmeq (part 2)

Appendix C.

Language Simplification

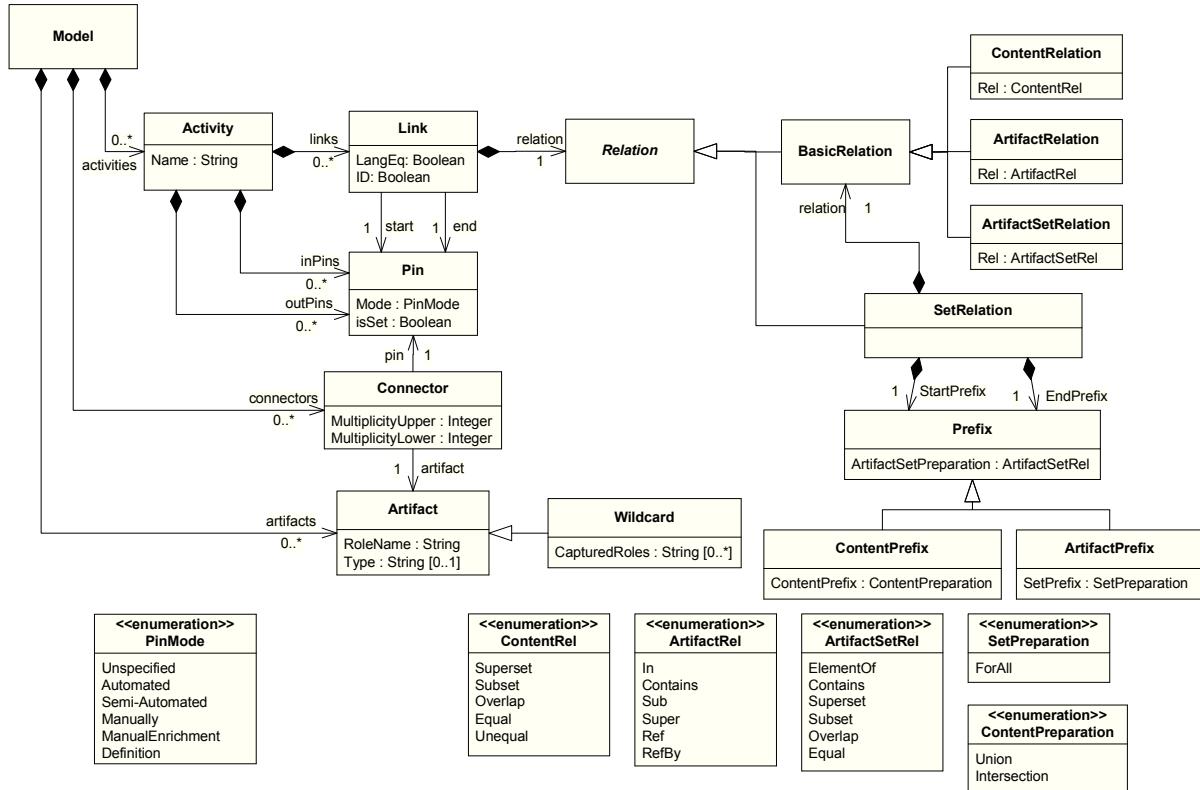


Figure C.1.: Meta model of initial Software Manufacture Model language

In the following, the difference between the initial Software Manufacture Model language that was presented in [P6] and the Software Manufacture Model language that is presented in this thesis are described. The simplifications have two kinds of impacts. On the one hand, a simplification can reduce the syntactic complexity of the language. Goal of such reductions is a reduced effort for creating and reading Software Manufacture Models. On the other hand, a simplification can affect semantic that is covered with the language. Thus, the set of Software Manufacture Model activities that can be expressed is reduced. This is mainly reached by reducing the degree to which details can be differentiated.

Notation and Formal Association of Degree of Automation

The first simplification concerns the degree of automation. In the initial version of the Software Manufacture Model language the degree of automation was defined for the pins. This enables to illustrate which relations in a semiautomated activity are established manually, semiautomated, or automated. This degree of detail can for example be useful when refactoring a Software Manufacture Model and the associated MDE setting. Thus, the differentiation gives a direct hint, how a semiautomated activity might be split more clearly into an automated and a manual activity. Alternatively this differentiation can be used to decide whether a semiautomated activity is a candidate for a full automation. In the Software Manufacture Model language the degree of automation is no longer associated to the pins, but to the activity. In consequence, the difference between manual and automated activities can be

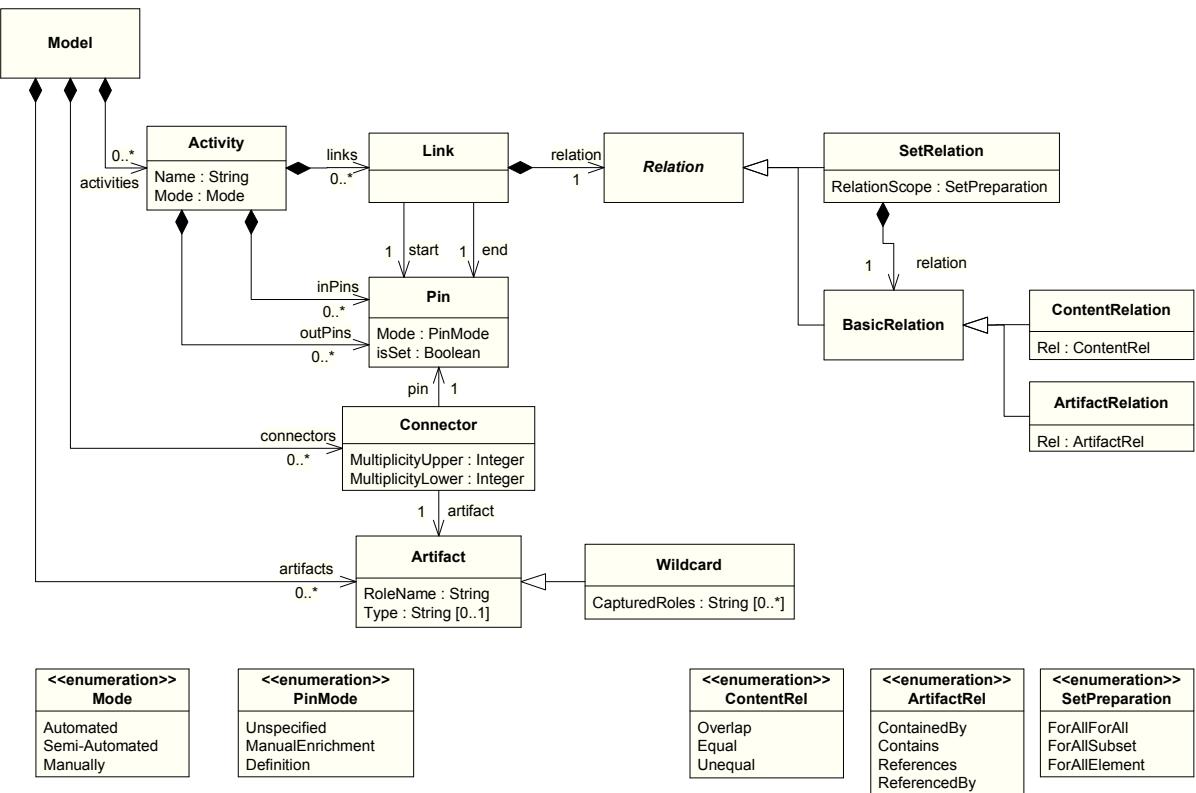


Figure C.2.: Meta model of Software Manufacture Model language

visualized with a single symbol. This reduces redundancy for all activities where all relations are established with the equal degree of automation. Further, the pin notation is no longer overloaded with this information. Both effects reduce the syntactic complexity. It is no longer possible to directly capture detailed information which relations of a semiautomated activity are established with which degree of automation. However, it is still possible to illustrate the different parts of such semiautomated activities in form of two or more activities as a decomposition. Thus, the information can still be captured in Software Manufacture Model language when required. Only 2 out of 193 so far documented activities from practice contain pins of two out of the three automation modes (manual, semiautomated, automated). Thus, the number of activities that are affected by this change is very low.

No Separation of Manual Enrichment Information

A purely syntactic reduction is the decision that there is no longer a separated manual enrichment pin if other outgoing pins for an artifact exist. The information about manual enrichment is annotated to one of the other outgoing pins that are connected to the described artifact. Only in case that no other outgoing pin is connected to an artifact, the manual enrichment is annotated with an own outgoing pin. This change positively affects the syntactical complexity by reducing the number of model elements. For the 193 so far documented activities from practice, 83 manual enrichment pins are modeled. Only 18 of these 83 manual enrichment pins are connected to artifacts that are not connected to any other outgoing pin of the activity. Thus, in 193 activities 65 manual enrichment pins have been found that can be saved due to the simplification. Only one out of these 65 manual enrichment pins is connected to an artifact that is connected to more than one additional output pin of the activity. In this rare case the decision which of these output pins is enriched with the information about the manual enrichment is arbitrary.

Simplification on Links

Two further simplifications are the removal of the identity property and the language equals property of links. Both properties were introduced in the initial version of the Software Manufacture Model language to provide the associated information explicitly as part of the activity characterization. Thereby, the equality of input and output languages is used as one basic property for differentiating between different types of transformations [48]. Both changes reduce the syntactic complexity by reducing the number of annotations in the model. The changes have no relevant semantic effect, since the information is still implicitly available in the model. The analysis techniques for changeability (changeability pattern and length of activity chain) do not depend on this information. Thus, there is currently no disadvantage associated to this simplification.

A further syntactic simplification is the changed annotation of the direction of the relation. So far the direction of the relation is annotated in the relation label as arrowhead. In the Software Manufacture Model language this annotation moves to the link, where it is directly annotated as arrowhead at one of the link ends. Consequently, the direction is easier and more intuitive to read. This holds especially in case of links between two incoming pins or two outgoing pins, respectively. The change affects all modeled links.

Reduction of Possibilities for Relations

Finally, the possibilities to describe relations have been reduced.

First of all, only two of the three basic relation types remained in the Software Manufacture Model. Thereby the *artifact set relations* have been removed, since there have been 7 uses, only. All of these 7 uses have been explicit expressions, that a specific artifact instance is chosen out of a set of instances. However, this choice is happens before each activity, anyway, and therefore does not need to be made explicit.

As a next step, the set of possible *artifact relations* have been limited. Thereby, the fine grained differentiation between *contains* and *super* (and *in* and *sub*) was not required in the modeled Software Manufacture Models. Consequently, only *contains* and *containedBy* relations remained (besides *references* and *referencedBy*). Similarly *content relations* have been limited, since not all degrees of detail have been differentiated in the models. Thereby, only *equal*, *overlap*, and *unequal* remain in the Software Manufacture Model language.

Finally, the *set relations* have been strongly simplified. *Content preparations* have not been used at all and *artifact set preparations* were used to express that single elements of a set are affected, only. As a consequence, the *start* and *end* prefixes have been substituted by a simple *relation scope*. This *relation scope* covers the actually used combinations of *set preparations* and *artifact set preparations*, which are *ForAllForAll*, *ForAllSubset*, and *ForAllElement*.

Appendix D.

Notation of Software Manufacture Model Language in Graphical Editor

As mentioned in Chapter 8 the Software Manufacture Model notation that is presented in this thesis differs from the notation in the graphical editor. In this section the differences are listed in Tableles D.1, D.2, D.3, and D.4.

Language concept	Notation in this thesis	Notation in graphical editor
automated activity		
manual activity		
semi-automated activity		

Table D.1.: Comparison of Software Manufacture Model notations (part 1)

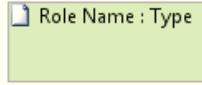
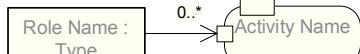
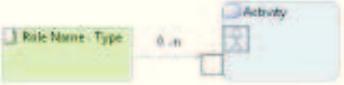
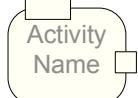
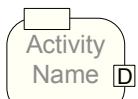
Language concept	Notation in this thesis	Notation in graphical editor
artifact		
wildcard		not supported by graphical editor
arbitrary number of artifacts are consumed by activity		
one artifact is produced by activity		
pin		
set pin		
manual enrichment pin		
definition pin		

Table D.2.: Comparison of Software Manufacture Model notations (part 2)

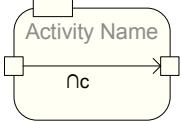
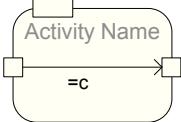
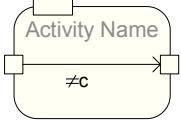
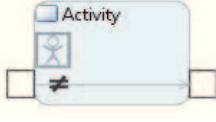
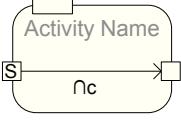
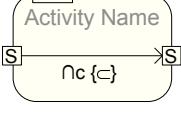
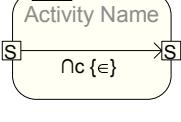
Language concept	Notation in this thesis	Notation in graphical editor
link with overlap relation		
link with equal relation		
link with unequal relation		
link with for-all-for-all preparation		
link with for-all-subset preparation		
link with for-all-element preparation		

Table D.3.: Comparison of Software Manufacture Model notations (part 3)

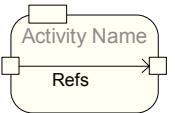
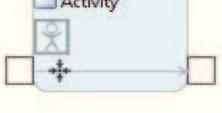
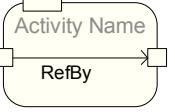
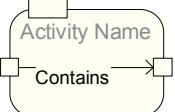
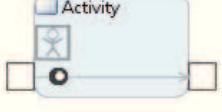
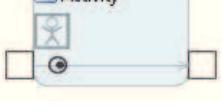
Language concept	Notation in this thesis	Notation in graphical editor
link with references relation		
link with referenced by relation		
link with contains relation		
link with contained by relation		

Table D.4.: Comparison of Software Manufacture Model notations (part 4)

Appendix E.

WSDL Example

Listing E.1 shows the WSDL example from <http://cs.au.dk/~amoeller/WWW/webservices/wsdlexample.html> (last access at November 9th, 2013) that is used in Section 7.1.

```
1  <?xml version="1.0"?>
2  <definitions name="StockQuote"
3      targetNamespace="http://example.com/stockquote.wsdl"
4      xmlns:tns="http://example.com/stockquote.wsdl"
5      xmlns:xsd1="http://example.com/stockquote.xsd"
6      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7      xmlns="http://schemas.xmlsoap.org/wsdl/">
8
9      <types>
10         <schema targetNamespace="http://example.com/stockquote.xsd"
11             xmlns="http://www.w3.org/2000/10/XMLSchema">
12             <element name="TradePriceRequest">
13                 <complexType>
14                     <all>
15                         <element name="tickerSymbol" type="string"/>
16                     </all>
17                 </complexType>
18             <element name="TradePrice">
19                 <complexType>
20                     <all>
21                         <element name="price" type="float"/>
22                     </all>
23                 </complexType>
24             </element>
25         </schema>
26     </types>
27     <message name="GetLastTradePriceInput">
28         <part name="body" element="xsd1:TradePriceRequest"/>
29     </message>
30     <message name="GetLastTradePriceOutput">
31         <part name="body" element="xsd1:TradePrice"/>
32     </message>
33     <portType name="StockQuotePortType">
34         <operation name="GetLastTradePrice">
35             <input message="tns:GetLastTradePriceInput"/>
36             <output message="tns:GetLastTradePriceOutput"/>
37         </operation>
38     </portType>
39     <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
40         <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
41         <operation name="GetLastTradePrice">
42             <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
43             <input>
44                 <soap:body use="literal"/>
45             </input>
46             <output>
47                 <soap:body use="literal"/>
48             </output>
49         </operation>
50     </binding>
51     <service name="StockQuoteService">
52         <documentation>My first service</documentation>
53         <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
54             <soap:address location="http://example.com/stockquote"/>
55         </port>
56     </service>
57 </definitions >
```

Listing E.1: WSDL example from <http://cs.au.dk/~amoeller/WWW/webservices/wsdlexample.html>

Appendix F.

Code Listings

In this Appendix, in detail commented java code listings for the analysis are listed. First the code for the identification of predecessor sets, predecessors and successors is documented. Afterwards listings for the identification of synchronization points and the length of activity chains is presented.

F.1. Identification of Precesessor Sets

Listings F.1 and F.2 show the algorithm for the identification of predecessor sets for a given activity. In this context the recursive method `extendCandidateSet` (shown in Listings F.3 and F.4) is used to extend the candidate predecessor sets transitively by traversing through already identified predecessors and identifying their predecessor sets, respectively.

```
1  EList<EList<Activity>> getPredecessorSets(
2      Activity activity,
3      Model swmamo)
4  {
5      /** 1. get all artifacts that are consumed by an activity */
6      EList<Artifact> consumedArtifacts = getConsumedArtifacts(activity, swmamo);
7
8      EList<EList<EList<Activity>>>
9          uncombinedListsOfPredecessorSetCandidatesForConsumedArtifacts =
10         new BasicEList<EList<EList<Activity>>>();
11
12     /** 2. for each artifact find all activities that
13      /* produce or manipulate this artifact */
14     for (Artifact artifact : consumedArtifacts) {
15         EList<EList<Activity>> predecessorSetCandidates =
16             getPredecessorSetCandidatesForArtifact(artifact, swmamo);
17
18         /** 3. if an artifact is optional input an alternative empty
19         /* predecessor set candidate for this artifact is added */
20         if(this.isOptionalConsumedArtifact(artifact, activity, swmamo))
21             predecessorSetCandidates.add(new BasicEList<Activity>());
22
23         /** FILTER 1: remove predecessor set candidates that contain the activity
24         /* (i.e. an activity cannot be its own predecessor) */
25         filterRemovePredecessorSetCandidatesThatContainActivity(
26             predecessorSetCandidates, activity);
27
28         uncombinedListsOfPredecessorSetCandidatesForConsumedArtifacts.add(
29             predecessorSetCandidates);
30     }
31
32     /** 4. get all combinations of candidate predecessor sets
33     /* (one candidate predecessors set per consumed activity) */
34     EList<EList<Activity>> initialCandidateSets = this.combineSetCandidats(
35         uncombinedListsOfPredecessorSetCandidatesForConsumedArtifacts);
36
37     /** FILTER 2: Remove candidate sets that are not able to fulfill precondition of
38     /* activity (precondition links that are connected to optional input
39     /* artifacts do not need to be fulfilled, when the optional input
40     /* artifact is not provided by the predecessor set) */
41     filterRemovePredecessorSetsThatCanNotFulfillPreconditions(
42         activity, initialCandidateSets, swmamo);
43
44     /** 5. register these consumed artifacts and the activity as handled */
45     EList<Artifact> handledArtifacts = new BasicEList<Artifact>();
46     handledArtifacts.addAll(consumedArtifacts);
47     EList<Activity> handledActivities = new BasicEList<Activity>();
48     handledActivities.add(activity);
49
50 } [...]
```

Listing F.1: `getPredecessorSets()` (part 1)

Appendix F. CodeListings

```

1  EList<EList<Activity>> getPredecessorSets(
2      Activity activity ,
3      Model swmamo)
4  {
5      [...]
6
7      /** 6. extend all initial candidate predecessor sets
8      /* (i.e. retrieve predecessor sets for the initial candidates) */
9      EList<EList<Activity>> CandidateSets = new BasicEList<EList<Activity>>();
10     for(int i = 0;i<initialCandidateSets.size();i++)
11         CandidateSets.addAll(extendCandidateSet(initialCandidateSets.get(i),
12             activity , swmamo, handledArtifacts , handledActivities));
13
14     /** FILTER 3 (= FILTER 1) again remove candidate sets that contain the activity
15     /* (i.e. an activity cannot be its own predecessor) */
16     filterRemovePredecessorSetCandidatesThatContainActivity(CandidateSets , activity);
17
18     /** FILTER 4: remove clones (equal or cloned candidate sets can result form the
19     /* combination and extension of candidate sets for the different artifacts) */
20     filterRemoveClones(CandidateSets);
21
22
23     /** FILTER 5: remove predecessor sets with creation overkill (i.e. if for one
24     /* artifact in the model multiple activities in predecessor set (or the
25     /* activity considered here) can be used to create this artifact) */
26     filterRemovePredecessorSetsWithCreationOverkill(CandidateSets , activity , swmamo);
27
28     return CandidateSets;
29 }
```

Listing F.2: getPredecessorSets() (part 2)

```

1  EList<EList<Activity>> extendCandidateSet(
2      EList<Activity> initialCandidateSet ,
3      Activity initialActivity ,
4      Model swmamo,
5      EList<Artifact> handledArtifacts ,
6      EList<Activity> handledActivities)
7  {
8      EList<EList<Activity>> resultingSets = new BasicEList<EList<Activity>>();
9
10     /** 1. identify activities in initial candidate set that are not yet handled */
11     EList<Activity> localHandledActivities = new BasicEList<Activity>();
12     localHandledActivities.addAll(handledActivities);
13     EList<Activity> unhandledActivities = getUnhandledActivities(
14         initialCandidateSet , localHandledActivities);
15
16     /** 2. identify artifacts consumed by unhandled activities in
17     /* initial candidate set that are not yet handled */
18     EList<Artifact> localHandledArtifacts = new BasicEList<Artifact>();
19     localHandledArtifacts.addAll(handledArtifacts);
20     EList<Artifact> unhandledArtifacts = getUnhandledArtifacts(
21         unhandledActivities , localHandledArtifacts , swmamo);
22
23     /** 3. BREAK CONDITION: if no artifacts are left for treatment stop search and
24     /* return a set of candidate sets that include the initial candidate set, only*/
25     if(unhandledArtifacts.size()<1){
26         resultingSets.add(initialCandidateSet);
27         return resultingSets;
28     }
29
30     /** 4. create uncombined list of lists of candidate sets and add list that
31     /* includes initial candidate set, only, as basis for later combination */
32     EList<EList<EList<Activity>>> uncombinedListOfCandidateSets =
33         new BasicEList<EList<EList<Activity>>>();
34     EList<EList<Activity>> listWithInitialSet = new BasicEList<EList<Activity>>();
35     listWithInitialSet.add(initialCandidateSet);
36     uncombinedListOfCandidateSets.add(listWithInitialSet);
37
38     [...]
39 }
```

Listing F.3: extendCandidateSet() (part 1)

```

1 EList<EList<Activity>> extendCandidateSet(
2     EList<Activity> initialCandidateSet ,
3     Activity initialActivity ,
4     Model swmamo,
5     EList<Artifact> handledArtifacts ,
6     EList<Activity> handledActivities )
7 {
8     [...]
9
10    /** 5. handle all unhandled artifacts **/
11    for( Artifact unhandledArtifact: unhandledArtifacts){
12        /** 5a. retrieve candidate sets for all unhandled artifact **/
13        EList<EList<Activity>> candidateSetsForArtifact =
14            getPredecessorSetCandidatesForArtifact(unhandledArtifact , swmamo);
15
16        /** 5b. if an artifact is optional input for all activities in
17         *      initialCandidateSet that consume this activity an alternative empty
18         *      predecessor set candidate for this artifact is added **/
19        if(artifactIsOptionalForAllActivitiesInInitialCandidateSet(
20            unhandledArtifact , initialCandidateSet , swmamo))
21            candidateSetsForArtifact.add(new BasicEList<Activity>());
22
23        /** 5c. add resulting list of candidate sets for the unhandled artifacts
24         *      to the uncombinedListOfCandidateSets **/
25        uncombinedListOfCandidateSets.add(candidateSetsForArtifact);
26
27        /** 5d. register unhandledArtifact as handled **/
28        localHandledArtifacts.add(unhandledArtifact);
29    }
30
31    /** 6. get all combinations of candidate predecessor sets
32     *      (one candidate predecessors set per consumed activity) **/
33    EList<EList<Activity>> newCandidateSets = combineSetCandidats(
34        uncombinedListOfCandidateSets);
35
36    /** FILTER 6 (= FILTER 2) Remove candidate sets that are not able to fulfill
37     *      precondition of activity (precondition links that are connected to
38     *      optional input artifacts do not need to be fulfilled, when the
39     *      optional input artifact is not provided by the predecessor set) **/
40    filterRemovePredecessorSetsThatCanNotFulfillPreconditions(
41        activity , newCandidateSets , swmamo);
42
43    /** FILTER 7 (= FILTER 3 = FILTER 1) again remove candidate sets that contain the
44     *      very initial activity (i.e. an activity cannot be its own predecessor) **/
45    filterRemovePredecessorSetCandidatesThatContainActivity(
46        newCandidateSets , activity );
47
48    /** 7. register unhandled activities as handled **/
49    localHandledActivities.addAll(unhandledActivities);
50
51    /** 8. recursively call this method for each new candidate set **/
52    for(EList<Activity> candidate : newCandidateSets)
53        resultingSets.addAll(this.extendCandidateSet(candidate ,
54            initialActivity , swmamo, localHandledArtifacts , localHandledActivities));
55
56    /** 9. return resulting list of candidate sets **/
57    return resultingSets;
58 }
```

Listing F.4: extendCandidateSet() (part 2)

F.2. Identification of Phases and Synchronization Points

In this Section the code for the derivation of phases and synchronization points is listed. Thereby, Listing F.5 shows the frame of the algorithm, Listing F.6 includes the check whether an activity is a local synchronization point within a given local focus, and Listing F.7 includes the derivation of the phases based on a given local focus with given local synchronization points. Further, the derivation of local synchronizations points including the search for promising local foci is shown in Listings F.8, F.9, F.10, and F.11.

Appendix F. CodeListings

```

1 void identifyGlobalSynchronizationPoints(
2     Model swmamo,
3     EMap<Activity, EList<Activity>> predecessorsPerActivity,
4     EMap<Activity, EList<Activity>> successorsPerActivity,
5     EMap<Activity, EList<EList<Activity>>> predecessorSetsPerActivity)
6 {
7     /** for each activity identify whether the activity is a local synchronization
8      * point for a focus that covers the whole model */
9     EList<Activity> synchs = new BasicEList<Activity>();
10    for(Activity activity : swmamo.getActivity()){
11        if(this.isLocalSynchronizationPoint(activity, swmamo.getActivity(),
12            predecessorsPerActivity, successorsPerActivity, predecessorSetsPerActivity))
13            synchs.add(activity);
14    }
15
16    this.printoutPhasesForCombinedLocalSynchs(synchs, swmamo.getActivity(), swmamo,
17        predecessorsPerActivity, successorsPerActivity, predecessorSetsPerActivity);
18
19    this.identifyLocalSynchronizationPoints(swmamo, predecessorsPerActivity,
20        successorsPerActivity, predecessorSetsPerActivity);
21 }
```

Listing F.5: identifyGlobalSynchronizationPoints()

```

1 boolean isLocalSynchronizationPoint(
2     Activity candidate,
3     EList<Activity> localActivities,
4     EMap<Activity, EList<Activity>> predecessorsPerActivity,
5     EMap<Activity, EList<Activity>> successorsPerActivity,
6     EMap<Activity, EList<EList<Activity>>> predecessorSetsPerActivity)
7 {
8     EList<Activity> successors = successorsPerActivity.get(candidate);
9     EList<Activity> predecessors = predecessorsPerActivity.get(candidate);
10
11    /** 1. for each activity in the given focus check whether this activity is either
12     * predecessor or successor (and not both) of the candidate */
13    boolean allArtifactsInStrictOrder = true;
14    for(Activity a: localActivities){
15        boolean strictOrderExists = false;
16        if(a.equals(synch)) strictOrderExists = true;
17        if(successors.contains(a) && (!predecessors.contains(a)))
18            strictOrderExists = true;
19        if(!successors.contains(a) && predecessors.contains(a))
20            strictOrderExists = true;
21
22        if(!strictOrderExists)
23            allArtifactsInStrictOrder = false;
24    }
25
26    /** 2. if relative order to all other artifacts is clear the activity
27     * is a local synchronization point */
28    if(allArtifactsInStrictOrder)
29        return true;
30
31    return false;
32 }
```

Listing F.6: isLocalSynchronizationPoint()

```

1 void printoutPhasesForCombinedLocalSynchs(
2     EList<Activity> synchs ,
3     EList<Activity> localSet ,
4     Model swnamo ,
5     EMap<Activity , EList<Activity>> predecessorsPerActivity ,
6     EMap<Activity , EList<Activity>> successorsPerActivity ,
7     EMap<Activity , EList<EList<Activity>>> predecessorSetsPerActivity )
8 {
9     /** 1. put synchronization points in order */
10    EList<Activity> orderedSynchronizationPoints = new BasicEList<Activity>();
11    if (!synchs.isEmpty())
12        orderedSynchronizationPoints.add(synchs.get(0));
13    for (int i = 1; i < synchs.size(); i++) {
14        Activity s = synchs.get(i);
15        /** identify new position */
16        int newPosition = orderedSynchronizationPoints.size();
17        for (int j = 0; j < orderedSynchronizationPoints.size(); j++) {
18            if (successorsPerActivity.get(s).contains(orderedSynchronizationPoints.get(j)))
19                if (j < newPosition)
20                    newPosition = j;
21        }
22        EList<Activity> helper = new BasicEList<Activity>();
23        for (int k = 0; k < orderedSynchronizationPoints.size(); k++) {
24            if (k == newPosition)
25                helper.add(s);
26            helper.add(orderedSynchronizationPoints.get(k));
27        }
28        orderedSynchronizationPoints = helper;
29    }
30
31    /** 2. iterate ordered synchronization points and print out all activities
32     * that are predecessors of that activity */
33    EList<Activity> toConsider = new BasicEList<Activity>();
34    toConsider.addAll(localSet);
35    for (int i = 0; i < orderedSynchronizationPoints.size(); i++) {
36        Activity synch = orderedSynchronizationPoints.get(i);
37        System.out.println("before_synchronization_point_" + synch.getName());
38
39        EList<Activity> removeA = new BasicEList<Activity>();
40        for (Activity act: toConsider) {
41            if (act.equals(synch))
42                removeA.add(act);
43            if (predecessorsPerActivity.get(synch).contains(act)) {
44                removeA.add(act);
45                System.out.println(act.getName());
46            }
47        }
48        /** remove already considered activities from list */
49        for (Activity r : removeA)
50            toConsider.remove(r);
51    }
52
53
54    /** 3. printout activities that are successors of all synchronization points */
55    System.out.println("after_all_synchronization_points:");
56    for (Activity act: toConsider) {
57        System.out.println(act.getName());
58    }
59
60    return;
61 }
```

Listing F.7: printoutPhasesForCombinedLocalSynchs()

Appendix F. CodeListings

```

1 void identifyLocalSynchronizationPoints(
2     Model swmamo,
3     EMap<Activity, EList<Activity>> predecessorsPerActivity,
4     EMap<Activity, EList<Activity>> successorsPerActivity,
5     EMap<Activity, EList<EList<Activity>>> predecessorSetsPerActivity)
6 {
7     EMap<EList<Activity>, EList<Activity>> localSynchWithFocus =
8         new BasicEMap<Activity, EList<Activity>>();
9
10    /** consider each activity in the software manufacture model as potential
11     * local synchronization points */
12    for(Activity candidate : swmamo.getActivity()){
13        /** 1. identify all activities that have the candidate activity as successor */
14        EList<Activity> haveCandidateAsSuccessor = new BasicEList<Activity>();
15        for(Activity act : swmamo.getActivity())
16            if(successorsPerActivity.get(act).contains(candidate))
17                haveCandidateAsSuccessor.add(act);
18
19        /** 2. identify all activities that require the candidate to be executed
20         * before they can be executed */
21        EList<Activity> requireCandidateBefore = this.requireActivityBefore(
22            candidate, swmamo, predecessorSetsPerActivity);
23
24        /** 3. consider all candidates for which activities exist that have this
25         * candidate as successor as well as activities that require this
26         * candidate before they can be executed */
27        if((haveCandidateAsSuccessor.size() > 0) && (requireCandidateBefore.size() > 0)){
28
29            /** 4. retrieve maximal possible local focus in which the candidate can be
30             * a local synchronization point */
31            EList<Activity> localFocus = new BasicEList<Activity>();
32            localFocus.addAll(haveCandidateAsSuccessor);
33            localFocus.addAll(requireCandidateBefore);
34
35            /** 5. check whether candidate actually is local synchronization point in
36             * that local focus */
37            boolean isSynch = this.isLocalSynchronizationPoint(candidate, localFocus,
38                predecessorsPerActivity, successorsPerActivity,
39                predecessorSetsPerActivity);
40            if(isSynch){
41                /** candidate is local synchronization point in that local focus */
42
43                /** check whether other activities in this local focus become
44                 * local synchronization points, too */
45                EList<Activity> synchs = new BasicEList<Activity>();
46                synchs.add(candidate);
47
48                for(Activity a: localFocus)
49                    if(this.isLocalSynchronizationPoint(a, localFocus,
50                        predecessorsPerActivity, successorsPerActivity,
51                        predecessorSetsPerActivity))
52                        synchs.add(a);
53
54                localSynchWithFocus.put(synchs, localFocus);
55
56                this.printoutPhasesForCombinedLocalSynchs(synchs, localFocus, swmamo,
57                    predecessorsPerActivity, successorsPerActivity,
58                    predecessorSetsPerActivity);
59            }
60        }
61    }
62
63    /** 6. identify possible combinations of local foci */
64    EMap<EList<Activity>, EList<Activity>> combis =
65        this.getAllCombinationsOfLoalSynchs(localSynchWithFocus, swmamo,
66            predecessorsPerActivity, successorsPerActivity, predecessorSetsPerActivity);
67
68    return;
69 }
```

Listing F.8: identifyLocalSynchronizationPoints()

```

1 EMap<EList<Activity>,EList<Activity>> getAllCombinationsOfLoalSynchs(
2   EMap<Activity , EList<Activity>> localSynchs ,
3   Model swmamo ,
4   EMap<Activity , EList<Activity>> predecessorsPerActivity ,
5   EMap<Activity , EList<Activity>> successorsPerActivity ,
6   EMap<Activity , EList<EList<Activity>>> predecessorSetsPerActivity )
7 {
8   EMap<EList<Activity>,EList<Activity>> result =
9     new BasicEMap<EList<Activity>,EList<Activity>>();
10
11  /** 1. get list of all local synchronization points */
12  EList<Activity> synchs = new BasicEList<Activity>();
13  for(Entry<Activity , EList<Activity>> key: localSynchs)
14    synchs.add(key.getKey());
15
16  /** 2. retrieve all possible combinations of synchronization points */
17  EList<EList<Activity>> combis = new BasicEList<EList<Activity>>();
18  for(int i = 2;i<(localSynchs.size()+1);i++)
19    combis.addAll(this.getCombinationsOfI(i, synchs));
20
21  /** 3. for each possible combination of synchronization points get
22  /* combined local foci */
23  for(EList<Activity> c : combis){
24    EMap<Activity , EList<Activity>> param =
25      new BasicEMap<Activity , EList<Activity>>();
26    for(Activity a : c)
27      param.add(localSynchs.getKey(localSynchs.indexOfKey(a)));
28    EMap<EList<Activity>,EList<Activity>> r = this.getCombinedLocalizationOfSynchs(
29      param, swmamo, predecessorsPerActivity, successorsPerActivity,
30      predecessorSetsPerActivity);
31    result.addAll(r);
32  }
33
34  return result;
35 }
```

Listing F.9: getAllCombinationsOfLoalSynchs()

```

1 EMap<EList<Activity>,EList<Activity>> getCombinedLocalizationOfSynch(
2   EMap<Activity , EList<Activity>> localSynchs ,
3   Model swmamo ,
4   EMap<Activity , EList<Activity>> predecessorsPerActivity ,
5   EMap<Activity , EList<Activity>> successorsPerActivity ,
6   EMap<Activity , EList<EList<Activity>>> predecessorSetsPerActivity )
7 {
8   EMap<EList<Activity>,EList<Activity>> result =
9     new BasicEMap<EList<Activity>,EList<Activity>>();
10
11  /** 1. Check whether combination is possible: is each synchronization point part
12  /* of the local sets of each other synchronizations point */
13  for(Entry<Activity , EList<Activity>> e : localSynchs){
14    boolean inAllOtherFoci = true;
15    for(Entry<Activity , EList<Activity>> in : localSynchs){
16      boolean isInAllThisFocus = true;
17      if(!in.getKey().equals(e.getKey())){
18        boolean test = false;
19        for(Activity a: in.getValue())
20          if(a.equals(e.getKey()))
21            test = true;
22        if(!in.getValue().contains(e.getKey()))
23          isInAllThisFocus = false;
24      }
25      if(!isInAllThisFocus)
26        inAllOtherFoci = false;
27    }
28
29    /** if synchronization point is not part of other local foci
30    /* return empty result */
31    if(!inAllOtherFoci)
32      return result;
33  }
34
35 } [...]
```

Listing F.10: getCombinedLocalizationOfSynch() (part 1)

```

1 EMap<EList<Activity>,EList<Activity>> getCombinedLocalizationOfSynch(
2     EMap<Activity ,EList<Activity>> localSynchs ,
3     Model swmamo,
4     EMap<Activity ,EList<Activity>> predecessorsPerActivity ,
5     EMap<Activity ,EList<Activity>> successorsPerActivity ,
6     EMap<Activity ,EList<EList<Activity>>> predecessorSetsPerActivity)
7 {
8     [...]
9
10    /** 2. retrieve list of all activities is local foci */
11    EList<Activity> allActs = new BasicEList<Activity>();
12    for(Entry<Activity ,EList<Activity>> e : localSynchs)
13        for(Activity a: e.getValue())
14            if(!allActs.contains(a))
15                allActs.add(a);
16
17    /** 3. combine local foci, such that new local set combines all activities
18     *      that are part of each local focus */
19    EList<Activity> newLocalSet = new BasicEList<Activity>();
20    for(Activity a: allActs){
21        boolean isInAll = true;
22        for(Entry<Activity ,EList<Activity>> e : localSynchs){
23            if(!e.getValue().contains(a))
24                isInAll = false;
25        }
26        if(isInAll)
27            newLocalSet.add(a);
28    }
29
30    /** 4. check whether all combined local synchronization points remain
31     *      local synchronization points in new local focus */
32    boolean allSynchs = true;
33    EList<Activity> synchs = new BasicEList<Activity>();
34    for(Entry<Activity ,EList<Activity>> e : localSynchs){
35        if(!this.isLocalSynchronizationPoint(e.getKey(), newLocalSet,
36            predecessorsPerActivity , successorsPerActivity , predecessorSetsPerActivity))
37            allSynchs = false;
38        else
39            synchs.add(e.getKey());
40    }
41
42    /** 5. collect all additional activities that are local synchronization
43     *      points in that new local focus */
44    if(allSynchs){
45        for(Activity a: newLocalSet)
46            if(this.isLocalSynchronizationPoint(a, newLocalSet, predecessorsPerActivity ,
47                successorsPerActivity , predecessorSetsPerActivity ))
48                if(!synchs.contains(a))
49                    synchs.add(a);
50    }
51
52    result.put(synchs, newLocalSet);
53
54    this.printoutPhasesForCombinedLocalSynchs(synchs, newLocalSet, swmamo,
55        predecessorsPerActivity , successorsPerActivity , predecessorSetsPerActivity );
56
57    return result;
58 }
```

Listing F.11: getCombinedLocalizationOfSynch() (part 2)

F.3. Identification of Lengths of Activity Chains

In this Section the code for the identification of lengths of activity chains is listed. Thereby, Listing F.12 contains the frame code and the Listings F.13, F.14, and F.15 show the calculation of the lengths of activity chains for a given pair of artifacts. The Listings F.16, F.17, and F.18 include methods for the identification of relevant activities for a given artifact.

```

1 void calculateLengthsOfActivityChains(
2     Model swmamo,
3     EMap<Activity , EList<Activity>> predecessorsPerActivity ,
4     EMap<Activity , EList<Activity>> successorsPerActivity ,
5     EMap<Activity , EList<EList<Activity>>> predecessorSetsPerActivity )
6 {
7     /** 1. get list of all artifacts in the model (the code is written with respect
8      * to the fact that an artifact might occur as model element multiple times ,
9      * such that readability is increased) */
10    EList<Artifact> artifacts = swmamo.getArtifacts();
11    EList<Artifact> uniqueArtifacts = new BasicEList<Artifact>();
12    for(Artifact art: artifacts){
13        boolean isIn = false;
14        for(Artifact b: uniqueArtifacts)
15            if(this.sameArtifact(art, b)) isIn=true;
16        if(!isIn)
17            uniqueArtifacts.add(art);
18    }
19
20    /** 2. get all pairs of two artifacts */
21    EList<EList<Artifact>> pairs = new BasicEList<EList<Artifact>>();
22    for(int i = 0;i<uniqueArtifacts.size();i++){
23        for(int j = i+1;j<uniqueArtifacts.size();j++){
24            EList<Artifact> pair = new BasicEList<Artifact>();
25            pair.add(uniqueArtifacts.get(i));
26            pair.add(uniqueArtifacts.get(j));
27            pairs.add(pair);
28        }
29    }
30
31    /** 3. For each pair of artifacts length of artifacts chains between them is
32     * calculated. This calculation is done in both directions (i.e. such that
33     * both artifacts are once the first and once the second artifact). */
34    for(int k = 0; k< pairs.size();k++){
35        Artifact first = pairs.get(k).get(0);
36        Artifact second = pairs.get(k).get(1);
37        this.calculateLengthsOfActivityChainsForArtifactPair(first, second, swmamo,
38                  predecessorsPerActivity, successorsPerActivity, predecessorSetsPerActivity);
39        this.calculateLengthsOfActivityChainsForArtifactPair(second, first, swmamo,
40                  predecessorsPerActivity, successorsPerActivity, predecessorSetsPerActivity);
41    }
42
43    return;
44 }
```

Listing F.12: calculateLengthsOfActivityChains()

```

1 void calculateLengthsOfActivityChainsForArtifactPair(
2     Artifact first,
3     Artifact second,
4     Model swmamo,
5     EMap<Activity ,EList<Activity>> predecessorsPerActivity ,
6     EMap<Activity ,EList<Activity>> successorsPerActivity ,
7     EMap<Activity ,EList<EList<Activity>>> predecessorSetsPerActivity )
8 {
9     /** 1. retrieve activities that are relevant for the creation , manipuation ,
10      /* and/or consumption of the first and second artifact */
11     EList<Activity> accessFirst =
12         this.getActivitiesThatCreateConsumeOrManipulateArtifact(swmamo, first);
13     EList<Activity> consumeFirst =
14         this.getActivitiesThatConsumeArtifact(swmamo, first);
15     EList<Activity> outputSecond =
16         this.getActivitiesThatCreateOrManipulateArtifact(swmamo, second);
17
18     /** 2. Get all predecessor sets of the second artifact in order to identify
19      /* all activities that produce or manipulate second artifact. The predecessor
20      /* sets of these activities are relevant in the following. */
21     EList<EList<Activity>> artifactRelatedPredecessorSets =
22         new BasicEList<EList<Activity>>();
23     for(Activity creator: outputSecond){
24
25         /** 3. for each activity that creates or manipulates the second activity:
26         /* retrieve all predecessor sets */
27         EList<EList<Activity>> sets = predecessorSetsPerActivity.get(creator);
28         for(EList<Activity> set: sets){
29
30             /** 4. check whether predecessor set is relevant */
31             /** 4a. check whether predecessor set includes activities that consume
32             /* the first artifact */
33             boolean accessesFirstArtifact = false;
34             for(Activity a: set){
35                 if(consumeFirst.contains(a))
36                     accessesFirstArtifact = true;
37             }
38
39             /** 4b. Alternative the considered activity that creates or manipulates
40             /* the second artifact is consuming the first artifact directly. Note
41             /* that this needs to be handled separately , since an activity is not
42             /* part of its own predecessor set. */
43             if(consumeFirst.contains(creator))
44                 accessesFirstArtifact = true;
45
46             /** 5. if the predecessor set is relevant and not empty add a copy to
47             /* the selection */
48             if(accessesFirstArtifact)
49                 if(!set.isEmpty()){
50                     EList<Activity> copiedSet = new BasicEList<Activity >();
51                     copiedSet.addAll(set);
52                     artifactRelatedPredecessorSets.add(copiedSet);
53                 }
54             }
55         }
56     }
57     [...]
58 }
```

Listing F.13: calculateLengthsOfActivityChainsForArtifactPair() (part 1)

```

1 void calculateLengthsOfActivityChainsForArtifactPair(
2     Artifact first,
3     Artifact second,
4     Model swnamo,
5     EMap<Activity , EList<Activity>> predecessorsPerActivity ,
6     EMap<Activity , EList<Activity>> successorsPerActivity ,
7     EMap<Activity , EList<EList<Activity>>> predecessorSetsPerActivity )
8 {
9     [...]
10
11    /** 6. Not all activities within a relevant predecessor set might be relevant ,
12    /* too. Therefore, each predecessor set is filtered , such that only activities
13    /* remain that are used for a change of the first artifact or the propagation
14    /* of such a change to the second artifact. These are on the one hand
15    /* activities that manipulate or create the first activity or on the other
16    /* hand activities that are successors of an activities that manipulate or
17    /* create the first activity. */
18    EList<EList<Activity>> filteredArtifactRelatedPredecessorSets =
19        new BasicEList<EList<Activity>>();
20    for(EList<Activity> set : artifactRelatedPredecessorSets){
21        EList<Activity> filteredSet = new BasicEList<Activity>();
22        for(Activity a: set){
23            boolean isAccessingFirst = false;
24            boolean isSuccessorFirst = false;
25
26            /** activity consumes , manipulates , or creates first artifact */
27            if(accessFirst.contains(a))
28                isAccessingFirst = true;
29
30            /** activity is successor of an activity that consumes , manipulates ,
31            /* or creates first artifact */
32            for(Activity access: accessFirst){
33                EList<Activity> successors = successorsPerActivity.get(access);
34                if(successors.contains(a))
35                    isSuccessorFirst = true;
36            }
37
38            if(isAccessingFirst)
39                filteredSet.add(a);
40            else if(isSuccessorFirst)
41                filteredSet.add(a);
42        }
43        filteredArtifactRelatedPredecessorSets.add(filteredSet);
44    }
45
46    /** 7. Check whether an activities that creates or manipulates the second
47    /* artifact directly consumes the first artifact */
48    boolean directRelation = false;
49    for(Activity creator: outputSecond){
50        if(consumeFirst.contains(creator))
51            directRelation=true;
52    }
53
54    /** 8. if no connection between both artifacts can be identified ,
55    /* stop algorithm here */
56    if(filteredArtifactRelatedPredecessorSets.size() < 1)
57        if(!directRelation)
58            return;
59
60    [...]
61 }
```

Listing F.14: calculateLengthsOfActivityChainsForArtifactPair() (part 2)

Appendix F. CodeListings

```

1 void calculateLengthsOfActivityChainsForArtifactPair(
2     Artifact first,
3     Artifact second,
4     Model swmamo,
5     EMap<Activity ,EList<Activity>> predecessorsPerActivity ,
6     EMap<Activity ,EList<Activity>> successorsPerActivity ,
7     EMap<Activity ,EList<EList<Activity>>> predecessorSetsPerActivity )
8 {
9     [...]
10
11    /** 9. Identify minimum and maximum number of activities that have to be executed
12     * to change and propagate a change from first artifact to the second
13     * artifact. The size of an activity chain is thereby equal to the size of
14     * a filtered predecessor set + 1, since the filtered predecessor set
15     * includes exactly the activities that are sufficient and actually used to
16     * propagate a change from the first artifact to the second. Thereby, the
17     * last activity in the chain (i.e. the activity that manipulates or creates
18     * the second artifact) is not part of the predecessor set as described above.
19     * Therefore the lengths have to be increased by 1. The difference between
20     * minimum and maximum length results from the differences between the
21     * predecessor sets. Note that some first artifacts stem from outside the MDE
22     * setting and are only used within the setting. As a result, no activity for
23     * changing such an artifact can be counted. **/
24    int min = 0;
25    int max = 0;
26    int size;
27
28    /** 9a. initialize min and max with length of first filtered predecessor set. ***/
29    if(filteredArtifactRelatedPredecessorSets.size()>0) {
30        size = filteredArtifactRelatedPredecessorSets.get(0).size();
31        min = size;
32        max = size;
33
34        /** 9b. Iterate through filtered predecessor sets and update min and
35         * max accordingly. */
36        for(int i = 1; i<filteredArtifactRelatedPredecessorSets.size();i++){
37            EList<Activity> set = filteredArtifactRelatedPredecessorSets.get(i);
38            size = set.size();
39            if(size<min)
40                min = size;
41            if(size>max)
42                max = size;
43        }
44
45        /** 9c. increase min and max by one to represent activity that manipulates
46         * or created the second artifact */
47        min = min +1;
48        max = max +1;
49
50    } else if(directRelation){
51        /** 9d. Handle special case where no filtered predecessor set exists, but
52         * the activity that manipulates or creates the second artifact consumes
53         * the first artifact (which can happen when the first artifact stems
54         * from outside the MDE setting). In this case set min and max to 1. ***/
55        min = 1;
56        if(max==0)
57            max = 1;
58    }
59
60    /** 10. printout result */
61    System.out.print("from_artifact_" + first.getRoleName() + "_to_artifact_" +
62                     second.getRoleName() + ":" );
63    System.out.println(min + " -- " + max);
64
65    return;
66}
67

```

Listing F.15: calculateLengthsOfActivityChainsForArtifactPair() (part 3)

```

1 EList<Activity> getActivitiesThatCreateConsumeOrManipulateArtifact(
2     Model swmamo,
3     Artifact artifact)
4 {
5     EList<Activity> result = new BasicEList<Activity>();
6     /** 1. iterate all activities in the model and select relevant activities**/
7     EList<Activity> activities = swmamo.getActivity();
8     for(Activity act: activities){
9         boolean accessesart =false;
10        /** 1a. an activity is relevant if it consumes the considered artifact **/
11        EList<Artifact> consumedartifacts = this.getConsumedArtifacts(act, swmamo);
12        for(Artifact a : consumedartifacts)
13            if(this.sameArtifact(artifact, a))
14                accessesart=true;
15        /** 1b. and an activity is relevant if it produces the artifact under
16        /* consideration */
17        EList<Artifact> producedartifacts = this.getProducedArtifacts(act, swmamo);
18        for(Artifact a : producedartifacts)
19            if(this.sameArtifact(artifact, a))
20                accessesart=true;
21        if(accessesart)
22            result.add(act);
23    }
24    return result;
25 }
```

Listing F.16: getActivitiesThatCreateConsumeOrManipulateArtifact()

```

1 EList<Activity> getActivitiesThatConsumeArtifact(
2     Model swmamo,
3     Artifact artifact)
4 {
5     EList<Activity> result = new BasicEList<Activity>();
6     /** 1. iterate all activities in the model and select relevant activities **/
7     EList<Activity> activities = swmamo.getActivity();
8     for(Activity act: activities){
9         boolean accessesart =false;
10        /** 1a. an activity is relevant if it consumes the considered artifact **/
11        EList<Artifact> consumedartifacts = this.getConsumedArtifacts(act, swmamo);
12        for(Artifact a : consumedartifacts)
13            if(this.sameArtifact(artifact, a))
14                accessesart=true;
15        if(accessesart)
16            result.add(act);
17    }
18    return result;
19 }
```

Listing F.17: getActivitiesThatConsumeArtifact()

```

1 EList<Activity> getActivitiesThatCreateOrManipulateArtifact(
2     Model swmamo,
3     Artifact artifact)
4 {
5     EList<Activity> result = new BasicEList<Activity>();
6     /** 1. iterate all activities in the model and select relevant activities **/
7     EList<Activity> activities = swmamo.getActivity();
8     for(Activity act: activities){
9         boolean producesart =false;
10        /** 1a. an activity is relevant if it produces the considered artifact */
11        EList<Artifact> producedartifacts = this.getProducedArtifacts(act, swmamo);
12        for(Artifact a : producedartifacts)
13            if(this.sameArtifact(artifact, a))
14                producesart=true;
15        if(producesart)
16            result.add(act);
17    }
18    return result;
19 }
```

Listing F.18: getActivitiesThatCreateOrManipulateArtifact()

Appendix G.

Analysis Details for Changeability Patterns and MDE Traits

This Appendix includes the detailed results of the analyses performed in Chapter 7.

G.1. Pattern Occurrences for the 11 Case Studies

In this Section the identified occurrences of the proto-patterns from Section 7.1 are listed together with identified mitigating factors. Table G.1 and Figure G.1 summarize the pattern occurrences.

Table G.1.: Summary of identified matches for proto-patterns from Section 7.1 in the case studies from the first and third study (see Chapter 4).

	Subsequent Adjustment	Creation Dependence	De-facto	Split Manufacture	Manufacture	Anchor
BO	3		1			
BRF						
BW						
Oberon	5					1
SIW	3				2	
VC	3		1			
VCat	3				2	
Carmeq						
Cap1	1		1			
Cap2a	1					
Cap2b			4			
All	19		7		4	1
Average	1.72		0.63		0.36	0.09

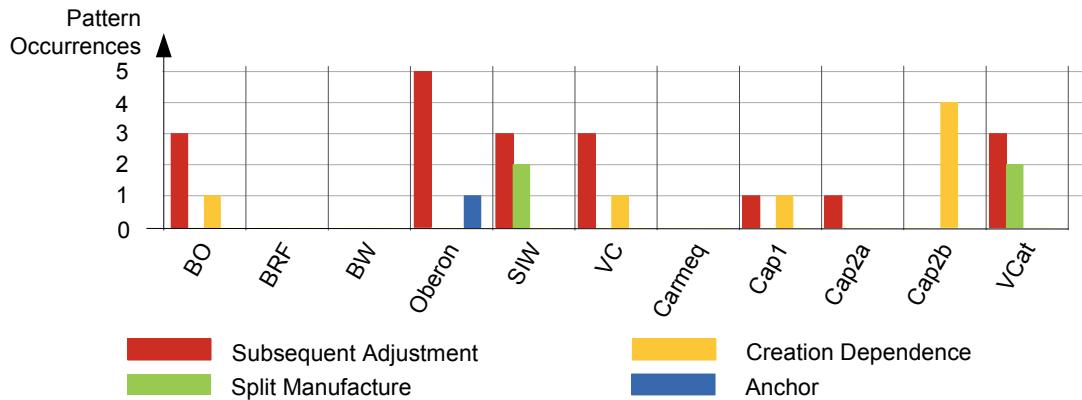


Figure G.1.: Occurrences of patterns on changeability concerns within the case studies.

Identified mitigating factors for *subsequent adjustment* in practice are:

- **No change:** 9 times it can be expected that the base artifact will not be changed. Thereby, 4 times the base artifact is a template.
- **Minimal loss:** 1 time the amount of content that can be lost is minimal.
- **No need to maintain:** 1 time there was no need to maintain the resulting artifacts through the whole life cycle.
- **Resolved by alternative:** 2 times there is an alternative activity that is used instead the second execution of the initial creation

- For 6 matches no mitigating factors could be identified. Instead it was reported that, e.g. developers use a copy and paste workaround to preserve manually created content.

Identified mitigating factors for *creation dependence* in practice are:

- **No amplification:** 5 times no changeability issues could be identified for the successor activities of the matched activity. Thus, no amplification is to be expected.
- For 2 match no mitigating factors could be identified.

List of matched pattern activities per MDE setting:

- BO
 - *creation dependence*:
 - * “*ApplyChecks*” (depending artifact: *ResultList*) (**No amplification**)
 - *subsequent adjustment*:
 - * “*Generate Code & Tables*” and “*ModelS&AMs*”
 - * “*Generate Code & Tables*” and “*Write Business Logic*”
 - * “*Project to BO Model*” and “*Adapt BusinessObject Model*” (**No change**)
- BRF – no matches identified
- BW – no matches identified
- Oberon
 - *subsequent adjustment*:
 - * “*Create DataModel*” and “*Bind to Business Object*” (**No change**)
 - * “*Create Floorplan*” and “*Bind to Business Object*” (**No change**)
 - * “*Create Floorplan*” and “*BindToQuery*” (**No change**)
 - * “*Create Floorplan*” and “*Manipulate Floorplan*” (**No change**)
 - * “*Create Floorplan*” and “*Add Event Handler*” (**No change**)
 - *anchor*:
 - * “*define change transaction*” and “*apply changes*”
- SIW
 - *split manufacture*:
 - * “*regenerateCode*” and “*mapToInternalSignature*”
 - * “*regenerateCode*” and “*AddCodeToSlot*”
 - *subsequent adjustment*:
 - * “*Generate Proxy*” and “*Regenerate Proxy*” (**Minimal loss**)
 - * “*GenerateCode*” and “*mapToInternalSignature*” (possible Base Artifacts: Template, Contextparameters) (**Resolved by alternative**: regenerateCode)
 - * “*GenerateCode*” and “*AddCodeToSlot*” (possible Base Artifacts: Template, Contextparameters) (**Resolved by alternative**: regenerateCode)
- VC
 - *creation dependence*:
 - * “*Generate process Context*” (can amplify occurrences of *subsequent adjustment* with “*Generate process Context*” as initial creation)
 - *subsequent adjustment*:
 - * “*Create Service Component*” and “*Define Data*” (**No need to maintain**)
 - * “*Generate process Context*” and “*Import Data Service*”
 - * “*Generate process Context*” and “*Create Service Component*”
- VCat
 - *split manufacture*:
 - * “*install extension*” and “*customize extension*”
 - * “*install extension*” and “*update extension*”
 - *subsequent adjustment*:
 - * “*acquire zip-file*” and “*update extension*” (**No change**)
 - * “*acquire zip-file*” and “*customize extension*” (**No change**)
 - * “*run installation script*” and “*customize extension*” (**No change**)
- Carmeq – no matches identified
- Cap1
 - *subsequent adjustment*:
 - * “*Create Constructionmodel*” and “*Manipulate Constructionmodel*”
 - *creation dependence*:
 - * “*Create Constructionmodel*”
- Cap2a

- *subsequent adjustment*:
 - * “Merge Parts of Model” and “Manipulate Result2”
- Cap2b
 - *creation dependence*:
 - * “Generate Result7” (**No amplification**)
 - * “Generate Result8” (**No amplification**)
 - * “Word Import Result2” (**No amplification**)
 - * “Word Import Result3” (**No amplification**)

G.2. Manual Information Propagation within the 11 Case Studies

Following the matches of pattern manually transformed artifact pair are listed. Note that some artifact pairs are affected by multiple activities. Table G.2 and Figure G.2 summarize the number of manually transformed artifacts pairs per case study.

Table G.2.: Summary of identified matches for pattern *manual information propagation* in the case studies from the first and third study (see Chapter 4).

	# Activities that match pattern manual information propagation	# manually transformed artifact pairs
BO	5	7
BRF	3	4
BW	2	3
Oberon	2	2
SIW	1	2
VC	0	0
VCat	3	4
Carmeq	7	10
Cap1	5	9
Cap2a	1	1
Cap2b	0	0
All	29	42
Average	2.63	3.82

List of activities matched to *manual information propagation* per MDE setting:

- BO (5 activities, 7 manually transformed artifact pairs)
 - “Model BusinessObject (BO)”: BO Model - Integration Scenario
 - “Model BusinessObject (BO)”: BO Model - BO Specification
 - “Analyse Requirements”: BO Specification - Requirements
 - “Transform to ESR Model”: ESR BO Model - BO Model
 - “Extend Template”: BO Template - BO Specification
 - “Extend Template”: BO Template - Integration Scenario
 - “Create TestCase”: Integration Scenario - TestCase
- BRF (3 activities, 4 manually transformed artifact pairs)
 - “TransformToBRF+Rules”: DecisionTable - Unstructured Rules
 - “TransformToBRF+Rules”: Rule - Unstructured Rules
 - “ChangeRuleSet”: Ruleset - Rule
 - “Create BRSs 4”: BusinessRuleService - Unstructured Rules
- BW (2 activities, 3 manually transformed artifact pairs)
 - “Model Container”: Container - InfoObject
 - “Model Transformation”: Transformation Model - InfoSource
 - “Model Transformation”: Transformation Model - Container
- Oberon (2 activities, 2 manually transformed artifact pairs)
 - “Define Change Transaction”: Change Transaction - Floorplan
 - “Manipulate Floorplan”: Floorplan - Anchor
- SIW (1 activities, 2 manually transformed artifact pairs)

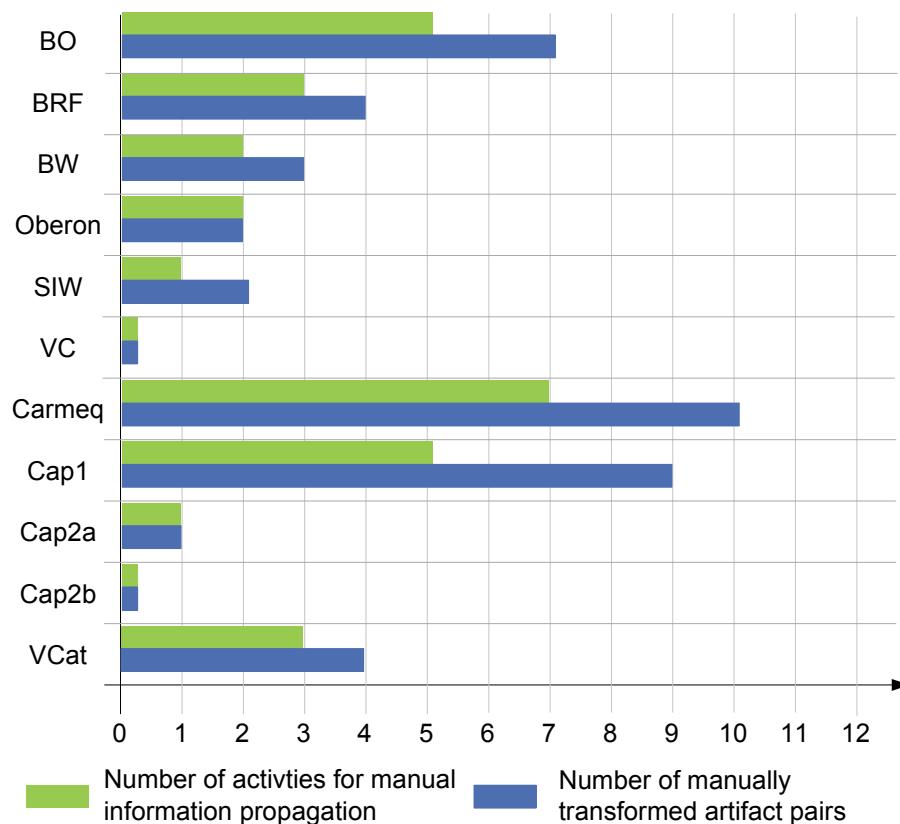


Figure G.2.: Manual information propagation and manually transformed artifact pairs within the case studies.

- “*map to internal signature*”: MappingSlot - BOR/Bapi Internal Signature
- “*map to internal signature*”: MappingSlot - Service Signatur
- VC (0 activities, 0 manually transformed artifact pairs)
- VCat (3 activities, 4 manually transformed artifact pairs)
 - “*create new extension*”: empty extension template - local extension
 - “*customize extension*”: local extension - TYPO3 database
 - “*customize extension*”: local extension examples - local extension
 - “*create template*”: TYPO3 template - design specification
- Carmeq (7 activities, 10 manually transformed artifact pairs)
 - “*Create Standard Document*” SW Specification Template - AUTOSAR Standard Document Word
 - “*Create Standard Document*” Concept Document - AUTOSAR Standard Document Word
 - “*Manipulate Standard Document*” AUTOSAR Standard Document Word - Concept Document
 - “*Model*” Basic SW Model - Graphics
 - “*Model Change*” Graphics - Basic SW Model
 - “*Model*” Basic SW Model - Concept Document
 - “*Model Change*” Concept Document - Basic SW Model
 - “*Model*” Basic SW Model - Table
 - “*Model Change*” Table - Basic SW Model
 - “*Model*” Basic SW Model - Workingcopy of BSW Model
 - “*Model Change*” Workingcopy of BSW Model - Basic SW Model
 - “*Create Concept Document*” Concept Document Template - Concept Document
 - “*Publish Standard Document*” AUTOSAR Standard Document PDF - AUTOSAR Standard Document Word
 - “*Review 1: Tables*” Released Table HTML - Table HTML
 - “*Review 2 Figures*” Figure PNG - Released Figure PNG
- Cap1 (5 activities, 9 manually transformed artifact pairs)
 - “*Write Code*”: Handwritten Code - UI-Prototype
 - “*Write Code*”: Handwritten Code - Construction Model

- “Manipulate Constructionmodel”: Construction Model - Specification model
- “Create Specification Model”: UI-Specification - Specification model
- “Create Specification”: Specification - Feature Lists
- “Create Specification”: Specification - Specification model
- “Create Specification”: Specification - GUI Widget Lists
- “Manipulate UI-Prototype”: Code - UI-Specification
- “Merge Versions of Specification Model”: Specification model - Specification model to be merged
- Cap2a (1 activities, 1 manually transformed artifact pair)
 - “Manipulate Model”: Partial Model - Mail
- Cap2b (0 activities, 0 manually transformed artifact pairs)

G.3. Complexity of Activity Chains of the 11 Case Studies

This Section lists the identified lengths of activity chains. Thereby, Figure G.3 summarizes the collected data.

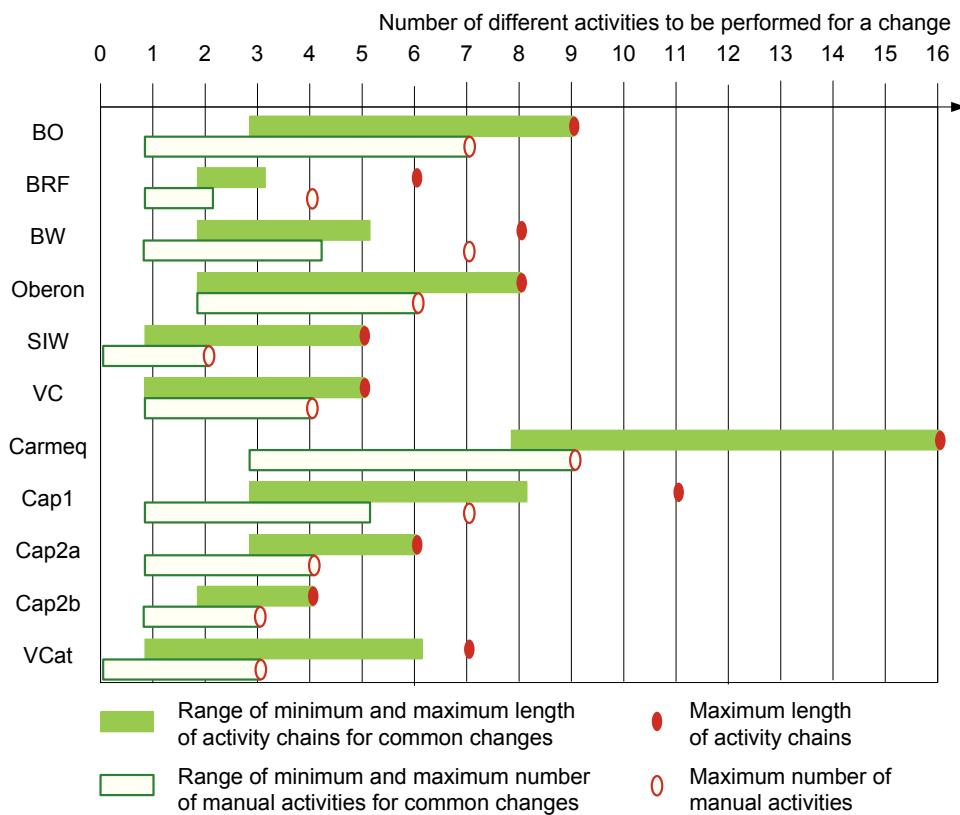


Figure G.3.: Summary of lengths of activity chains

List of lengths of activity chains measured per MDE setting:

- BO
 - *Change artifacts*: BO Model, Requirements
 - *Target artifacts*: S&AM Model, Tables, BO Code, DB Schema
 - *Changes*:
 - * from artifact Requirements to artifact BO Code: 5 - 9 (Manual activities 4 - 7)
 - * from artifact Requirements to artifact Tables: 5 - 7 (Manual activities 4 - 5)
 - * from artifact Requirements to artifact DB Schema: 5 - 7 (Manual activities 4 - 5)
 - * from artifact Requirements to artifact S&AM Model: 6 - 9 (Manual activities 5 - 7)
 - * from artifact BO Model to artifact BO Code: 3 - 6 (Manual activities 1 - 5)
 - * from artifact BO Model to artifact Tables: 3 - 4 (Manual activities 1 - 3)
 - * from artifact BO Model to artifact DB Schema: 3 - 4 (Manual activities 1 - 3)
 - * from artifact BO Model to artifact S&AM Model: 4 - 6 (Manual activities 2 - 5)
 - *Common changes*: all listed

- * Range: 3 - 9
 - * Range manual activities: 1 - 7
 - Maximum length of activity chains: 9
 - Maximum number of manual activities: 7
- BRF
 - Change artifacts: BusinessRuleService, SAPProcess, Ruleset, DecisionTable, Rule
 - Target artifacts: DBEntry, RuleClass
 - Changes:
 - * from artifact SAPProcess to artifact RuleClass: 4 - 6 (Manual activities 2 - 4)
 - * from artifact SAPProcess to artifact DBEntry: 4 - 6 (Manual activities 2 - 4)
 - * from artifact BusinessRuleService to artifact RuleClass: 2 - 5 (Manual activities 1 - 3)
 - * from artifact BusinessRuleService to artifact DBEntry: 2 - 5 (Manual activities 1 - 3)
 - * from artifact Ruleset to artifact RuleClass: 2 - 3 (Manual activities 1 - 2)
 - * from artifact Ruleset to artifact DBEntry: 2 - 3 (Manual activities 1 - 2)
 - * from artifact DecisionTable to artifact RuleClass: 2 - 2 (Manual activities 1 - 1)
 - * from artifact DecisionTable to artifact DBEntry: 2 - 2 (Manual activities 1 - 1)
 - * from artifact Rule to artifact RuleClass: 2 - 2 (Manual activities 1 - 1)
 - * from artifact Rule to artifact DBEntry: 2 - 2 (Manual activities 1 - 1)
 - Common changes: Change introduction in artifacts Ruleset, Rule, or Decision Table
 - * Range: 2 - 3
 - * Range manual activities: 1 - 2
 - Maximum length of activity chains: 6
 - Maximum number of manual activities: 4
- BW
 - Change artifacts: InfoObject, Container, InfoSource, Layout, Query
 - Target artifacts: Layout, HTML Report
 - Changes:
 - * from artifact InfoObject to artifact Layout: 5 - 5 (Manual activities 5 - 5)
 - * from artifact InfoObject to artifact HTML Report: 6 - 8 (Manual activities 5 - 7)
 - * from artifact Container to artifact Layout: 4 - 4 (Manual activities 4 - 4)
 - * from artifact Container to artifact HTML Report: 5 - 7 (Manual activities 4 - 6)
 - * from artifact InfoSource to artifact HTML Report: 3 - 5 (Manual activities 2 - 4)
 - * from artifact Query to artifact Layout: 2 - 2 (Manual activities 2 - 2)
 - * from artifact Query to artifact HTML Report: 3 - 3 (Manual activities 2 - 2)
 - * from artifact Layout to artifact HTML Report: 2 - 2 (Manual activities 1 - 1)
 - Common changes: Addition of new artifacts of role InfoSource and for changes in Layout and Query
 - * Range: 2 - 5
 - * Range manual activities: 1 - 4
 - Maximum length of activity chains: 8
 - Maximum number of manual activities: 7
- Oberon
 - Change artifacts: Floorplan Template, Floorplan, Change Transaction
 - Target artifacts: Floorplan, Configuration
 - Changes:
 - * from artifact Floorplan Template to artifact Change Transaction: 2 - 7 (Manual activities 1 - 5)
 - * from artifact Floorplan Template to artifact Configuration: 3 - 8 (Manual activities 2 - 6)
 - * from artifact Floorplan to artifact Configuration: 3 - 8 (Manual activities 2 - 6)
 - * from artifact Change Transaction to artifact Configuration: 2 - 2 (Manual activities 2 - 2)
 - Common changes: changes are most probably introduced on artifacts Floorplan and Change Transaction
 - * Range: 2 - 8
 - * Range manual activities: 2 - 6
 - Maximum length of activity chains: 8
 - Maximum number of manual activities: 6
- SIW
 - Change artifacts: Service Signatur, Configuration, BOR/Bapi Internal Signature
 - Target artifacts: Proxy Class - Method, ABAPObject - Slot, Mapping Class - Mapping Slot
 - Changes:
 - * from artifact Configuration to artifact ABAPObject: 3 - 3 (Manual activities 1 - 1)

- * from artifact Configuration to artifact Slot: 3 - 4 (Manual activities 1 - 2)
- * from artifact Configuration to artifact MappingSlot: 3 - 4 (Manual activities 1 - 2)
- * from artifact Configuration to artifact MappingClass: 3 - 3 (Manual activities 1 - 1)
- * from artifact Configuration to artifact Method: 3 - 3 (Manual activities 1 - 1)
- * from artifact Service Signatur to artifact ABAPObject: 4 - 4 (Manual activities 1 - 1)
- * from artifact Service Signatur to artifact MappingSlot: 4 - 5 (Manual activities 1 - 2)
- * from artifact Service Signatur to artifact Slot: 4 - 5 (Manual activities 1 - 2)
- * from artifact Service Signatur to artifact MappingClass: 4 - 4 (Manual activities 1 - 1)
- * from artifact Service Signatur to artifact Method: 4 - 4 (Manual activities 1 - 1)
- * from artifact Service Signatur to artifact ProxyClass: 1 - 1 (Manual activities 0 - 0)
- * from artifact BOR/Bapi Internal Signature to artifact MappingSlot: 1 - 1 (Manual activities 1 - 1)
- *Common changes:* Changes are most probably introduced in Service Signature
 - * Range: 1 - 5
 - * Range manual activities: 0 - 2
- *Maximum length of activity chains:* 5
- *Maximum number of manual activities:* 2
- VC
 - *Change artifacts:* VC Model, Process Model, Data Service
 - *Target artifacts:* VC Model
 - *Changes:*
 - * from artifact VC Model to artifact VC Model: 1 (Manual activities 1)
 - * from artifact Data Service to artifact VC Model: 1 - 5 (Manual activities 1 - 4)
 - * from artifact Process Model to artifact VC Model: 2 - 5 (Manual activities 1 - 4)
 - *Common changes:* all listed
 - * Range: 1 - 5
 - * Range manual activities: 1 - 4
 - *Maximum length of activity chains:* 5
 - *Maximum number of manual activities:* 4
- VCat
 - *Change artifacts:* local installation file, local extension, TYPO3 template, design specification
 - *Target artifacts:* TYPO3 template, local extension, TYPO3 database
 - *Changes:*
 - * from artifact local installation file to artifact TYPO3 database: 2 - 6 (Manual activities 0 - 2)
 - * from artifact local installation file to artifact local extension: 3 - 6 (Manual activities 0 - 2)
 - * from artifact local installation file to artifact TYPO3 template: 3 - 7 (Manual activities 1 - 2)
 - * from artifact local installation file to artifact database table: 3 - 6 (Manual activities 0 - 1)
 - * from artifact local extension to artifact TYPO3 database: 2 - 5 (Manual activities 0 - 3)
 - * from artifact local extension to artifact TYPO3 template: 2 - 6 (Manual activities 1 - 3)
 - * from artifact local extension to artifact database table: 2 - 5 (Manual activities 0 - 2)
 - * from artifact design specification to artifact TYPO3 template: 1 - 1 (Manual activities 1 - 1)
 - *Common changes:* changes are introduced in local extension or design specification
 - * Range: 1 - 6
 - * Range manual activities: 0 - 3
 - *Maximum length of activity chains:* 7
 - *Maximum number of manual activities:* 3
- Carmeq
 - *Change artifacts:* Concept Document, Basic SW Model, AUTOSAR Standard Document Word
 - *Target artifacts:* AUTOSAR Standard Document PDF
 - *Changes:*
 - * from artifact Concept Document to artifact AUTOSAR Standard Document PDF: 10 - 16 (Manual activities 5 - 9)
 - * from artifact Basic SW Model to artifact AUTOSAR Standard Document PDF: 8 - 13 (Manual activities 3 - 6)
 - * from artifact AUTOSAR Standard Document Word to artifact AUTOSAR Standard Document PDF: 4 - 7 (Manual activities 2 - 3)
 - *Common changes:* change introduction at Concept Document and Basic SW Model (for changes that concern the figures and tables)
 - * Range: 8 - 16
 - * Range manual activities: 3 - 9

- Maximum length of activity chains: 16
- Maximum number of manual activities: 9
- Cap1
 - *Change artifacts*: UI-Specification, Specification model, Construction model
 - *Target artifacts*: Specification, UI-Prototype, Generated Code, Handwritten code
 - *Changes*:
 - * from artifact UI-Specification to artifact UI-Prototype: 2 - 2 (Manual activities 1 - 1)
 - * from artifact UI-Specification to artifact Specification: 4 - 7 (Manual activities 3 - 5)
 - * from artifact UI-Specification to artifact Generated Code: 4 - 8 (Manual activities 2 - 5)
 - * from artifact UI-Specification to artifact Handwritten Code: 8 - 11 (Manual activities 5 - 7)
 - * from artifact Specification model to artifact Specification: 3 - 6 (Manual activities 2 - 4)
 - * from artifact Specification model to artifact Generated Code: 3 - 7 (Manual activities 1 - 4)
 - * from artifact Specification model to artifact Handwritten Code: 5 - 8 (Manual activities 3 - 5)
 - * from artifact Specification model to be merged to artifact Specification: 4 - 5 (Manual activities 2 - 3)
 - * from artifact Specification model to be merged to artifact Generated Code: 4 - 6 (Manual activities 1 - 3)
 - * from artifact Specification model to be merged to artifact Handwritten Code: 6 - 7 (Manual activities 3 - 4)
 - * from artifact Construction Model to artifact Generated Code: 2 - 3 (Manual activities 0 - 1)
 - * from artifact Construction Model to artifact Handwritten Code: 4 - 4 (Manual activities 2 - 2)
 - *Common changes*: Introduction of changes at Specification model
 - * Range: 3 - 8
 - * Range manual activities: 1 - 5
 - Maximum length of activity chains: 11
 - Maximum number of manual activities: 7
- Cap2a
 - *Change artifacts*: Plain Text Part, Partial Model, Configuration
 - *Target artifacts*: Result1, Result2
 - *Changes*:
 - * from artifact Plain Text Part to artifact Result1: 2 - 5 (Manual activities 1 - 3)
 - * from artifact Plain Text Part to artifact Result2: 2 - 6 (Manual activities 1 - 4)
 - * from artifact Partial Model to artifact Result1: 3 - 5 (Manual activities 2 - 3)
 - * from artifact Partial Model to artifact Result2: 3 - 6 (Manual activities 2 - 4)
 - * from artifact Configuration to artifact Result1: 2 - 2 (Manual activities 1 - 1)
 - * from artifact Configuration to artifact Result2: 2 - 3 (Manual activities 1 - 2)
 - *Common changes*: changes in Partial Model
 - * Range: 3 - 6
 - * Range manual activities: 2 - 4
 - Maximum length of activity chains: 6
 - Maximum number of manual activities: 4
- Cap2b
 - *Change artifacts*: Partial Engineering Model, Partial Model
 - *Target artifacts*: Result8, Placeholder Result3, Placeholder Result2, Result2, Result3
 - *Changes*:
 - * from artifact Partial Engineering Model to artifact Result8: 2 - 3 (Manual activities 1 - 2)
 - * from artifact Partial Engineering Model to artifact Result2: 2 - 4 (Manual activities 1 - 3)
 - * from artifact Partial Engineering Model to artifact Placeholder Result2: 2 - 3 (Manual activities 1 - 2)
 - * from artifact Partial Engineering Model to artifact Result3: 2 - 4 (Manual activities 1 - 3)
 - * from artifact Partial Engineering Model to artifact Placeholder Result3: 2 - 3 (Manual activities 1 - 2)
 - * from artifact Partial Model to artifact Result8: 1 - 2 (Manual activities 0 - 1)
 - * from artifact Partial Model to artifact Result2: 1 - 3 (Manual activities 0 - 2)
 - * from artifact Partial Model to artifact Placeholder Result2: 1 - 2 (Manual activities 0 - 1)
 - * from artifact Partial Model to artifact Result3: 1 - 3 (Manual activities 0 - 2)
 - * from artifact Partial Model to artifact Placeholder Result3: 1 - 2 (Manual activities 0 - 1)
 - *Common changes*: change introduced at Partial Engineering Model
 - * Range: 2 - 4
 - * Range manual activities: 1 - 3
 - Maximum length of activity chains: 4
 - Maximum number of manual activities: 3

G.4. Phases within the 11 Case Studies

In the following *phases* and *synchronization points* for the different case studies are listed. Tables G.3 and G.4 as well as Figure G.4 summarize the identified local foci, local synchronization points, global synchronization points and phases.

Table G.3.: Summary of *synchronization points* in the case studies from the first and third study (see Chapter 4).

	# global synchronization points	# synchronization points in main focus	# global phases	# local phases in main focus
BO	0	6	1	3
BRF	1	1	2	2
BW	1	3	1	1
Oberon	1	3	1	1
SIW	0	1	1	2
VC	0	0	1	1
VCat	0	2	1	1
Carmeque	2	3	2	3
Cap1	1	6	1	4
Cap2a	3	3	2	2
Cap2b	2	4	1	2
All	11	32	14	22
Average	1	2.9	1.27	2

Table G.4.: Summary of *synchronization points* local foci of the case studies from the first and third study (see Chapter 4).

	local focus	# global synchronization points	# local synchronization points	# local phase per focus
BO	a	0	4	1
	b	0	6	3
BRF	a	1	2	2
BW	a	1	2	1
	b	1	5	1
	c	1	2	1
Oberon	a	1	2	1
SIW	a	0	1	2
	b	0	3	1
VCat	a	0	2	1
Carmeque	a	2	1	2
	b	2	1	3
	c	2	5	4
	d	2	5	4
Cap1	a	1	3	2
	b	1	3	3
	c	1	5	4
Cap2b	a	2	2	2
	b	2	2	2
All		11	56	40
Average		1	2.94	2.1

List of synchronization points identified per MDE setting:

- BO

- *Extensive manual activities (potential miniphases)*: Define Requirements with Customer, Model BusinessObject (BO), Adapt BusinessObject Model, ModelS&AMs, Write Business Logic,
- *Global synchronization points*: 0
- *Global Phases*: 1 (18 activities)
- *Miniphases*: 0
- *Local focuses with local synchronization points*
 - * **a** Preparation of new (extended) templates for the development
 - *Local synchronization points*: Extend Template , Analyse Requirements , Define Requirements with Customer, Project to BO Model
 - *Combined local focus*: Extend Template , Analyse Requirements, Define Requirements with Customer, Project to BO Model
 - *Size of local focus*: 4 activities out of 18 activities
 - *Phases*: 0
 - *Miniphases*: 1
 - * **b** Main development path
 - *Local synchronization points*: Generate Code & Tables , Transform to ESR Model , Analyse Requirement, Define Requirements with Customer, Adapt BusinessObject Model, Test Local Interaction
 - *Combined local focus*: Generate Code & Tables , Transform to ESR Model , Analyse Requirement, Define Requirements with Customer, Model BusinessObject (BO) , Extend Template , Adapt BusinessObject Model , Project to BO Model, ModelS&AMs, Test Local Interaction, Write Business Logic
 - *Size of local focus*: 11 activities out of 18 activities
 - *Phases*: 2
 - *Miniphases*: 1 (Define Requirements with Customer) – Adapt BusinessObject Model is combined with first phase
 - *After: Define Requirements with Customer; Analyse Requirements – before: Adapt BusinessObject Model; Transform to ESR Model; Generate Code & Tables*: Model BusinessObject (BO) , Extend Template, Project to BO Model
 - *After Generate Code & Tables – before Test Local Interaction*: ModelS&AMs, Write Business Logic
- BRF
 - *Extensive manual activities (potential miniphases)*: ChangeRuleSet, TransformToBRF+Rules, Implement Process 1, DefineUnstructuredRules
 - *Global synchronization points*: 1
 - * TieBRSToCode 1
 - *Global Phases*: 2 (together 13 activities)
 - *Miniphases*: 0
 - * *Before TieBRSToCode 1*: Implement Process 1, Create BRS 2, Create BRS 3, DefineUnstructuredRules, Create BRSs 4
 - * *After all*: DefineRuleset, ChangeRuleSet, ConvertToDBEntriesAndRuleClass, Simulation, GapAnalysis, OverlapContradictionAnalysis, TransformToBRF+Rules
 - *Local focuses with local synchronization points*
 - * **a** Preparation of process for configuration
 - *Local synchronization points*: DefineRuleset, TieBRSToCode 1, ChangeRuleSet
 - *Combined local focus*: Implement Process 1, Create BRS 2, Create BRS 3, DefineUnstructuredRules, Create BRSs 4, TieBRSToCode 1, ChangeRuleSet, DefineRuleset
 - *Size of local focus*: 8 activities out of 13 activities
 - *Phases*: 1
 - *Miniphases*: 1 (ChangeRuleSet)
 - *Before all*: Implement Process 1, Create BRS 2, Create BRS 3, DefineUnstructuredRules, Create BRSs 4
- BW
 - *Extensive manual activities (potential miniphases)*: Model Container, Model Data, Model Transformation, Program Transformation, DefineContainerSubset, DefineQuery, Define Layout, CreateInfoSource, DefineScheduling
 - *Global synchronization points*: 1
 - * Interpret
 - *Global Phases*: 1 (10 activities)
 - *Miniphases*: 0
 - * *Before Interpret*: Model Container, Model Data, CreateInfoSource, Model Transformation, Program Transformation, DefineContainerSubset, DefineScheduling, DefineQuery, Define Layout
 - *Local focuses with local synchronization points*:

- * **a** Development of layout of reports including transformation of data
 - *Local synchronization points:* Model Container, Model Data, Interprete
 - *Combined local focus:* Model Container, Model Data, Model Transformation, Program Transformation, DefineContainerSubset, DefineQuery, Define Layout, Interprete
 - *Size of local focus:* 8 activities out of 10 activities
 - *Phases:* 1
 - *Miniphases:* 0 (Model Container and Model Data are combined with the phase)
 - *After: Model Container; Model Data – before: Interprete:* Model Transformation, Program Transformation, DefineContainerSubset, DefineQuery, Define Layout
- * **b** Development of layout of reports (local to development of layout of reports including transformation of data)
 - *Local synchronization points:* Define Layout, DefineQuery, DefineContainerSubset, Model Container, Model Data, Interprete
 - *Combined local focus:* Define Layout, DefineQuery, DefineContainerSubset, Model Container, Model Data, Interprete
 - *Size of local focus:* 6 activities out of 10 activities
 - *Phases:* 0
 - *Miniphases:* 1 phase (results from combination of 5 miniphases)
- * **c** Configuration of data retrieval
 - *Local synchronization points:* DefineScheduling, CreateInfoSource, Interprete
 - *Combined local focus:* DefineScheduling, CreateInfoSource, Interprete
 - *Size of local focus:* 3 activities out of 10 activities
 - *Phases:* 0
 - *Miniphases:* 1 phase (results from combination of 2 miniphases)
- Oberon
 - *Extensive manual activities (potential miniphases):* Manipulate Floorplan, Define Change Transaction
 - *Global synchronization points:* 1
 - * Create Floorplan
 - *Global Phases:* 1 (9 activities)
 - *Miniphases:* 0
 - * *After Create Floorplan:* Create DataModel, BindToQuery, Bind to Business Object, Add Event Handler, RegisterOBNReference, Define Change Transaction, Change Configuration, Manipulate Floorplan
 - *Local focuses with local synchronization points:*
 - * **a** Main path
 - *Local synchronization points:* Define Change Transaction, Create Floorplan, Change Configuration
 - *Combined local focus:* Define Change Transaction, Create Floorplan, Create DataModel, BindToQuery, Bind to Business Object, Add Event Handler, Manipulate Floorplan, Change Configuration
 - *Size of local focus:* 8 activities out of 9 activities
 - *Phases:* 1
 - *Miniphases:* 0 (Define Change Transaction is combined with the phase)
 - *After: Create Floorplan – before: Define Change Transaction; Change Configuration:* Create DataModel, BindToQuery, Bind to Business Object, Add Event Handler, Manipulate Floorplan
- SIW
 - *Extensive manual activities (potential miniphases):* AddCodeToSlot, mapToInternalSignature
 - *Global synchronization points:* 0
 - *Global Phases:* 1 (8 activities)
 - *Miniphases:* 0
 - *Local focuses with local synchronization points:*
 - * **a** Main path
 - *Local synchronization points:* GenerateCode
 - *Combined local focus:* Create SIW Project, Set Parameters, Generate Proxy, AddCodeToSlot, mapToInternalSignature, GenerateCode
 - *Size of local focus:* 6 activities out of 8 activities
 - *Phases:* 2
 - *Miniphases:* 0
 - *Before: GenerateCode:* Create SIW Project, Set Parameters, Generate Proxy
 - *After: GenerateCode:* AddCodeToSlot, mapToInternalSignature
 - * **b** Local to main path: preparation of project
 - *Local synchronization points:* GenerateCode, Set Parameters, Create SIW Project

- *Combined local focus:* Create SIW Project, AddCodeToSlot, mapToInternalSignature, GenerateCode, Set Parameters
 - *Size of local focus:* 5 activities out of 8 activities
 - *Phases:* 1
 - *Miniphases:* 0
 - *After:* AddCodeToSlot, mapToInternalSignature
- VC
 - *Global synchronization points:* 0
 - *Global Phases:* 1 (10 activities)
 - *Miniphases:* 0
 - *Candidates: Calculated local synchronization points:* 0
 - *Extensive manual activities (potential miniphases):* ManipulateVCModel, Create Dummy Data, Create Service Component, Define Data, Model Process, Import Data Service
- VCat
 - *Extensive manual activities (potential miniphases):* Create Template, Customize Extension, Perform UI Test, Create New Extension
 - *Global synchronization points:* 0
 - *Global Phases:* 1 (10 activities)
 - *Miniphases:* 0
 - *Local focuses with local synchronization points:*
 - * **a** Main path
 - *Local synchronization points:* run installation script, download files of tested TYPO3 version
 - *Combined local focus:* install extension, perform UI-Test, customize extension, install configuration extension, create template, download files of tested TYPO3 version, run installation script
 - *Size of local focus:* 7 activities out of 10 activities
 - *Phases:* 1
 - *Miniphases:* 0
 - *After all:* install extension, perform UI-Test, customize extension, install configuration extension, create template
- Carmeq
 - *Extensive manual activities (potential miniphases):* Create Concept Document, Model, Model Change, Manipulate Standard Document, Review 1: Tables, Review 2 Figures, Review 3 (Integrated Figures and Tables), Review 4 (Document Text)
 - *Global synchronization points:* 2
 - * Create Bugzilla Management Tasks
 - * Create Concept Document
 - *Global Phases:* 1 (18 activities)
 - *Miniphases:* 1 (Create Concept Document)
 - * *After all:* Model Change, Model, Generate Figures and Tables, Table-Diff (SVN), Figure-Diff (Beyond Compare), Review 1: Tables, Review 2 Figures, Create Standard Document, Manipulate Standard Document, Integrate Table To Word (Using Macro), Update Tables, Review 3 (Integrated Figures and Tables), Review 4 (Document Text), Publish Standard Document, Integrate Figure To Word (Using Macro), Update Figures
 - *Local focuses with Local synchronization points::*
 - * **a** work on standard document without figures and tables
 - *Local synchronization points:* Create Standard Document, Create Bugzilla Management Tasks, Create Concept Document,
 - *Combined local focus:* Manipulate Standard Document, Integrate Table To Word (Using Macro), Update Tables, Review 3 (Integrated Figures and Tables), Review 4 (Document Text), Publish Standard Document, Integrate Figure To Word (Using Macro), Update Figures, Create Concept Document, Create Standard Document, Create Bugzilla Management Tasks
 - *Size of local focus:* 11 activities out of 18 activities
 - *Phases:* 1
 - *Miniphases:* 1 (Create Concept Document)
 - *After all:* Manipulate Standard Document, Integrate Table To Word (Using Macro), Update Tables, Review 3 (Integrated Figures and Tables), Review 4 (Document Text), Publish Standard Document, Integrate Figure To Word (Using Macro), Update Figures
 - * **b** Main Path
 - *Local synchronization points:* Generate Figures and Tables, Create Bugzilla Management Tasks, Create Concept Document

- *Combined local focus:* Generate Figures and Tables, Create Bugzilla Management Tasks, Create Concept Document, Model Change, Model, Table-Diff (SVN), Figure-Diff (Beyond-Compare), Review 1: Tables, Review 2 Figures, Integrate Table To Word (Using Macro), Update Tables, Review 3 (Integrated Figures and Tables), Publish Standard Document, Integrate Figure To Word (Using Macro), Update Figures
- *Size of local focus:* 15 activities out of 18 activities
- *Phases:* 2
- *Miniphases:* 1 (Create Concept Document)
- *After:* Create Bugzilla Management Tasks; Create Concept Document – before: Generate Figures and Tables: Model Change, Model
- *After: Generate Figures and Tables:* Table-Diff (SVN), Figure-Diff (Beyond-Compare), Review 1: Tables, Review 2 Figures, Integrate Table To Word (Using Macro), Update Tables, Review 3 (Integrated Figures and Tables), Publish Standard Document, Integrate Figure To Word (Using Macro), Update Figures
- * **c Creation of Tables**
 - *Local synchronization points:* Review 1: Tables, Table-Diff (SVN), Generate Figures and Tables, Create Bugzilla Management Tasks, Create Concept Document, Integrate Table To Word (Using Macro), Update Tables
 - *Combined local focus:* Review 1: Tables, Table-Diff (SVN), Generate Figures and Tables, Create Bugzilla Management Tasks, Create Concept Document, Model Change, Model, Integrate Table To Word (Using Macro), Update Tables, Review 3 (Integrated Figures and Tables), Publish Standard Document
 - *Size of local focus:* 11 activities out of 18 activities
 - *Phases:* 2
 - *Miniphases:* 2 (Create Concept Document, Review 1: Tables)
 - *After: Create Bugzilla Management Tasks; Create Concept Document – before: Generate Figures and Tables:* Model Change, Model
 - *After all:* Review 3 (Integrated Figures and Tables), Publish Standard Document
- * **d Creation of Figures**
 - *Local synchronization points:* Review 2 Figures, Figure-Diff (Beyond-Compare), Generate Figures and Tables, Create Bugzilla Management Tasks, Create Concept Document, Integrate Figure To Word (Using Macro), Update Figures
 - *Combined local focus:* Review 2 Figures, Figure-Diff (Beyond-Compare), Generate Figures and Tables, Create Bugzilla Management Tasks, Create Concept Document, Model Change, Model, Integrate Figure To Word (Using Macro), Update Figures, Review 3 (Integrated Figures and Tables), Publish Standard Document
 - *Size of local focus:* 11 activities out of 18 activities
 - *Phases:* 2
 - *Miniphases:* 2 (Create Concept Document, Review 2 Figures)
 - *After: Create Bugzilla Management Tasks; Create Concept Document – before: Generate Figures and Tables* Model Change , Model
 - *After all:* Review 3 (Integrated Figures and Tables), Publish Standard Document, Integrate Figure To Word (Using Macro), Update Figures
- Cap1
 - *Extensive manual activities (potential miniphases):* Manipulate UI-Prototype, Create/Manipulate UI-Specification, Manipulate Specification Model, Create Specification, Manipulate Construction Model, Write Code
 - *Global synchronization points:* 1
 - * Create/Manipulate UI-Specification
 - *Global Phases:* 1 (14 activities)
 - *Miniphases:* 0 (Create/Manipulate UI-Specification is combined with phase)
 - * *After all:* Generate UI-Prototype, Manipulate UI-Prototype, Create Specification Model, Generate Parts for Specification, Create Specification, Manipulate Specification Model, Diff. Models, Merge Versions of Specification Model, Create Constructionmodel, Manipulate Constructionmodel, Validation, Generated Code, Write Code
 - *Local focuses with Local synchronization points::*
 - * **a** Development of UI-Prototype (which is later reused to write code)
 - *Local synchronization points:* Manipulate UI-Prototype, Generate UI-Prototype, Create/Manipulate UI-Specification, Write Code
 - *Combined local focus:* Create/Manipulate UI-Specification, Manipulate UI-Prototype, Generate UI-Prototype, Write Code
 - *Size of local focus:* 4 activities out of 14 activities
 - *Phases:* 0
 - *Miniphases:* 2 (Create/Manipulate UI-Specification and Manipulate Prototyp with Write Code)

- * **b** Development of Specification as additional product
 - *Local synchronization points*: Generate Parts for Specification, Create Specification Model, Create/Manipulate UI-Specification, Create Specification
 - *Combined local focus*: Generate Parts for Specification, Create Specification Model, Create/Manipulate UI-Specification, Manipulate Specification Model, Diff. Models, Merge Versions of Specification Model, Create Specification
 - *Size of local focus*: 7 activities out of 14 activities
 - *Phases*: 1
 - *Miniphases*: 2 (Create/Manipulate UI-Specification, Create Specification)
 - *After*: *Create Specification Model*; *Create/Manipulate UI-Specification – before*: *Generate Parts for Specification*; *Create Specification*: Manipulate Specification Model, Diff. Models, Merge Versions of Specification Model
- * **c** Main development path
 - *Local synchronization points*: Generated Code, Create Constructionmodel, Create Specification Model, Create/Manipulate UI-Specification, Manipulate Constructionmodel, Write Code
 - *Combined local focus*: Create/Manipulate UI-Specification, Generated Code, Create Constructionmodel, Create Specification Model, Manipulate Specification Model, Diff. Models, Merge Versions of Specification Model, Manipulate Constructionmodel, Write Code
 - *Size of local focus*: 9 activities out of 14 activities
 - *Phases*: 1
 - *Miniphases*: 3 (Create/Manipulate UI-Specification, Manipulate Constructionmodel, Write Code)
 - *After*: *Create Specification Model*; *Create/Manipulate UI-Specification – before*: *Create Constructionmodel*; *Manipulate Constructionmodel*; *Generated Code*; *Write Code*: Manipulate Specification Model, Diff. Models, Merge Versions of Specification Model
- Cap2a
 - *Extensive manual activities (potential miniphases)*: Create/Manipulate Plain Text Part, Manipulate Model, Manipulate Result2
 - *Global synchronization points*: 3
 - * Create/Manipulate Configuration
 - * Merge Parts of Model
 - * Manipulate Result2
 - *Global Phases*: 1 (7 activities)
 - *Miniphases*: 1 (Manipulate Result2)
 - * *Before all*: Create/Manipulate Plain Text Part, Create Model, Manipulate Model, Check References in Model
- Cap2b
 - *Extensive manual activities (potential miniphases)*: Manipulate Engineering Model, Manipulate Result3, Manipulate Result2
 - *Global synchronization points*: 2
 - * Create Engineering Model
 - * Manipulate Engineering Model
 - *Global Phases*: 1 (11 activities)
 - *Miniphases*: 0 (Manipulate Engineering Model is connected to phase)
 - * *After all*: Check Engineering Model, Generate Result9, Generate Result5, Generate Result7, Generate Result8, Word Import Result2, Manipulate Result2, Word Import Result3, Manipulate Result3
 - *Local focuses with Local synchronization points*:
 - * **a** Development of EVKonzept
 - *Local synchronization points*: Word Import Result2, Create Engineering Model, Manipulate Engineering Model, Manipulate Result2
 - *Combined local focus*: Word Import Result2, Create Engineering Model, Manipulate Engineering Model, Manipulate Result2
 - *Size of local focus*: 4 activities out of 11 activities
 - *Phases*: 0
 - *Miniphases*: 2 (Manipulate Engineering Model, Manipulate Result2)
 - * **b** Development of Result3
 - *Local synchronization points*: Word Import Result3, Create Engineering Model, Manipulate Engineering Model, Manipulate Result3
 - *Combined local focus*: Word Import Result3, Create Engineering Model, Manipulate Engineering Model, Manipulate Result3
 - *Size of local focus*: 4 activities out of 11 activities
 - *Phases*: 0
 - *Miniphases*: 2 (Manipulate Engineering Model, Manipulate Result3)

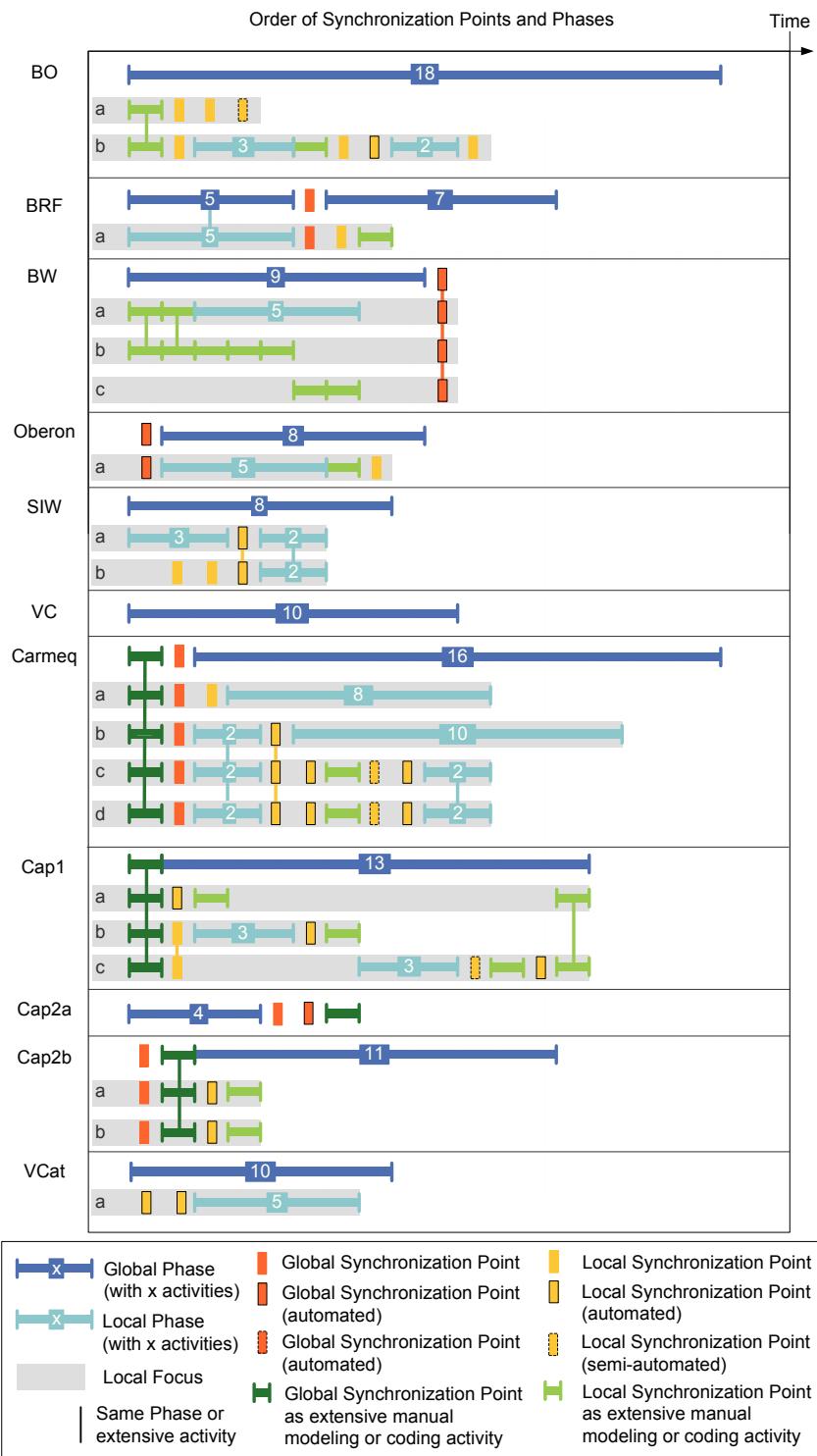


Figure G.4.: Summary of order of global and local synchronization points and phases

Appendix H.

Literature on Support for Evolution in Context of MDE

As indicated in the conclusion in Section 12.3.2, a small classification of what change types are considered within exemplary approaches for evolution support from literature was made. This appendix shows this classification. An overview how evolution of languages or transformations is addressed in literature is given in this section (summarized in Table H.1). Note that evolution of single tools is not in focus of the following discussed research works.

First there are approaches that support specific changes of model operations (*C1*), e.g. MDPE workbench (as described by Fritzsche and Johannes [83] and Mohagheghi et al. [146]) capsules the application of an extensible set of performance analysis techniques on models.

Another example for a specific supported change is the MasterCraft code generator presented by Kulkarni et al. in [120]. The generator is built such that it can easily be configured to implement architectural decisions taken within a project.

Other approaches address the change of automated activities in a more general form. For example, in [121] Küster et al. discuss a method for incremental development of a transformation chain and Yie et al. [216] approach the adaption of a fully automated transformation chain. Both approaches work for automated activities that are implemented in form of a transformation chain.

Some approaches deal with language evolution and migration of models, such that they become valid for a new version of the language (*C2*). For example, Garcés et al. use the differences between two metamodels to generate a transformation that migrates the corresponding models [87]. Further examples are the Modif metamodel presented by Babau and Kerboeuf in [12], Model Change Language (MCL) presented by Narayanan et al. in [150], or the usage of higher-order model transformations for co-evolution of metamodels and models (as presented by Cicchetti et al. [43], Meyers et al. [142], or Van den Brand et al. [199]). An example for co-evolution support through the use of transformation pattern are presented by Wachsmuth [208]. Finally, Gruschko et al. present a migration process to synchronize models with evolving meta models [93].

A special example for an approach that supports language evolution is presented by Demuth et al. in [49]. This approach supports the adaption of constraints according to the evolution of a meta model.

From the viewpoint of an MDE setting, all these approaches address the evolution of modeling languages (including co-evolution of meta models and models), only, and conform to change type *C2*.

Some MDE approaches expect specific simultaneous changes of used languages and transformations (co-evolution of meta models and transformations). For example, the OMG's MDA [154] and the migration process presented in [72] expect exchange of languages and transformations to address a new target platform for a software system. Other approaches deal with systematically adapting transformations according to language evolution (*C1* and *C2*). For example, Herrmannsdoerfer et al. propose in [97] a systematic strategy and Levendovszky et al. propose in [126] a semi-automated support for the adaption of the model transformation according to changes in a meta model are proposed. Vermolen et al. lift the semi-automated adaption of transformations to modeling languages on a more technology-independent layer by allowing also migrations of programming languages or data structures [203].

Meyers et al. subdivide metamodel- and model evolution into four primitive scenarios, describing how evolution of models, metamodels, or transformations enforces co-evolution among each other [141, 140]. The resulting scenarios are combinations of the change types *C1* and *C2*.

Finally, there are approaches that lead to an addition of input models to an automated activity, which is exchanged or evolved. Specific side effects are an increasing number of models (*C4*) and potentially modeling languages (*C5*). Further, in most cases the introduction of a new input model leads to an additional manual modeling activity for creating this new model (*C6*).

An example is presented by Johannes and Fernandez in [104], where a system of DSLs that are used in combination can be extended due to the hierarchical structure of the DSLs. Similarly, Estublier et al. present in [61] a modifiable interpreter. Here a composition model is used to define how different domain-specific models are related.

To summarize, none of the approaches considers substantial structural evolution. Further, support for structural evolution that is not substantial is provided by two approaches for specific changes only.

Table H.1.: Change types considered in literature on evolution support (· = specific changes covered; ○ = approach provides solution with assumptions on the language or implementation to be changed; ● = approach with a general coverage of the change type)

Kind of Changes\Approaches	[146]	[121]	[87]	[150]	[154]	[97]	[203]	[141]	[104]	[61]
	[120]	[216]	[12]	[43]	[72]	[126]		[140]		
				[142]	[199]					
<i>Non-Structural Changes</i>										
C1 exchange automated activity	·	○			·	·	·	○	○	○
C2 exchange language		○			○	●	○			○
C3 exchange tool										
<i>Structural Changes</i>										
C4 change number of artifacts							·	·		
C5 change number of languages							·	·		
C6 change number of manual activities							·	·		
C7 change number of tools										
C8 change number of automated activities										
C9 change order of manual / automated activities										

Appendix I.

Overview of Publications

Following, publications are listed that were created in context of this research and are basis for the thesis.

Selected Publications In [P1] the research plan for this thesis is presented and discussed. In [P5] a survey on mega model research is given. The Software Manufacture Model language is presented in [P6]. [P2] introduces the Software Manufacture Model Pattern notation together with four proto-patterns. The research on MDE evolution will be published in [P4].

Further, the case studies that were captured in context of this research, are summarized and discussed in different reports. Thereby, [P3] (SAP case studies) is published as technical report. For the other reports (for the Capgemini case studies, the Carmeq case study, and the VCat case study) the final permission for publication from the cooperation partners is still missing. The attached CD includes these reports and models of the case studies VCat, Carmeq, Cap1, Cap2a and Cap2b. Since all three reports are not yet published as technical reports, the three reports should be treated confidential and are explicitly attached for the reviewers of the thesis, only. They shall not be shared with third parties. As mentioned in Chapter 4, the names of activities and artifacts in the Capgemini case studies have been substituted within this thesis, in order to not get in conflict with the permission process. The CD includes the mapping between names used within this thesis and in the reports (in form of an excel file).

Additional Publications In addition to the publications that will be basis for the thesis, some additional work has been done in context of the research on model management at the System Analysis and Modeling group. Thereby, the consequences of the structure of MDE on the design of build servers ([A12]) is discussed. Further works concern the tracing in model management ([A1], [A10]) and automated composition of model operations ([A11]).

Finally, some publications on related topics are: [A6], [A5], [A3], [A7], [A8], [A4], [A9], and [A2].