Model-Based Software Engineering: A Multiple-Case Study on Challenges and Development Efforts

Rodi Jolak, Truong Ho-Quang, Michel R.V. Chaudron
Chalmers | Univesity of Gothenburg
Gothenburg, Sweden
{jolak,truongh,chaudron}@chalmers.se

Ramon R.H. Schiffelers
Technische Universiteit Eindhoven | TUE
& ASML Netherlands B.V.
Eindhoven, Netherlands
r.r.h.schiffelers@tue.nl

ABSTRACT

A recurring theme in discussions about the adoption of Model-Based Engineering (MBE) is its effectiveness. This is because there is a lack of empirical assessment of the processes and (tool-)use of MBE in practice. We conducted a multiple-case study by observing 2 two-month MBE projects from which software for a Mars rover were developed. We focused on assessing the distribution of the total software development effort over different development activities. Moreover, we observed and collected challenges reported by the developers during the execution of projects. We found that the majority of the effort is spent on the *collaboration and communication* activities. Furthermore, our inquiry into challenges showed that tool-related challenges are the most encountered.

CCS CONCEPTS

• **Software and its engineering** → *Software development techniques*; *System modeling languages*; *Development frameworks and environments*; *Collaboration in software development*;

KEYWORDS

Software Engineering, Model-Based Engineering, Effort Distribution, Modeling Tools, MBE Challenges, Case Study Design

ACM Reference format:
Rodi Jolak, Truong Ho-Quang, Michel R.V. Chaudron and Ramon R.H. Schiffelers. 2018. Model-Based Software Engineering: A Multiple-Case Study on Challenges and Development Efforts. In *Proceedings of ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems, Copenhagen, Denmark, October 14–19, 2018 (MODELS '18),* 11 pages.
https://doi.org/10.1145/3239372.3239404

1 INTRODUCTION

Models provide effective means for supporting the communication between stakeholders, and serve as specification for the implementation of software systems. Model-Based Engineering (MBE) is a software development approach in which models play an important-*central* role [5]. MBE aims to increase the abstraction level and aims to promote the automation of the development process [26].

MODELS '18, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-4949-9/18/10…$15.00
https://doi.org/10.1145/3239372.3239404

Empirical assessment of the use and process of MBE is scarce. The adoption of MBE is still debated in practice. On the one hand, MBE has been applied effectively in several application sectors, e.g., embedded systems [18] and telecommunication [2]. Furthermore, by focusing on practitioners' experiences and perceptions, several studies claimed that the adoption of MBE helps to (i) improve the productivity of the developing teams by increasing the abstraction level, (ii) enhance the quality of the software and (iii) support its maintainability [18–20]. On the other hand, some practitioners consider MBE as a time-consuming and unproven approach that merely complicates matters [26].

1.1 Rationale

Generally, the field of software engineering perceives a discrepancy between empirical software engineering findings and developers' (*a priori*) beliefs and opinions, which are often based only on personal perspectives on the development processes. Devanbu et al. [7] suggested that more in-depth studies that address the interplay of belief and evidence in software practices are needed. The same issue is pointed out by Ralph [24] who differentiated between two paradigms of software development research: *Empirical* and *Rational*. Empiricists believe that knowledge can only be justified by sense experience and observation. In contrast, rationalists accept that some knowledge can be justified by observation, but claim that other knowledge is justified by reason or intuition. Ralph claims that the rational paradigm continues to dominate the software engineering standards and approaches: many developers and researchers hold beliefs that are incongruous with empirical evidence. This, according to Ralph, would undermine the software engineering community's scientific credibility.

1.2 Objective and Contribution

This study contributes to the body of knowledge on the process and use of MBE in practice. In particular, we conducted a multiple-case study [32] by analyzing and discussing empirical data collected from 2 two-month MBE projects carried out at the Technical University of Eindhoven, the Netherlands. The main contributions of this paper are two-fold:

- Firstly, we shed light on the distribution of development efforts in MBE. The resulting observations on effort distribution could lead to improved MBE project planning and organization (e.g., resource allocation and risk management), which in turn could lead to cost reduction.
- Secondly, we report and further analyze different challenges to the process and use of MBE in practice. Exposing such

challenges would make them a candidate subject for research that are concerned with MBE process improvement. Moreover, understanding and providing ways to overcome these challenges could bring a significant impact to the effectiveness and efficiency of the MBE approach.

In this paper, we address the following research questions:

- **R.Q.1** How is the total effort spent on MBE distributed over different development activities?
- **R.Q.2** How is the effort spent on different MBE development activities distributed over time?
- **R.Q.3** How large is the portion of collaborative work in MBE projects?
- **R.Q.4** What are the challenges that affect MBE in practice?
- **R.Q.5** How are the challenges that affect MBE distributed over project time?

The remainder of this paper is organized as follows: in Section 2, we consider and discuss the related work. We describe the case study design in Section 3. We present and discuss the results in Section 4. We discuss the threats to the validity of this study in Section 5. Finally, we conclude and discuss the future work in Section 6.

2 RELATED WORK

In this section, we review the published work on: (i) measuring effort distribution between MBE development phases, and (ii) challenges encountered when adopting MBE.

2.1 Effort distribution in MBE

Distribution of effort in software engineering processes is largely researched in the context of estimation and planning of software projects [15]. Several practitioners studied the effort required for different software development activities, and provided rules of thumb such as the "40-20-40" rule of Pressman [23], that is 40% on analysis and design, 20% on coding and 40% on integration and testing. Other rules of thumb were provided by: Ambler [1], Boehm [3], Boehm et al. [4], Brooks [6] and Zelkowitz [33].

In his text book, Sommerville [27] estimates effort distribution by measuring cost units in different development activities, i.e., 15 units on specification, 25 units on design, 20 units on development/implementation and 40 units on testing.

Yang et al. [31] empirically studied development effort distribution of 75 projects from 46 software organizations from the China Software Benchmarking Standard Group (CSBSG) database. The development approaches defined in CSBSG database roughly follow the waterfall model, including planing, requirements, design, coding, testing and transition. The following mean efforts over each development phases are reported: 16.14% for planning and requirements, 14.88% for design, 40.36% for coding (including unit test and integration), 21.57% for testing (system testing), 7.06% for transition (including installation, acceptance test and user training).

Recent works including the one by Papatheocharous et al. [22] studied effort distribution based on projects obtained from the International Software Benchmarking Standards Group (ISBSG) R10 dataset [12]. The six development phases declared by ISBSG are: planning, specification, design, build, test and implementation. The mean efforts spent on each development phase are reported as follows: 8.2% for planning, 7.9% for specification, 11.9% for design, 36.8% for developing and building, 15.5% for testing, 5.6% for implementation and 14.0% for unphased activities.

On the basis of data collected from 20 industrial software development projects, Heijstek and Chaudron [9] reported effort distribution over various disciplines, defined by the Rational Unified Process (RUP), in MBE. The following effort distribution was reported: 11% for analysis & design, 8% for requirements analysis, 12% for testing, 38% for implementation, 13% for project management, 4% for change & configuration management, 3% environment, 2% for deployment, 9% for others choices. This is, according to the authors, surprisingly similar to the RUP Hump chart, and thus underlines the similarity between MBE and traditional development approaches. To the best of our knowledge, this study is so far the only one that investigates effort distribution in MBE projects.

2.2 Challenges in MBE

Although MBE claims many potential benefits, e.g., gains in productivity, portability, maintainability and interoperability [13, 14, 19], its adoption has been facing a number of challenges. These challenges are discussed in academic forums and empirically investigated in a number of industrial cases.

Van Der Straeten et al. [29] summarize outcomes of a plenary session at the MODELS'08 workshop on "Challenges in Model-Driven Software Engineering" where participants discussed challenges in the field of MDE. Discussed challenges included: management of models quality, lack of focus on modeling process and models at run-time, and insufficient MDE tool-support.

Lately, Mussbacher et al. [21] reflected opinions of 15 MDE experts on the biggest problems with MDE technologies over the last 20 years. The authors highlighted that tools *usability* and *adoption*, people's diverse perception of MDE, inconsistencies between software artifacts, and lack of fundamentals in MDE are considered as hindrances to MBE adoption.

Baker et al. [2] discussed experiences with MBE/MDE at Motorola over a time span of almost 20 years. A number of challenges were reported, such as poor tools and generated code performance, lack of integrated tools, and lack of scalability.

Hutchison et al. [11] analyzed 250 survey-responses and 22 interviews, as well as did on-site observations of MDE. They found that the main challenges to MDE adoption are significantly related to Domain-Specific Languages (DSLs) and MDE tools, as well as to organizational factors and human training issues. Based on a survey involving 113 software practitioners, Forward and Lethbridge [8] reported common problems with model-centric development approaches. These problems are related to inconsistency of models over time, model interchange between tools, and heavyweight modeling tools.

Similarly, by surveying 155 Italian software professional, Torchiano et al. [28] considered lack of competencies and supporting tools as the main show stoppers preventing altogether the adoption of modeling and model-driven techniques.

In the embedded systems domain, Liebel et al. [17] analyzed survey-responses from 122 professionals working with MBE, and considered that interoperability between (MBE/MDE) tools as a main challenge to MBE adoption. Moreover, other factors such as,

high effort to train developers and tools (poor) usability, were also identified as secondary MBE challenges.

While the above-mentioned studies help in exploring challenges, there are a number of other studies which focus on specific challenges, especially tool-related ones. By performing a series of interviews with 20 engineers and managers at General Motors, Kuhn et al. [16] identified five points of friction in MDE. All of them are related to MDE tools. Similarly, by analyzing a total of 39 interviews with industrial practitioners, Whittle et al. [30] identified a taxonomy of technical, social and organizational issues related to MDE tool use in practice. Addressing such issues together with modeling tools-related issues identified by this study, would probably ameliorate the effectiveness and efficiency of MBE.

3 CASE STUDY DESIGN

Yin defined case studies as empirical inquiries to perform a deep investigation of a particular phenomenon, where the boundary between the phenomenon and its real-life context cannot be clearly specified [32]. Our multiple-case study is an *exploratory- inductive* empirical research [25] conducted to identify patterns in observations, seek new insights, and generate ideas and hypotheses for new research. Figure 1 shows the design of our multiple-case study. Further details regarding the case study design will be presented in the following subsections.

3.1 Purpose and Cases

Multiple-case studies are regarded as being more robust than single-case studies, and the evidence from multiple cases is often considered more compelling [10]. The intention of the study is to explore the effort distribution over different MBE development activities. Moreover, the study seeks to identify challenges and impediments that could hinder the use of MBE in practice. In particular, two cases were examined:

(1) *MathWorks*: MBE of the software of a Mars rover using MBE tools as provided by MathWorks technologies, e.g., Matlab and Simulink.
(2) *PolarSys*: MBE of the software of the same rover using MBE tools as provided by PolarSys open source technologies, e.g., Papyrus and Capella.

3.2 Units of Analysis

The multiple-case study is embedded [32], with two Units of Analysis (UoA):

(1) *Effort Distribution*: Analysis of the distribution of the total MBE effort over different development activities over time.
(2) *MBE Challenges*: Analysis of the challenges that could hinder the adoption of MBE in practice.

3.3 Propositions

The two cases are selected to predict possible contrasting results (*theoretical replication*) on MBE challenges and development efforts by altering one condition: the used MBE tools (MathWorks vs. PolarSys). Furthermore, the findings of the this study will be compared to those of other related work in order to find out any eventual supportive similarities or contradicting differences.

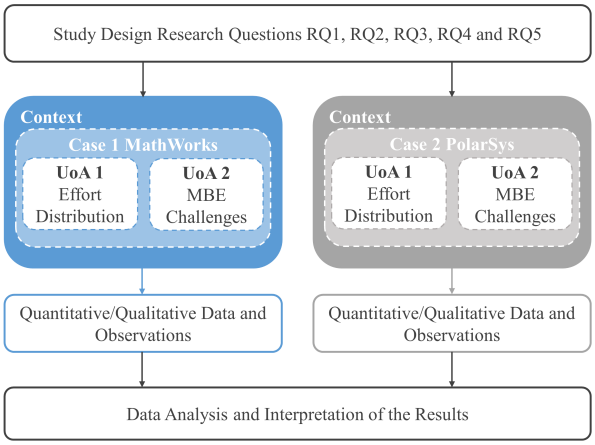


Figure 1: Case Study Design.

In particular, based on the related work we state the following two propositions:

- *Proposition A*: We propose that the distribution of development effort over different MBE activities follows the rules-of-thumb e.g., the "40-20-40" rule. *If* our findings are not compliant, *then* a deeper investigation of the MBE approach is needed in order to understand why the effort distribution deviates from the standard rules.
- *Proposition B*: We propose that poor tool-support is the most frequently reported challenge that affect the adoption of MBE in practice. *If* the perceived challenges in our study do not match, *then* a deeper investigation of the severity of the perceived challenges (both of our study and related work) is needed.

Moreover, for the scope of this multiple-case study, and based on the planned cross-case analysis, we state the following two additional propositions:

- *Proposition C*: We propose that the distribution of the development effort in case 1 matches that of case 2. *If* the distributions do not resemble, *then* the choice of tools and technologies in MBE could affect the development effort. Hence, a deeper investigation of the impact of MBE tools on the development effort is needed.
- *Proposition D*: We propose that similar challenges would be perceived in the two cases. *If* different challenges with different severities are perceived, and if the differences are mainly related to the used MBE tools, *then* we suggest that there is a difference in the maturity between the two technologies (i.e., MathWorks vs. PolarSys).

3.4 Context

The Professional Doctorate in Engineering (PDEng) program of the TU/e aims to train graduates that have (a non-CS) MSc diploma to become professional software engineers. A group of 17 trainees of this program were involved in two projects to develop software for a couple of collaborating concurrently-maze-discovering rovers.

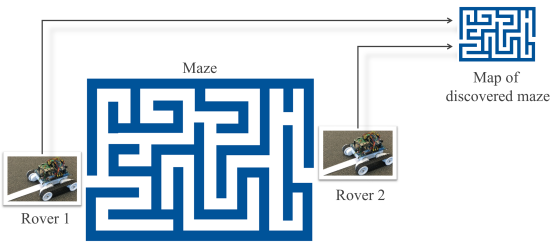


Figure 2: Maze-discovering rovers assignment.

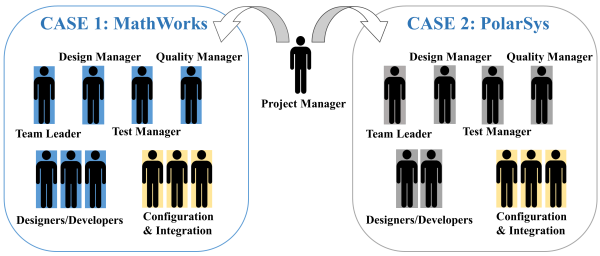


Figure 3: Organization of the development teams.

The rover system was chosen because of its similarity to the type of software that is developed at the ASML company[1] in Eindhoven.

For both projects, it was mandatory to use MBE as the development approach (e.g., *almost* all software were developed and generated based on, and from, models). The objective of the projects was to develop control software for two rovers that would concurrently discover a single maze of roads as fast as possible. See Figure 2. Here, a road consists of a small line of tape with a reflection that differs sufficiently from the underground it is mounted on. The rovers were equipped with IR sensors and a (low-resolution) PID-controller that enabled the rovers to autonomously follow the roads. The rovers had to communicate with each other in order to complete their task in discovering different parts of the same maze. The discovered map of the maze had to be visualized to the end-user during the discovery, and had to be persisted as an end-result.

The trainees were bootstrapped with the hardware and a software API to control the hardware (e.g., motion of the rover), as well as to get the sensors' readings (for e.g., line tracking). The software API was provided at the start of the projects, the hardware itself was provided late in the projects as usually is the case in real-life situations. The main deliverable of the two projects was to produce the rovers' software application. However, in order to test the maze-discovering software application, the trainees needed to develop a software simulator of the hardware. Once the rovers' software application was verified and validated, the simulator was replaced by the actual hardware.

The supervisors of the projects could accept deviations from the initial requirements w.r.t. the development process. That would be possible if and only if such deviations were very well motivated and supported by the trainees.

The group of trainees was organized as follows:

- *Team MathWorks*: Consisted of seven trainees: One team leader responsible of general team tasks as well as team process, risks and design. One design manager appointed to collaborate with the PM on the architecture. One test manager responsible for managing UI tests, unit tests and acceptance tests. One quality manager responsible for code-review for quality assurance and managing documentation (coding and documentation standards). Three developers responsible for modeling, code generation, and manual coding as well as testing. The *MathWorks* team had to develop both software systems (the rover software application and the simulator) using MBE/MDE tools as provided by the MathWorks[2] technologies, e.g., Matlab[3] and Simulink[4].
- *Team PolarSys*: Consisted of six trainees. One team leader, one design manager, one test manager, one quality manager and two developers. These roles had the same responsibilities as described in team *MathWorks*. The *PolarSys* team had to develop the same software applications using MBE/MDE tools as provided by the PolarSys[5] open source technologies, specifically, *Papyrus*[6] and *Capella*[7]. Other tools used by the two teams are reported later in Section 4.4.
- *Configuration & Integration Support*: Three trainees given the task to setup, configure and support a continuous development and integration environment for both *MathWorks* and *PolarSys* teams.
- *Project Management (PM)*: Both teams were managed by one trainee, who was responsible of the plan, process, risks, architecture and integration management of the two teams.

The two teams organized themselves in an agile way and had to complete their projects in two months working full-time (i.e., each trainee worked approximately 8 hours per day). The trainees of each team had to meet with the PM and discuss the progress on a weekly basis. Figure 3 summarizes the organization of the development teams augmented with size and role information.

3.5 Data collection and Analysis

The data collection consisted of weekly questionnaires as well as developers' time and actions tracking tools. Each week, the developers had to answer a questionnaire[8] in which we collected, amongst others, evidence on: (i) the perceived effort spent on different MBE activities, and (ii) challenges and impediments that affected the development process. The *ProcrastiTracker*[9] software was used to automatically track for each developer which applications and documents were used on their computer and for how long. We also collected the recorded log files produced by this software for each developer on a weekly basis.

We intend to analyze the results by means of *pattern matching* and *cross-case synthesis* [32]. Pattern matching helps to compare

[1] https://www.asml.com

[2] https://se.mathworks.com
[3] https://se.mathworks.com/products/matlab.html
[4] https://se.mathworks.com/products/simulink.html
[5] https://www.polarsys.org
[6] https://www.eclipse.org/papyrus
[7] https://www.polarsys.org/capella
[8] http://www.rodijolak.com/pdf/WeeklyQuestionnaire.pdf
[9] http://strlen.com/procrastitracker/

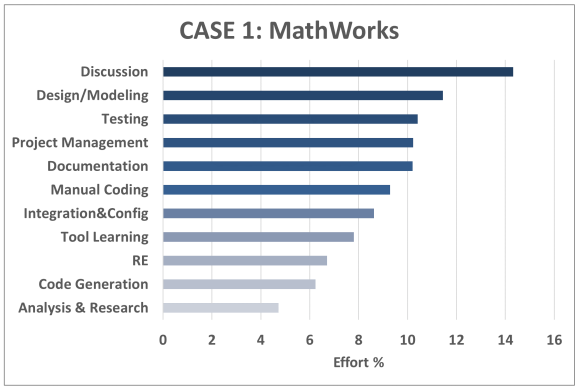


Figure 4: Total Effort Distribution Case 1

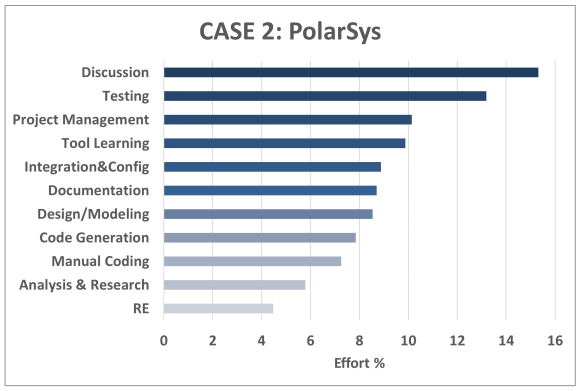


Figure 5: Total Effort Distribution Case 2

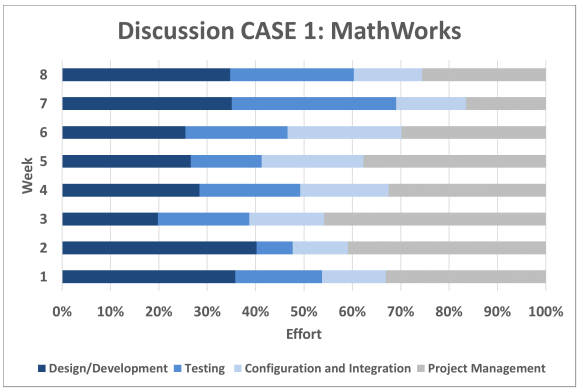


Figure 6: Discussion Effort Distribution Case 1

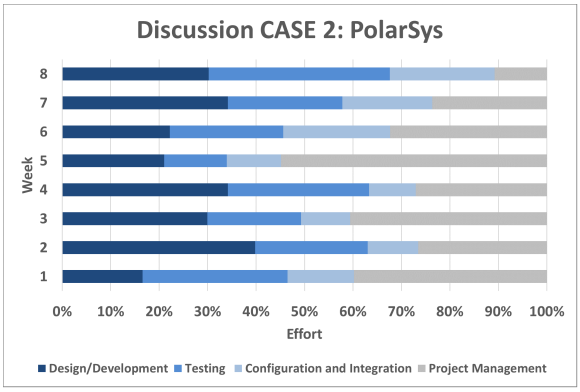


Figure 7: Discussion Effort Distribution Case 2

an empirically observed pattern with another pattern. When they agree then the pattern is true. Whereas, cross-case synthesis can be used in multiple-case studies to investigate and compare the different cases. Moreover, we intend to use NVivo[10] for qualitatively analyzing the data related to the experienced challenges to MBE approaches.

4 RESULTS

In this section, we present the findings of this study together with their interpretation, as well as in relation to the published work and stated propositions. We recall that the findings are based on the considered multiple-case study and its context. Also, as mentioned previously in Section 3.5, we recall that we used *pattern matching* and *cross-case synthesis* for data analysis and interpretation.

4.1 Development Efforts (R.Q.1)

Figures 4 and 5 orderly arrange the percentage values of the efforts spent on different MBE activities in case 1 (*MathWorks*) and case 2 (*PolarSys*), respectively. As can be noted, the majority of the effort is spent on *Discussion* (14.32% in case 1 (*MathWorks*) and 15.32% in case 2 (*PolarSys*)). More details regarding the topics/arguments of the discussions related to case 1 and 2 are provided by Figures 6 and 7, respectively. In particular, these figures report an estimation of the percentages of the topics that were discussed each week. We got these estimations by matching the reported percentages on development discussions with the role responsibility of the reporting developer. Based on this, four main discussion topics were identified: Design and Development, Testing, Configuration and Integration, and Project Management. It can be observed that the majority of the discussions were about *Project Management* and *Design and Development*.

Table 1 shows a comparison of the effort phase distribution between the related work and our multiple-case study. This table is inspired by Papatheocharous [22]. First of all, it seems that the effort phase distributions in our two cases are compliant with the RUP's effort distribution reported by [1]. Moreover, it seems that the development effort distribution in the two cases is compliant with the "40-20-40" rule-of-thumb. This suggests that the effort distribution in MBE projects does not deviate from the distribution defined by the standard rules (*Proposition A*).

Giving a look on the separate development phases, we note the effort spent on *planning* (i.e., project management activities such as: define project scope, allocation, estimate cost, risks and schedule, etc.), *RE*, *specifications* and *testing* in our two cases seem to be in-line with the efforts reported by the related work, especially the

[10]https://www.qsrinternational.com/nvivo

Table 1: Effort phase distribution reported in literature

Study	Planning	Requirements	Specifications	Design	Coding	Testing	Integration
Ambler (RUP)[1]		Inception (10%)	Elaboration (25%)		Construction (55%)		Transition (10%)
Zelkowitz[33]		RE (10%)	Spec (10%)	Design (15%)	Coding (20%)	Testing (45%)	
Brooks[6]	Planning (33%)				Coding (16%)	Testing (25%)	Integration (25%)
Sommerville[27]			Specs (15%)	Design (25%)	Development (20%)		Testing (40%)
Boehm[3]			RE - Analysis - Design (60%)		Coding (15%)		Testing (25%)
Pressman[23]			Analysis & Design (40%)		Coding (20%)		Testing & Integration (40%)
Papatheocharous[22]	Plan (9.6%)		Specs (9.3%)	Design (14.0%)	Build (42.3%)	Testing (18.2%)	Implement (6.6%)
Heijstek[9]	Planning (13%)	RE (8%)	Analysis & Design (11%)		Coding (38%)	Testing (12%)	Configuration (4%)
Yang[31]		Planning & RE (16.1%)		Design (14.9%)	Coding (40.3%)	Testing (21.6%)	Transition (7%)
Case Study 1 (MathWorks)	Planning (15.0%)	RE (10.0%)	Analysis & Research (7.0%)	Design & Modeling (17.0%)	Code Generation (9.2%) Manual Coding (13.6%)	Testing (15.4%)	Integration & Configuration (12.7%)
Case Study 2 (PolarSys)	Planning (15.3%)	RE (6.8%)	Analysis & Research (8.8%)	Design & Modeling (13.0%)	Code Generation (12.0%) Manual Coding (11.0%)	Testing (20.0%)	Integration & Configuration (13.2%)

recent works of Heijstek [9], Papatheocharous [22] and Yang [31]. Surprisingly, the effort spent on software design and modeling in our two cases is similar to the findings of the other related work, especially [22], [31] and [33]. This finding empirically suggests that MBE approaches do not require a lot of effort on design and modeling, as it is *believed* among many developers. Moreover, code-generation based on models allowed our teams to spend less effort on manual coding. In particular, the combined effort spent on code-generation and manual coding together in each of our two cases is less that the effort of coding reported by [9, 22] and [31]. This empirical finding confirms that MBE approaches require less effort on manual coding because most of the code is obtained from models via code-generation.

By doing a cross-case analysis, we interestingly note that the effort distributions across phases of the two cases are quite similar to each other. The use of different modeling tools in the two cases, may explain the small differences in efforts spent on *Design and Modeling*. In particular, developers in case 1 *MathWorks* spent more effort on design and modeling than the developers of case 2 *PolarSys*. Hence, it might be that different modeling tools only have a small difference in impact on the development effort (*proposition C*).

Based on empirical findings, we suggest that MBE approaches *do not* require a lot of effort on design and modeling. Moreover, they require a *little* effort on manual coding, as most of the code is obtained from models via code-generation.

4.2 Effort Distribution Over Time (R.Q.2)

Figures 8 and 9 present the effort distributions over each MBE activity during the two-month project period related to *MathWorks* case and *PolarSys* case, respectively. On the left side of the figures, eight main development activities are shown: Requirements Engineering, Analysis and Research, Design and Modeling, Code Generation, Manual Coding, Testing, Integration and Configuration, and Project Management. Whereas, on the right side of the figures, the effort distributions of other three secondary activities are reported: Documentation, Tool-Learning and Discussion.

By considering the effort spent on *design and modeling*, we notice a spike during the first week in both cases, *MathWorks* and *PolarSys*. This is because both teams used models at the beginning of the projects for ideation and discussion of design alternatives.

Team MathWorks spent more effort on *manual coding* than team PolarSys (as can be noticed by looking to figures 4 and 5), especially towards the end of the project. This phenomenon suggests that the code-generation facilities offered by MathWorks technologies are less than those of PolarSys. This is confirmed by the developers who reported that the tools offered by PolarSys (i.e., Papyrus and PapyrusRT) are more effective, user-friendly and generate code with more appropriate data structures and executables statements.

Both teams started with *testing* relatively early and throughout the projects. It is also notable that both teams spent more effort on *testing* towards the end of the projects. We think that this is a common trend in most software development projects, where more tests happen towards the end (e.g., system, integration and acceptance tests).

The effort spent on *integration and configuration* is quite similar between the two cases. Integration of software was occurring regularly in the two cases. In particular, for both cases, we notice a peak in the effort on week six. This is actually because the hardware was provided to both teams during that week, and the developers spent more effort on the configuration of the software and hardware.

As predicted, the effort on *tool learning* was high during the first weeks of the two cases, and gradually went down afterwards as the developers got more used to the tools over the time. The majority of the effort spent in the two cases was on *discussing* of the development activities. In particular, it seems that the discussions were regularly happening during the entire duration of the two projects, and not only during the planned weekly meetings.

Considering code-generation, we found that the tools offered by *PolarSys* open source technologies (i.e., Papyrus and Papyrus-RT) are *more mature* than the tools offered by *MathWorks* technologies (i.e., Matlab and Simulink).

4.3 Individual vs. Collaborative Effort (R.Q.3)

In the weekly questionnaire, we asked developers about their perceptions of the ratio of individual versus collaborative development effort that occurred during each week. Figures 10 and 11 provide an overview of the distribution of collaborative work over the entire duration of the project of case 1 *MathWorks* and case 2 *PolarSys*, respectively. Apparently in both projects, and for each week, the collaborative work was dominating. The collaborative work included discussions, group meetings, sharing knowledge and understanding, and collaborative development (e.g., definition of the

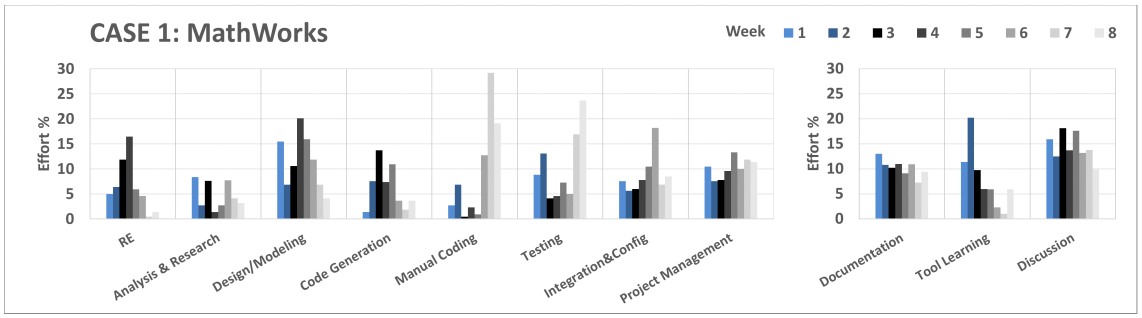


Figure 8: Effort Distribution Case 1

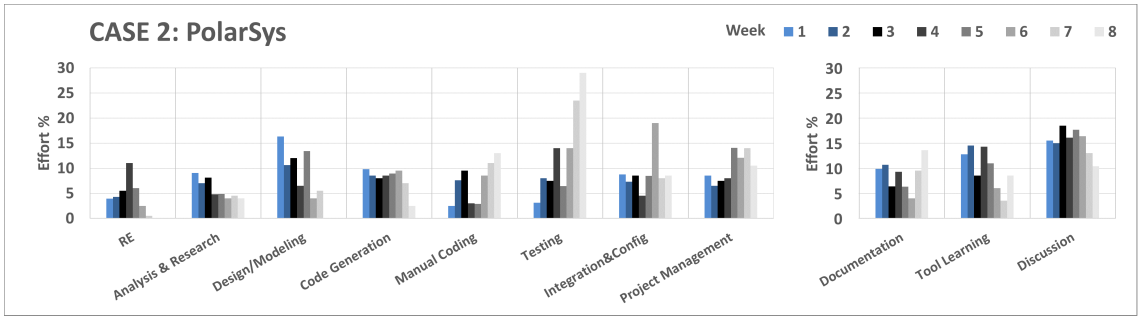


Figure 9: Effort Distribution Case 2

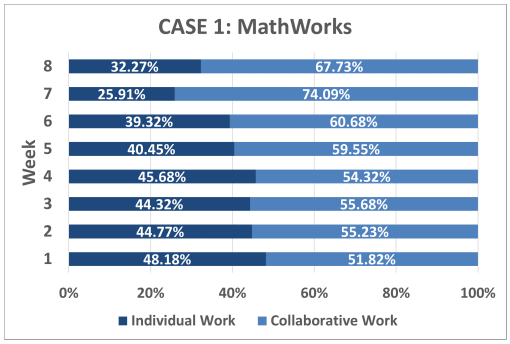


Figure 10: Case 1: individual versus collaborative work

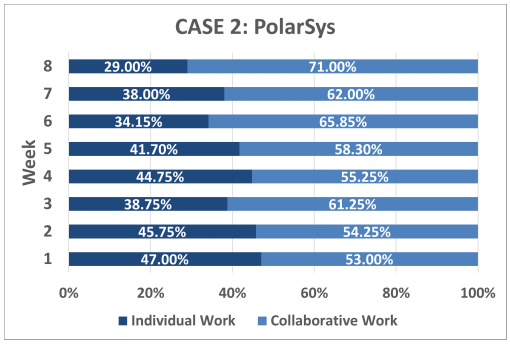


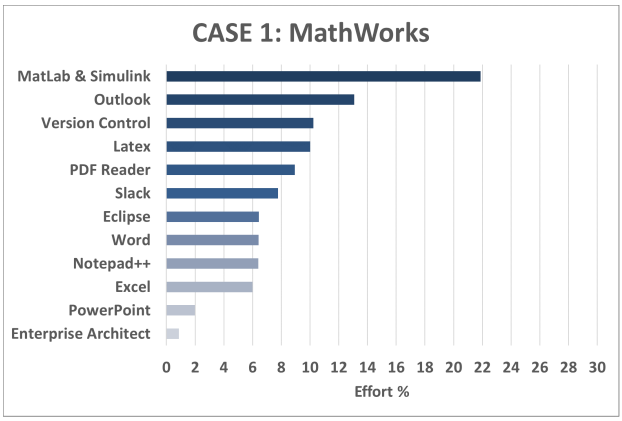
Figure 11: Case 2: individual versus collaborative work

data structure of the maze, modeling, code-generation, testing and pair programming). A further analysis of the patterns in figures 10 and 11 shows that more individual work happened at the beginning of both development projects, while more collaborative work happened towards the end. This can be explained by the fact that the developers worked individually on tools exploration and learning during the first weeks. Moreover, the developers reported that they spent more time on pair programming and testing meetings towards the end of the projects.

Our findings empirically indicate that model-based software development is an endeavor that requires intensive *communication and collaboration* between developers.

4.4 Tool-Chain

A variety of tools were used for the different development activities. These tools ranged from being main model-based development tools, such as Papyrus, PapyrusRT, Matlab/Simulink, Enterprise Architect and Capella, to other supportive tools such as Latex, Slack, PDF Reader, Version Control, Text Editor, Outlook, Power Point and Word. Figure 12 provides an overview on the tools which were used in the two cases: 1 and 2. This figure also highlights the distribution of the total tool effort percentage between the different used tools. About 22% of case 1 total tool-use effort was spent on Matlab and Simulink. Whereas, around 30% of case 2 total tool-use effort was spent on Papyrus and Papyrus RT. The focus on these tools was expected given the project's objective of tool use of the two development teams.



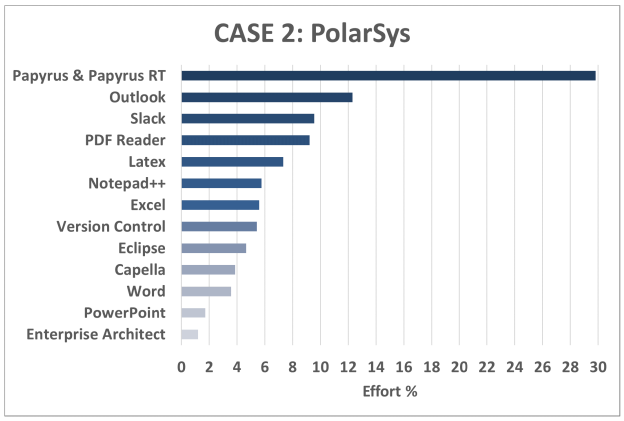


Figure 12: Overview of the used tools and their effort distribution

4.5 Experienced Challenges (R.Q.4)

Every week, the members of each project were asked to report the challenges that they experienced during the past week. The pie charts 13 and 14 orderly arrange the reported challenges ranging from the most experienced to the less experienced challenge during the execution of case 1 and case 2, respectively. *Tools Usability* was the most experienced challenge to MBE in the two cases (23% in case 1 and 25% in case 2). Challenges in *Tool-Chain Learning* were the second most experienced challenges (20% in case 1 and 19% in case 2). More details regarding the categories of the challenges are described in Table 2.

In both of the cases in our study, multiple challenges related to MBE tools were reported, such as tools- usability, learning, installation and configuration, and update. This is actually in-line with the related work (e.g., [21] and [30]) which states that poor tool-support is one of the main challenges to MBE (*proposition B*).

A cross-case analysis shows that the majority of the challenges were overall experienced similarly in both cases, especially tool-related challenges (*proposition D*). This finding indicates that the modeling tools provided by *MathWorks* and *PolarSys* are still immature and have to be enhanced in order to meet the needs of MBE and MBE developers. More information on *MathWorks* and *PolarSys* challenges are provided on-line[11].

Our findings show that tool-related challenges are the most encountered. These tool-challenges are due to: tools usability, tool-chain learning, interoperability of tools, and tools installation and configuration.

4.6 Challenges Distribution Over Time (R.Q.5)

Figure 15 presents the distribution of the experienced challenges over the eight weeks period of the two projects. It is remarkable that *Tool-chain Learning* and *Tools Usability* challenges were mostly experienced and reported during the first two weeks of the two project.

Challenges in tool-learning were also perceived in weeks 3 and 4 in both cases. For case 1 (*MathWorks*), the main reason was that the

[11] http://www.rodijolak.com/pdf/toolsChallenges.pdf

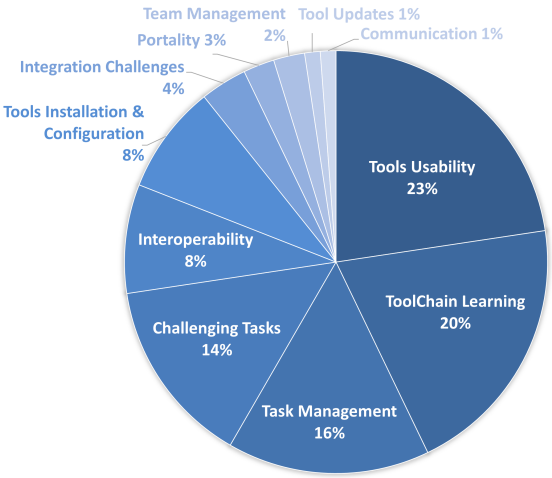


Figure 13: Experienced challenges in Case 1

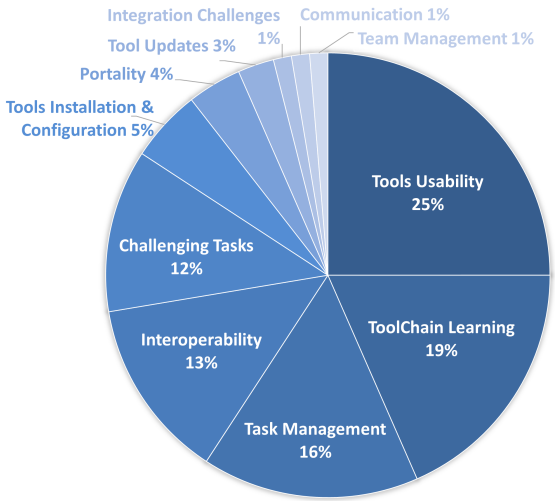


Figure 14: Experienced challenges in Case 2.

Table 2: Classification schema for the challenges

Category		Description
Tool-Chain Learning		Effort on learning the tools to be used in the project
Tools Installation & Configuration		Effort on the installation & config. of the tools on the machines of the developers
Interoperability		Missing the ability to exchange artifacts between the different tools
Tools Update		Effort on adapting the software to new tool/library versions
Tools Usability	Difficulty of Use	Complexity and cumbersomeness of the tools
	Effectiveness	Incompleteness, inaccuracy and inconsistency
	Efficiency	Long tasks' completion time
	UX	Uncomfortable tools and unacceptability of use
Task Management	Task Allocation	Effort on the organization and distribution of tasks between developers
	Synchronization	Effort on the synchronization of development activities
Team Management		Effort on the organization of the teams
Challenging Tasks		Complex development tasks that require a lot of mental effort
Portality		Difficulty in transferring software on different platforms, e.g., operative systems
Integration Challenges		Difficulty in integrating single different modules of the software
Communication		Problems caused by miss- or late communication between the stakeholders

developers spent a lot of time on learning Simulink testing-tool in order to test the simulator software. Whereas for case 2 (*PolarSys*), learning how to use third-party C++ code in a PapyrusRT project was perceived as challenging.

Tools usability challenges were reported regularly during the execution of the two projects. In particular, on week 6 challenges related to Simulink model advisor were reported in case *MathWorks*. In contrast, on weeks 5 and 6 several challenges were reported by team *PolarSys* related to a PapyrusRT update from version 0.7 to 0.8, which in turn caused lots of migration conflict issues.

Generally, it seems that the majority of the issues related to tools usability were encountered during the first weeks, when the developers started to get their hands dirty in the projects. More tool-related issues were reported afterwards by performing more activities in the projects, and hence by exploring and using more tools' functionalities.

For both cases, *Task management* challenges were basically encountered at the beginning, when the teams spent more effort on distributing the tasks between the developers, and also in different occasions afterwards (until week 7) where synchronization issues were reported.

In both cases *Challenging tasks* (see Table 2) were more encountered at the beginning- and towards the end of the projects. At the beginning, performing development activities was challenging as the developers were not so familiar with the tools. Whilst in order to finish the projects on time, the developers were over-allocated with multiple tasks towards the end of the projects.

Tools installation and configuration challenges were encountered during the first weeks of the two projects, as could be expected.

For both cases, tool-*interoperability* challenges were mainly encountered from week 3 to week 6 when several issues related to linking the produced software application with the API of the rover (e.g., coding the wrapper between the generated code and rover's API) were reported. This also caused *Portability* challenges as there were incompatibilities between some API-library dependencies and the used operative systems.

5 THREATS TO VALIDITY

We identified and grouped the threats to validity in our study according to Yin [32]:

5.1 Construct Validity

Constructs validity refers to how well operational measures represent what researchers intended them to represent in the study in question. The collection of subjective perceptions regarding development efforts and challenges after completing a project may not be optimal. This is because the subjects may fail to recall how much effort was given to a specific task or what challenges were encountered during their experience. To mitigate this, we collected the perceptions on a weekly basis: Once after the end of each week. Furthermore, we looked into the logs of the modeling tools and *Procrasti* activity tracker in order to triangulate the data which we got through collecting perceptions. In turn, the activity tracking and logging tools have a limitation. These tools log the activities only when there is an interaction between the users and their PCs. As a result, no activity does not imply that the subject is not working, for example one might be reading the document without touching the PC. We think that the two data collection approaches (i.e., collecting perceptions and logging developers' activities) adopted in this study have their own limitations. However, using multiple sources of evidence helped us to increase construct validity by encouraging convergent lines of inquiry.

5.2 Internal Validity

Internal validity concerns studies in which causal relationships are examined. Moreover, it concerns efforts made to ensure that possible confounding factors are identified and alleviated. The level of experience and expertise in MBE may influence the effort required by a developer to accomplish a specific MBE activity. This may lead to spending more or less effort on the development tasks. All of the subjects who took a part in our study are familiar with MBE because they participated in a workshop that taught the MBE development paradigm.

Some developers may consider some development tasks as challenging, while other developers may consider such tasks as less challenging. The subjects often discussed their reported challenges and motivated their perceptions. This was, to some extent, helpful to conceive the seriousness of the reported challenges. Moreover, we consider the challenges that were encountered and reported by more than one subject as more significant.

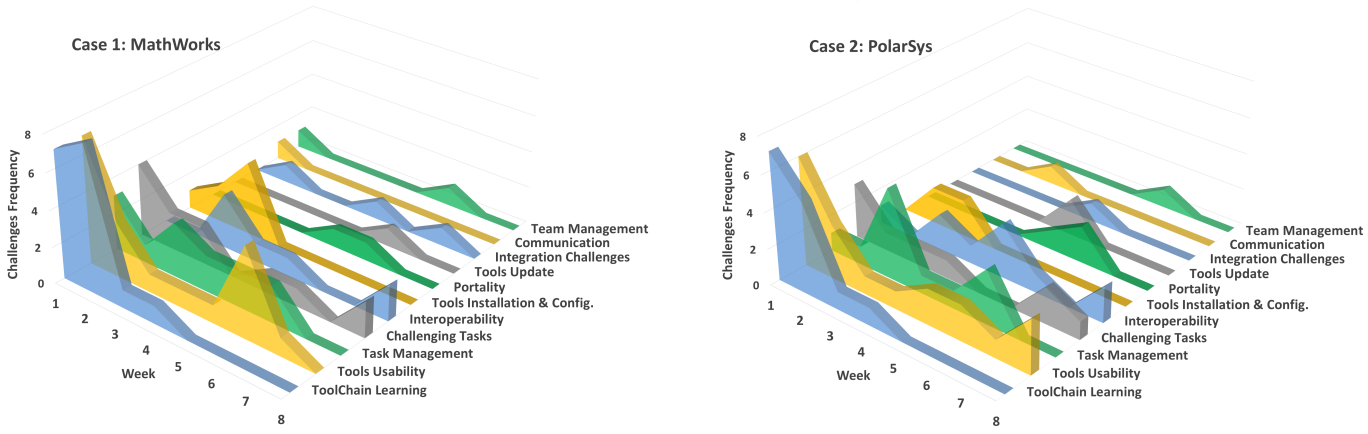


Figure 15: The distribution of the experienced challenges over the total period of the project.

We recall that our subjects are Professional Doctorate in Engineering (PDEng) trainees. This might have made the subject spending more effort on learning new tools and finding out how to work as an actual development team. However, our subjects know each other in advance. Furthermore, prior to our study, the subjects worked together on several other development projects.

5.3 External Validity

External validity concerns the extent to which results of a case study can be generalized. By design, case studies have a very limited external validity stemming from the fact that a topic is studied within its context. Therefore, we cannot claim that our findings are generalizable (i.e, generalizations to different projects in different domains might have different results). Instead, the case design and the replication logic with the cross-case analysis increases the external validity of this study. In particular, we tried to describe the case context as detailed as possible in order to allow practitioners to decide whether or not the findings might generalize to their own case context. Moreover, we underline that our study involved first-time tool users. Therefore, different results might be obtained if professionals with deep tool experience did the same projects.

5.4 Reliability

Reliability concerns the extent to which the operations of a study can be repeated by other researchers, achieving the same results. As a part of the case study design, we created a case study protocol which ensured that we conducted the study and collected the data in a consistent manner. By using this protocol, we believe that the study can be reproduced by other researchers.

6 CONCLUSION AND FUTURE WORK

In this paper we studied the effort distribution across various tasks for two projects that use different MBE tool-chains for developing an autonomous MARS rover. We obtained data both from the automatic logging of the tool-activities on the developers' computers as well as via weekly questionnaires.

Our study showed the patterns of effort distribution in MBE across different development activities as well as over time. This shows that there is no penalty in building models as part of the construction phase. Our study is the first to show that collaborative tasks make up the major part of the total of all development tasks. The resulting observations on effort distribution of this study could lead to improved MBE project planning and organization, which in turn could lead to cost reduction.

Our inquiry into challenges showed that tool-related challenges are the most encountered. We uncover that specific tool-challenges are due to: i) usability of the tools, ii) the learning of the tool-chain, iii) the interoperability of various tools and iv) the installation and configuration of the tools. Exposing such challenges would make them a candidate subject for research that are concerned with MBE process improvement. Moreover, understanding and providing ways to overcome these challenges could bring a significant impact to the effectiveness and efficiency of the MBE approach.

6.1 Future Work

As we found that the majority of the development effort is spent on the *collaboration and communication* activities, we would like to explore the effect of using models on software design communication. This in order to understand whether or not the use and share of software models could help in communicating and discussing software architectural/design decisions.

ACKNOWLEDGEMENT

The authors would like to thank Engr. Nontas Rontogiannis for his review and constructive feedback on this work.

REFERENCES

[1] Scott W Ambler. 2005. A manager's introduction to the Rational Unified Process (RUP). *http://www.ambysoft.com/downloads/managersIntroToRUP.pdf* (2005).
[2] Paul Baker, Shiou Loh, and Frank Weil. 2005. Model-Driven engineering in a large industrial contextâĂŤmotorola case study. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 476–491.
[3] Barry W Boehm. 1987. Industrial software metrics top 10 list. *IEEE software* 4, 5 (1987), 84–85.
[4] Barry W Boehm, Ray Madachy, Bert Steece, et al. 2000. *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR.
[5] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2012. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering* 1, 1 (2012), 1–182.

[6] Frederick P Brooks Jr. 1995. The mythical man-month (anniversary ed.). (1995).
[7] Premkumar Devanbu, Thomas Zimmermann, and Christian Bird. 2016. Belief & evidence in empirical software engineering. In *IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 108–119.
[8] Andrew Forward and Timothy C Lethbridge. 2008. 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*. ACM, 27–32.
[9] Werner Heijstek and Michel R. V. Chaudron. 2007. Effort distribution in model-based development. In *2nd Workshop on Model Size Metrics*.
[10] Robert E Herriott and William A Firestone. 1983. Multisite qualitative policy research: Optimizing description and generalizability. *Educational researcher* 12, 2 (1983), 14–19.
[11] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. 2011. Empirical assessment of MDE in industry. In *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 471–480.
[12] ISBSG. 2008. International Software Benchmarking Standards Group. The benchmark release 10. http:www.isbsg.org. (2008). [Online; accessed 25-April-2018].
[13] Damodaram Kamma and Sasi Kumar G. 2014. Effect of Model Based Software Development on Productivity of Enhancement Tasks – An Industrial Study. *2014 21st Asia-Pacific Software Engineering Conference* (2014), 71–77. https://doi.org/10.1109/APSEC.2014.20
[14] Anneke G Kleppe, Jos B Warmer, and Wim Bast. 2003. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional.
[15] Ekrem Kocaguneli, Tim Menzies, and Jacky W Keung. 2012. On the value of ensemble effort estimation. *IEEE Transactions on Software Engineering* 38, 6 (2012), 1403–1416.
[16] Adrian Kuhn, Gail C Murphy, and C Albert Thompson. 2012. An exploratory study of forces and frictions affecting large-scale model-driven development. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 352–367.
[17] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2014. Assessing the state-of-practice of model-based engineering in the embedded systems domain. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 166–182.
[18] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2016. Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Software & Systems Modeling* (2016), 1–23.
[19] Niklas Mellegård, Adry Ferwerda, Kenneth Lind, Rogardt Heldal, and Michel R. V. Chaudron. 2016. Impact of Introducing Domain-Specific Modelling in Software Maintenance: An Industrial Case Study. *IEEE Transactions on Software Engineering* 42, 3 (2016), 248–263. https://doi.org/10.1109/TSE.2015.2479221
[20] Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, Miguel A Fernandez, Bjørn Nordmoen, and Mathias Fritzsche. 2013. Where does model-driven engineering help? Experiences from three industrial cases. *Software & Systems Modeling* 12, 3 (2013), 619–639.
[21] Gunter Mussbacher, Daniel Amyot, Ruth Breu, Jean-Michel Bruel, Betty HC Cheng, Philippe Collet, Benoit Combemale, Robert B France, Rogardt Heldal, James Hill, et al. 2014. The relevance of model-driven engineering thirty years from now. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 183–200.
[22] Efi Papatheocharous, Stamatia Bibi, Ioannis Stamelos, and Andreas S Andreou. 2017. An investigation of effort distribution among development phases: A four-stage progressive software cost estimation model. *Journal of Software: Evolution and Process* 29, 10 (2017).
[23] Roger S Pressman. 2000. *Software engineering: a practitioner's approach (5th ed.)*. McGraw-Hill.
[24] Paul Ralph. 2018. The two paradigms of software development research. *Science of Computer Programming* (2018).
[25] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. 2012. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons.
[26] Bran Selic. 2012. What will it take? A view on adoption of model-based methods in practice. *Software & Systems Modeling* 11, 4 (2012), 513–526.
[27] Ian Sommerville. 2007. *Software Engineering 8*. Pearson Education.
[28] Marco Torchiano, Federico Tomassetti, Filippo Ricca, Alessandro Tiso, and Gianna Reggio. 2013. Relevance, benefits, and problems of software modelling and model driven techniquesâĂŤA survey in the Italian industry. *Journal of Systems and Software* 86, 8 (2013), 2110–2126.
[29] Ragnhild Van Der Straeten, Tom Mens, and Stefan Van Baelen. 2008. Challenges in model-driven software engineering. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 35–47.
[30] Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. 2017. A taxonomy of tool-related issues affecting the adoption of model-driven engineering. *Software & Systems Modeling* 16, 2 (2017), 313–331.
[31] Ye Yang, Mei He, Mingshu Li, Qing Wang, and Barry Boehm. 2008. Phase distribution of software development effort. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 61–69.
[32] Robert K Yin. 2017. *Case study research and applications: Design and methods*. Sage publications.
[33] Marvin V. Zelkowitz. 1978. Perspectives in Software Engineering. *ACM Comput. Surv.* 10, 2 (June 1978), 197–216. https://doi.org/10.1145/356725.356731