Leiden University
ICT in Business and the Public Sector

The data-driven development of a maturity model for machine learning teams

Name:	Ceyhan Deve
Student ID:	s1694294
Date:	31/01/2024
1st supervisor:	Prof.dr.ir. J.M.W. Visser
2nd supervisor:	dr. C.J. Stettina MSc

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA The Netherlands

Abstract

Background: Software systems are increasingly equipped with machine learning (ML) components. Due to the different nature of these ML-based software systems compared to traditional software systems, ML-based software systems require novel software engineering (SE) methods and guidelines in order to be developed robustly and responsibly. In literature, engineering best practices are described and grouped into collections to address the challenges that ML-based software systems pose. Nevertheless, the guidance and support of ML teams is limited regarding their current use of best practices and the improvement of their best practice adoption.

Objective: We aim to develop and validate a maturity model for ML-based software systems, which guides ML teams in assessing and improving their development processes based on a set of engineering practices.

Method: In order to develop and validate the maturity model for ML teams, we applied the design science methodology. We identified common steps in development frameworks for maturity models, which provided a direction for the development of the maturity model. The maturity model is created based on adoption data of these practices and existing maturity models in the field of software engineering. For the validation of the model, we conducted a case study involving multiple ML teams in order to determine whether it satisfies several criteria, such as comprehensibility and accuracy. Participating teams were asked to perform a maturity self-assessment with the model after which their opinion and feedback on the model were collected.

Results: We presented a maturity model for ML teams, which encompasses 45 engineering practices for the development of ML-based software systems. In the model, the practices are organised into two representations: the domain and maturity representation. The domain representation describes the maturity of a ML team on various aspects of their development process with the use of engineering capability (EC) levels. The maturity representation, however, focuses on the overall maturity of a team represented by maturity levels. The practices are ordered into the levels based, among others, on their adoption by 139 ML teams. Overall, we found during the model validation that ML teams perceive the model as comprehensible, easy to use, complete, correct and useful, despite minor issues. Teams, however, stated that the model does not represent their actual maturity completely and suggested altering the scoring method to allow for better comparison between teams. Moreover, some teams indicated that the practical use of the model is subject to the willingness of the teams themselves and support from the leaders of their organisations. Based on the obtained findings, we suggested some model refinements.

Conclusion: The development and validation of a maturity model for ML teams contribute to the robust and responsible development, deployment and maintenance of ML-based software systems by these teams. The developed model assists teams in obtaining insights into their development processes in terms of their practice adoption. The model and the ordering approaches devised during its development could serve as a basis for the operationalisation of practices and the development of other models in different domains. The model is positively received by ML teams. More research is, however, required into the creation of support from ML teams and stakeholders for the implementation of such a model in real-world settings.

Acknowledgements

The conduction of my master's thesis did not proceed as initially expected and was a challenging period of time for me. I was being faced with many unexpected external circumstances. Nevertheless, Prof.dr.ir. Joost Visser, my first supervisor, has always supported me throughout my master's thesis. I would like to express my deepest gratitude to him for his continued understanding, patience and flexibility. Without the guidance of Prof.dr.ir. Joost Visser, I would not have been able to complete my thesis. Furthermore, I would like to extend my sincere thanks to dr. Christoph Stettina MSc, my second supervisor, for providing new and insightful perspectives on my thesis at the end of each research phase.

Moreover, I would like to thank all the ML teams that participated in the case study, which was part of this master's thesis. These ML teams provided us with valuable practical insights that enabled us to validate our developed maturity model.

Finally, I am thankful to my parents and little brother for continuously being caring, supportive and encouraging during my entire studies.

Contents

1 Introduction **5**
1.1 Research problem . . . 5
1.2 Research question . . . 6
1.3 Methodology . . . 6
1.4 Outline . . . 7

2 Background **8**
2.1 Differences and challenges in ML . . . 8
2.2 Maturity models . . . 11
2.3 Engineering best practices in ML . . . 11

3 Design science **13**
3.1 The paradigm . . . 13
3.2 Design science framework . . . 13
3.3 Goals and problems . . . 14
3.4 Design cycle . . . 15
3.4.1 Problem investigation . . . 16
3.4.2 Treatment design . . . 16
3.4.3 Treatment validation . . . 17

4 Maturity models for software engineering **18**
4.1 Traditional software development models . . . 18
4.2 Agile software development models . . . 20
4.3 Software security models . . . 22
4.4 FLOSS development models . . . 23
4.5 Comparison of maturity models . . . 24

5 Design approaches **26**
5.1 Approaches for general models . . . 26
5.2 Approaches for specific models . . . 29
5.3 Common development phases . . . 30

6 Model development **33**
6.1 The scoping of the model . . . 33
6.2 The construction of the model . . . 35

6.2.1 The dataset . . . 35
6.2.2 The creation of a data pool . . . 35
6.2.3 The adoption of the practices . . . 37
6.2.4 The structure of the model . . . 38
6.2.5 The creation of the domain representation . . . 39
6.2.6 The creation of the maturity representation . . . 45
6.2.7 The application of the model . . . 50
6.2.8 Consistent and objective assessments . . . 51
6.2.9 The design decisions . . . 53

7 Model validation **54**
7.1 Case study design . . . 54
7.2 Results . . . 55
7.2.1 The participants . . . 55
7.2.2 The interviews . . . 56
7.2.3 The questionnaire . . . 60
7.2.4 Requirement satisfaction . . . 61
7.2.5 Model refinements . . . 62

8 Discussion **64**
8.1 The maturity model . . . 64
8.2 The model validation . . . 65
8.3 The knowledge questions . . . 66

9 Conclusion **69**

References **71**

A The domain representation **77**

B The maturity representation **80**

C The application guide **82**

D The practice cards **84**

E The interview guide **88**

F The questionnaire **90**

Chapter 1

Introduction

1.1 Research problem

In recent years, the use of machine learning (ML) has become prevalent in practical and industrial settings. Organisations increasingly adopt ML components in their software systems (Liu et al., 2020). These ML components can be applied to a wide variety of complex real-world problems, such as fraud detection, product recommendation and image recognition.

ML components have different characteristics than traditional software components (Ishikawa & Yoshioka, 2019). For example, ML components are more data-intensive. The behaviour of ML components is not directly programmed, but is acquired through learning on data (i.e. training data). Moreover, these components continuously evolve as a result of changes in input data. Furthermore, the outputs of ML components are uncertain for unseen data, data that was not part of the training data.

Due to different characteristics of ML components, the development process of ML-based software systems, i.e., software systems with these ML components, differs from that of traditional software systems (Nascimento et al., 2019). The development process of these systems involves, for example, different phases and tasks, such as data pre-processing, feature engineering and model training. The emergence of ML components in software systems has introduced a different paradigm of software development. Software engineering (SE) methods and guidelines for traditional software systems are mostly not sufficient for ML-based software systems. Therefore, in order to robustly and responsibly develop, deploy and maintain these systems, novel methods and guidelines are required (Ishikawa & Yoshioka, 2019; Lwakatare et al., 2019). Some of such methods and guidelines have already been presented in literature, such as DeepXplore (Pei et al., 2019), a white-box framework to systematically test deep learning (DL) systems.

The different nature of ML-based software systems makes the development, deployment and maintenance of these systems challenging (Ishikawa & Yoshioka, 2019; Lwakatare et al., 2020; Nascimento et al., 2019). In order to address the challenges that ML-based software systems pose, engineering best practices have started to emerge in both grey and academic literature (e.g. Amershi et al., 2019; Breck et al., 2017). Nevertheless, guidance and support of ML practitioners during the development process of ML-based software systems is still limited. In particular, ML practitioners lack insights into their current use of best practices and the improvement of their adoption.

1.2 Research question

As guidance and support of ML practitioners is limited, in this thesis we develop and validate a maturity model for the development processes of ML teams, teams that develop ML-based software systems. Maturity models are tools that assist in determining the maturity of objects (i.e. organisations, processes or people) along a defined evolution path of discrete stages (Becker et al., 2009; de Bruin et al., 2005; Otto et al., 2020). The maturity of an object under consideration could refer to the level of performance, effectiveness or competence of that object.

In literature, a few maturity models in the field of ML have already been proposed (e.g. Akkiraju et al., 2020; John et al., 2021; Mateo-Casalí et al., 2023). Most of these ML maturity models address the operationalisation of ML-based software systems within organisations (e.g. integrating ML-based software systems within manufacturing operations). In contrast, our maturity model will focus on the development processes of these systems on a team level.

Our maturity model will be developed based on a set of engineering best practices that is defined in the studies of Serban et al. (2020, 2021). In these studies, the researchers determine the state of the art of the development process of ML-based software systems based on the compilation of a catalogue of practices from literature, which is referred to as the SE4ML catalogue[1]. During the validation of the maturity model, we collect practical insights in order to propose further refinements of the model.

With the development and validation of a maturity model, we aim to provide teams with a straightforward framework that enables teams to assess their development processes and offers support and guidance in improving these processes. The underlying research question (RQ) is thus as follows:

RQ: How can the development processes of ML teams be assessed based on engineering best practices in order to provide these teams with guidance and support in improving their processes?

1.3 Methodology

In this study, we will use the design science methodology, which is a research paradigm that aims to enhance technical and scientific knowledge based on the development of artifacts that address problems and improve aspects of their contexts (Vom Brocke et al., 2020). In order to develop and validate a maturity model for ML teams, we will follow the (iterative) design cycle within the design science methodology, which consists of three tasks: problem investigation, treatment design and treatment validation.

For the development of the maturity model, common phases of model development frameworks in academic literature will be defined and followed. To validate the model and determine its relevance and accuracy, a case study involving the development processes of ML teams will be conducted. In this case study, ML teams will be asked to assess their development processes with the use of the maturity model, after which these teams will be interviewed about the application and the content of the developed model. Furthermore, teams will be asked to complete a questionnaire in which they are asked to rate the model on a set of criteria.

[1]https://se-ml.github.io/practices/

1.4 Outline

This thesis is structured as follows. Chapter 2 provides background information on the differences between traditional software development and ML development, and the challenges that ML-based software systems pose. In addition, background information is provided on maturity models and engineering best practices for ML-based software systems. Chapter 3 elaborates on the applied method, i.e., the design science methodology. Chapter 4 discusses different maturity models in the field of SE. Chapter 5 focuses on design approaches for maturity models and states the common phases in the development of the models. Chapter 6 describes the development of our maturity model while Chapter 7 presents the design of the validation of the model and the obtained findings. The model development and the findings of the model validation are discussed in Chapter 8. Finally, Chapter 9 concludes the thesis and provides recommendations for future research.

Chapter 2

Background

The following sections provide relevant background information on the differences between the development of traditional software systems and ML-based software systems, challenges in the field of ML, the notion of maturity models and the engineering best practices identified during the studies of Serban et al. (2020, 2021).

2.1 Differences and challenges in ML

As stated before, ML-based software systems are different than traditional software systems. Due to the different nature of ML-based software systems, the development, deployment and maintenance of these systems is challenging. In this section, we elaborate on the differences between the development process of traditional software systems and ML-based software systems and the development challenges that ML-based software systems pose in order to provide more context to the research problem described in Section 1.1.

A study that focuses on the differences between the development of ML-based software systems and traditional software systems was conducted by Wan et al. (2021). In order to determine the differences between the development of these two types of systems, the researchers followed a mixed method approach, i.e., a combination between qualitative and quantitative data collection methods. They interviewed several software practitioners with experience in both ML development and non-ML development. Based on these conducted interviews, they were able to define 80 statements that describe the differences between the two types of systems. These defined statements were refined and reduced to 31 statements during three focus groups (i.e. group interviews). The statements were subsequently used in a questionnaire among software practitioners. For each statement, the practitioners had to determine their degree of agreement with that statement on a 5-point Likert scale ranging from strongly disagree to strongly agree.

Wan et al. (2021) found that, in contrast to traditional software systems, the validation of requirements (i.e. determining whether requirements satisfy the requests of stakeholders) of ML-based software systems requires several preliminary experiments in which different ML algorithms are tested. As the performance of ML-based software systems might degrade after a period of time in production, the requirements should incorporate this degradation of performance. ML-based software systems are expected to be capable of detecting the performance degradation, and adapting to this degradation by incorporating new data or a complete new model. In addition, the researchers found that the detailed design of ML-based software systems is more time-consuming compared to traditional software systems and is performed in a more iterative manner. Furthermore, a good performance of ML-based software systems on test data does not guarantee the performance of these systems in production, which depends to a large extent on the similarities between the training data and the incoming data. Moreover, the researchers found that the capabilities of ML-based software systems largely depend on the processing of the data.

Besides the differences between the development processes of traditional systems and ML-based software systems, Wan et al. (2021) identified differences between these two types of systems in terms of work characteristics. The researchers found, for example, that developers of ML-based software systems require knowledge of mathematics, information theory and statistics. In addition, they found that it is more difficult to create an accurate plan for the development of ML-based software systems and there are no clear roadmaps for the development of effective ML components. Lastly, ML practitioners face more challenges in the communication with customers than traditional software practitioners as customers tend to find it difficult to interpret the results of the development of ML-based software systems.

Similar to Wan et al. (2021), Liu et al. (2020) address the differences between the development of traditional software systems and ML-based software systems. However, Liu et al. (2020) focus on emerging and changing tasks in the software development process when ML components are integrated in software systems. In order to identify these tasks, the researchers interviewed ML practitioners from both industry and academia. In total, they found 25 tasks that are related to different software development phases. Some of the tasks that emerge or change during the development of ML-based software systems are: data exploration and understanding, data cleaning, feature extraction, algorithm selection and criteria (i.e. performance metrics) selection.

In academic literature, several studies address the development challenges of ML-based software systems. Ishikawa and Yoshioka (2019), for example, identified engineering challenges of ML-based software systems and their causes (in general) based on a questionnaire among ML practitioners from Japan. They found that practitioners consider the decision making with customers and testing and quality assurance as the most difficult activities in the engineering of ML-based software systems. Furthermore, the researchers found that according to the practitioners the posed challenges by the systems mostly originate from the lack of test oracle (i.e. source of information that indicates whether systems behave correctly in terms of their outputs for given inputs), the imperfection of the systems and their behavioural uncertainty for unseen data. In the questionnaire, Ishikawa and Yoshioka (2019) asked the practitioners to motivate their provided answers. Based on the comments of the practitioners, the researchers were able to obtain further insights into their findings. Regarding the decision making with customers, they found, among other things, that ML practitioners stated that there is a gap in the understanding of ML between engineering teams and their customers. In addition, they found that customers have unrealistic expectations in terms of the functionality and the performance (e.g. the accuracy) of the systems. Regarding testing and quality assurance, the researchers found that the nondeterministic nature of the systems is perceived difficult by practitioners. Furthermore, the concept of test coverage is difficult to understand and apply effectively in the context of ML.

Similar to Ishikawa and Yoshioka (2019), Lwakatare et al. (2020) identified challenges and their corresponding solutions for the development and maintenance of ML-based software systems based on a systematic literature review. The identified challenges were organised across two dimensions: the ML development life-cycle and quality attributes. In the study, Lwakatare et al. (2020) distinguish the following four quality attributes: adaptability, scalability, privacy and safety. In total, the researchers found 23 challenges and 8 solutions. Most of these identified challenges are related to the adaptability and scalability of the systems. The most occurring challenges regarding the adaptability in literature are unstable data dependencies and data quality problems, and narrow evaluation metrics. In a continuously evolving system, training data is created in a development pipeline from a continuous stream of input data which, originates from sources with different qualities. These changes in input data could lead to a degradation of the performance of the system. Furthermore, traditional ML metrics (e.g. accuracy and precision) do not capture the business impact of the systems and are, therefore, considered insufficient to evaluate their overall performance. In order to address unstable data dependencies and data quality problems, Lwakatare et al. (2020) found that practitioners employ data validation tools, i.e., tools that check and monitor the quality of data. In order to deal with narrow evaluation metrics, they identified that tests and development strategies can be used that assist practitioners in determining whether ML-based software systems are ready to be deployed in production. Moreover, A/B model experiments, i.e., online experiments in which the performance of different variants of ML components of the systems are evaluated, can be conducted. In these experiments, the performances of these components are also evaluated on different defined business metrics (i.e. KPIs).

Regarding the scalability of the systems, challenges related to balancing the efficiency and effectiveness in ML end-to-end development processes, and design of a ML infrastructure are most frequently reported on in literature (e.g. Hazelwood et al., 2018; Polyzotis et al., 2018; Raeder et al., 2012). The researchers found that these challenges can be addressed by the use of ML frameworks and platforms, which assist practitioners, among other things, in the quality control and maintenance of ML-based software systems.

Another study that identified challenges of the development of ML-based software systems is conducted by Nascimento et al. (2019). In contrast to the other studies, Nascimento et al. (2019) focused especially on challenges that ML practitioners face during the development of ML-based software systems in small software companies. In order to identify the challenges, the researchers interviewed seven ML practitioners from three small software companies in Brazil. Based on these interviews, they found three main challenges of the development of ML-based software systems: the identification of business metrics of customers, the lack of a defined development process and the design of a structure for the required database. In order to address these identified challenges, Nascimento et al. (2019) proposed two checklists: CheckBM and CheckDP. CheckBM focuses on the identification of customer's business metrics while CheckDP addresses both the lack of a defined development process and the design of database structure. In order to create these checklists, the researchers used literature and practices used and known by the interviewed practitioners. Each checklist consists of a set of tasks with corresponding verification criteria (i.e. items) that need to be checked. For CheckBM, for example, Nascimento et al. (2019) determined tasks that are performed by the practitioners during the identification of customer's business metrics. For each task, they defined validation criteria related to a particular task.

2.2 Maturity models

Maturity models are tools that describe the maturity of objects (e.g. organisations, processes or people) as an evolution path of discrete stages (Becker et al., 2009; Otto et al., 2020). The maturity of a particular object could, for example, be defined as the performance, competence or effectiveness of that object. The sequence of discrete stages is referred to as the maturity levels. Van Steenbergen et al. (2010) refer to maturity models that distinguish a particular number of consecutive maturity levels as fixed-level maturity models.

The lowest maturity level mostly represents the initial maturity state of an object under consideration while the highest maturity level represents a state of total maturity of that object. For each maturity level, a model provides characteristics and criteria that need to be satisfied in order to reach that level. Based on these characteristics and criteria, an assessment with a maturity model could be conducted in order to determine the current maturity level of the object under consideration (i.e. the position of the object along the defined evolution path).

According to Pöppelbuß and Röglinger (2011), three *purposes of use* of maturity models can be distinguished:

- **Descriptive:** A maturity model with a descriptive purpose of use assists in determining the current maturity state (i.e. as-is situation) of the object under consideration based on an assessment regarding the given characteristics and criteria.
- **Prescriptive:** A maturity model with a prescriptive purpose of use assists in identifying areas of improvement and provides specific measures to guide these improvements.
- **Comparative:** A maturity model with a comparative purpose of use allows for comparing the object under consideration with other assessed objects based on historical assessment data.

These purposes of use are not mutually exclusive. For example, a descriptive maturity model could also have a comparative purpose of use. Furthermore, a model with a prescriptive purpose of use should also contain all characteristics of a descriptive model.

Since the introduction of the Capability Maturity Model (CMM; Humphrey, 1988), a maturity model for software development processes, in 1988, maturity models have gained significant popularity and a large number of models have been proposed in different domains ranging from software engineering to risk management (Pöppelbuß & Röglinger, 2011).

2.3 Engineering best practices in ML

Our maturity model for ML teams will be developed based on a set of engineering best practices for ML-based software systems, which is referred to as the SE4ML catalogue. Most of the practices (i.e. 29 of the 45 best practices) in this catalogue are identified during a study by Serban et al. (2020). In this study, Serban et al. (2020) aimed to determine how teams develop, deploy and maintain ML-based software systems based on the identification of best practices from grey and academic literature. The identified practices consist of traditional SE practices, modified SE practices to be suitable for ML-based software systems and new practices designed specifically for these systems. The practices are divided into six aspects related to the ML development life-cycle: data, training, deployment, coding, team and governance.

Besides the identification of best practices, the researchers determined the adoption of the identified practices and validated their perceived effects (agility, software quality, team effectiveness and traceability) through a questionnaire among ML practitioners (especially ML teams). Based on the obtained responses, Serban et al. (2020) ranked the practices on their degree of adoption and tested several relationships between the practices and their perceived effects using statistical models.

The rest of the practices in the catalogue are identified in a study by Serban et al. (2021), which complements their previously discussed work. This follow-up study focused on bridging the gap between proposed guidelines for the development of trustworthy ML (i.e. the development of ML in an ethical and robust manner) and operational practices for practitioners. In order to bridge this gap, the researchers identified practices for trustworthy ML based on a review of grey and academic literature. Similar to their previous study, Serban et al. (2021) also determined the adoption of the identified practices by practitioners through a questionnaire. The used questionnaire in this follow-up study is an extension of the questionnaire of the initial study with questions related to newly identified practices for the development of trustworthy ML. The extended questionnaire measures thus both the adoption of the initial set of practices and the set of practices for trustworthy ML.

Chapter 3

Design science

3.1 The paradigm

In this study, we apply the design science methodology as described by Wieringa (2014). Design science is the design and study of artifacts in particular problem contexts, i.e., contexts in which problems are situated (e.g. organisations, business processes and people). Artifacts in design science are, for example, methods, frameworks and algorithms. The designed artifacts interact with problem contexts in order to improve some aspects of these contexts and address the problems. With the development and study of artifacts in problem contexts, design science aims to enhance scientific and technical knowledge.

In this case, the artifact is a maturity model for ML-based software systems and the problem context with which the model interacts, is the development processes of ML teams.

3.2 Design science framework

The two parts of design science, the design and study of artifacts, are connected to a social and knowledge context, and together form the design science framework. This conceptual framework, which represents the entire context of design science research, is shown in Figure 3.1. The social context consists of stakeholders who affect the research or are affected by it, such as sponsors of the research and users of the artifact. The knowledge context comprises all knowledge available prior to design science research, such as scientific theories, design specifications of existing artifacts and lessons learned from previous design science research. Design science research uses this knowledge and may expand it with the development of a new artifact or answering knowledge questions. In Section 3.3, we elaborate on knowledge questions.

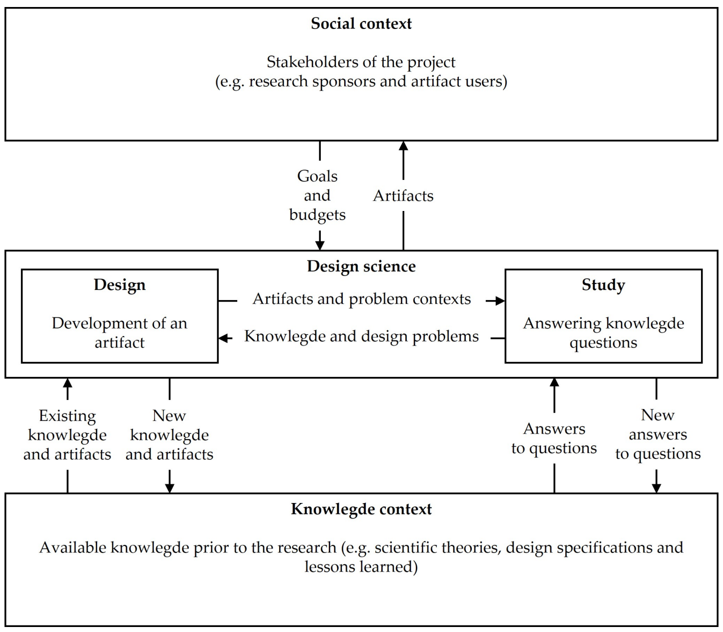

Social context
Stakeholders of the project
(e.g. research sponsors and artifact users)
Goals and budgets
Artifacts
Design science
Design
Development of an artifact
Artifacts and problem contexts
Knowlegde and design problems
Study
Answering knowlegde questions
Existing knowlegde and artifacts
New knowlegde and artifacts
Answers to questions
New answers to questions
Knowlegde context
Available knowlegde prior to the research (e.g. scientific theories, design specifications and lessons learned)

Figure 3.1: The design science framework. The framework illustrates all concepts, entities and relationships of design science research. Figure is based on Wieringa (2014).

3.3 Goals and problems

Design science research has two kinds of goals: design goals and knowledge goals. Design goals can be divided into instrument and artifact design goals. Instrument design goals are the goals to design research instruments during the research, while artifact design goals are the goals to design artifacts. The instrument design goals could support the achievement of knowledge goals, which are the goals related to the creation of knowledge with the research. The artifact design goals contribute to the improvement of the problem context, which in turn supports the achievement of the goals of external stakeholders. The improvement of the problem context and the goals of external stakeholders form the goals in the social context of design science research.

The design and study of artifacts are related to two research problems, namely design problems and knowledge questions. A design problem, also referred to as a technical research question, connects the problem context with the external stakeholder goals and requires the development of an artifact such that its interaction with the problem context contributes to the achievement of the goals of external stakeholders. A design problem is thus the problem to design an artifact in order to improve a problem context and is related to an artifact design goal of design science research. Knowledge questions are empirical or analytical questions about the artifact, its problem context and the interaction between them. These knowledge questions are derived from the knowledge goals of design science research.

In this study, we define the design problem, which is a transformation of the defined research question, as follows: **improve the development processes of ML teams by developing a maturity model for ML-based software systems that operationalises engineering best practices in order to provide these teams with guidance and support during their development processes**. The artifact design goal that corresponds to this design problem is to **develop a maturity model for ML-based software systems**.

The knowledge questions (KQ) with their underlying goals (G) are defined as follows:

KQ1: *What are the current maturity models in the domain of SE?*
KQ2: *What approaches are available for the development of maturity models?*
KQ3: *How can the adoption of engineering best practices in the development processes of ML teams be operationalised?*
KQ4: *How accurate and relevant is the created maturity model for the development processes of ML teams?*

G1: Obtain insights into the content and structure of available maturity models in the domain of SE.
G2: Define the common phases in the development of maturity models.
G3: Determine how the adoption of engineering best practices can be measured in an objective and general manner.
G4: Determine to which extent the developed maturity model helps ML teams with assessing and improving their development processes in practice.

We omit the instrument design goals of our study in this section as these goals do not correspond to one of the research problems of design science and are indirectly attained when other defined goals are attained.

3.4 Design cycle

Design problems are addressed by following the design cycle, which is shown in Figure 3.2. The design cycle, which is a part of the larger engineering cycle, comprises three tasks: problem investigation, treatment design and treatment validation. A treatment can be defined as an artifact that interacts with the problem context. In the engineering cycle, a validated treatment, which is the result of the design cycle, is implemented in a real-world setting and evaluated on its effectiveness. Design science research often iterates several times over the design cycle.

Although the design cycle provides a sequence of tasks, the cycle does not prescribe how to actually perform these tasks. The design cycle only indicates that the problem should be identified, the treatment should be designed and validated in order to address a design problem.

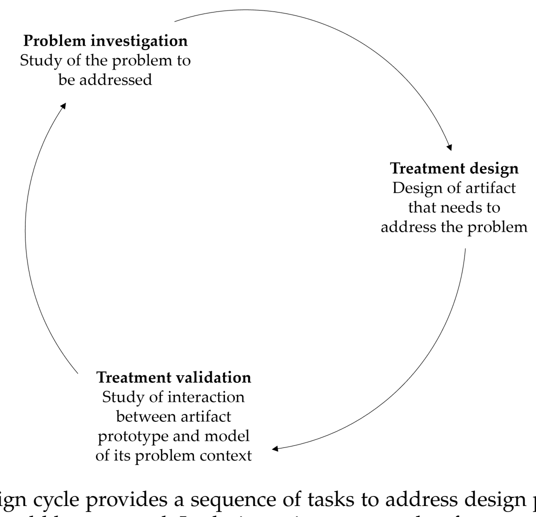

Problem investigation
Study of the problem to be addressed
Treatment design
Design of artifact that needs to address the problem
Treatment validation
Study of interaction between artifact prototype and model of its problem context

Figure 3.2: The design cycle. The design cycle provides a sequence of tasks to address design problems. Nevertheless, the cycle does not prescribe how these tasks should be executed. In design science research, often several iterations through the design cycle are performed.

In this study, we perform one iteration over the design cycle in order to develop and validate the maturity model for ML-based software systems. In the following sections, we elaborate on each task of the cycle.

3.4.1 Problem investigation

The problem investigation is the study of a problem, which serves as a preparation for the design of the artifact. During the problem investigation, the problem to be addressed is defined and the stakeholder goals are identified. As part of the study of a problem, the aspects of the problem, their causes and effects on stakeholders goals are determined.

3.4.2 Treatment design

During treatment design, the artifact that needs to address the problem is designed (i.e. the treatment). Prior to the design of a treatment, requirements (R), desired properties of a treatment from the perspective of stakeholders, are specified. In design science, specified requirements, are justified with contribution arguments (CA), which argue that the treatment contributes to stakeholders goals in case it satisfies the specified requirements. A requirement could have several contribution arguments and contribute as a result to several stakeholder goals. As contribution arguments argue that the interaction between an artifact and its problem context contributes to stakeholder goals, these contribution arguments can be considered as predictions.

We define the requirements for the maturity model that we develop and validate with corresponding contribution arguments as follows:

R1: The maturity model should provide ML teams with insights into the current state of their development processes based on their adoption of best practices.

CA1: If teams are able to provide their adoption of best practices accurately and the maturity model is able to provide teams with insights into the current state of their processes based on their adoption of practices, then the model enables teams to assess their processes based on best practices.

R2: The maturity model should guide ML teams to their desired state of their development processes.

CA2: If the maturity model guides teams to their desired state of their processes and teams are able to specify their desired state, then the model enables teams to plan and execute the improvement of their processes.

R3: The maturity model should be understandable and applicable with minimal effort (i.e. at most three hours to understand the model and conduct an assessment) by ML teams.

CA3: If the maturity model is understandable and applicable with minimal effort (i.e. at most three hours), and teams have a particular proficiency in ML and sufficient insight into the used approaches in their processes, then the model enables teams to assess and improve their processes in a straightforward way.

3.4.3 Treatment validation

The validation of a treatment focuses on determining whether the artifact addresses the defined problem. As real-world implementations of a designed artifact do not (yet) exist, a prototype of the artifact and a model of its problem context, which simulate a real-world implementation, are used to validate an artifact. A prototype of an artifact, which interacts with a model of its problem context is referred to as a validation model. During treatment validation, the interaction between the prototype of the artifact and a model of its problem context is studied in order to predict what the effects would be and whether the defined requirements are satisfied in case the artifact is implemented in a real-world setting. In our study, our maturity model will be prototyped in Excel. The development processes of ML teams that will participate in our case study for the validation of the model will serve as models of its problem context.

Chapter 4

Maturity models for software engineering

Various maturity models for different application domains (e.g. IT management and process management) have been proposed in academia and industry. In the following sections, we discuss and compare existing maturity models in order to determine their design, which can serve as a guide for the construction of our own model. As we develop a maturity model for the development processes of ML teams, we focus on maturity models that guide the improvement of processes in the field of software engineering. We distinguish four types of such maturity models: traditional software development models, agile software development models, software security models and Free/Libre Open Source Software (FLOSS) development models.

We exclusively discuss maturity models of which the design or development is documented extensively.

4.1 Traditional software development models

The Capability Maturity Model Integration for Development

One of the most applied maturity models for software process improvement in practice is the Capability Maturity Model Integration for Development (CMMI-DEV), which was developed by industry, the US government and the Software Engineering Institute (SEI; CMMI Product Team, 2010; Fontana et al., 2015). CMMI-DEV guides organisations in improving their processes for the development of products and services based on best practices. CMMI-DEV comprises 22 process areas, which are divided into four categories: process management, project management, engineering and support. A process area is a set of related practices in the corresponding area that, when adopted, satisfies goals that are considered significant for improving that area. Each process area contains three model components: required, expected and informative components. Required components are generic and specific goals that are essential for improving the process in a particular process area. The expected components are generic and specific practices that are significant for satisfying generic goals and specific goals (the required components), respectively. Informative components are components that assist model users in understanding these goals and practices, such as explanations, notes and example work products. Generic goals and practices apply to several process areas while specific goals and practices correspond to one particular process area.

CMMI-DEV distinguishes two improvement paths: the continuous and staged representations. The continuous representation enables organisations to improve their processes regarding a set of selected process areas while the staged representation enables organisations to improve their processes incrementally by addressing consecutive sets of predefined process areas. In the continuous representation, capability levels indicate the state of processes of organisations regarding a process area. The following capability levels are defined: 0) Incomplete, 1) Performed, 2) Managed and 3) Defined. The staged representation, however, uses maturity levels that represent the overall state of processes of organisations. The following maturity levels are distinguished: 1) Initial, 2) Managed, 3) Defined, 4) Quantitatively managed and 5) Optimizing. The first levels of both representations represent the initial state of the processes of organisations.

The Software Engineering Maturity Model

Another maturity model for software process improvement is the Software Engineering Maturity Model (SEMM), which is intended for small enterprises in the Spanish software industry (Garzás et al., 2013). The model was developed by academia, industry and the Spanish government under supervision of the Spanish Association for Standardization and Certification (AENOR) in order to improve the quality of software development in these enterprises. SEMM comprises three components based on several ISO/IEC standards (i.e. international norms): an assessment model for process capability and organisational maturity, a software lifecycle model and an auditing process.

The structure of the assessment model is based on ISO/IEC 15504-7, which provides a framework for the assessment of the maturity of organisations. The assessment model defines four maturity levels of organisations: 0) Immature, 1) Basic, 2) Managed and 3) Established. For each maturity level (except level one), a set of processes (e.g. project planning process and software integration process) is defined based on studies on software process improvement in small enterprises and ISO/IEC 12207, which describes the development process of software systems. Within the assessment model, a set of process attributes is adopted from ISO/IEC 15504-5, which represents required characteristics for the institutionalisation of the defined processes. This set of process attributes apply to all processes. The process attributes are divided into three capability levels: 1) Performed, 2) Managed and 3) Established.

These process attributes consists of general practices and work products. The set of general practices of a process attribute should be adopted in order to satisfy the purpose of that particular process attribute. The general practices are adaptations of practices in an exemplary assessment model presented in ISO/IEC 15504-5. Besides the general process components (i.e. process attributes), processes contain process specific components: outcomes and activities. The outcomes and activities of processes are defined in the software lifecycle model of SEMM, which is based on ISO/IEC 12207. The outcomes are required characteristics for the implementation of the process. Activities represent the tasks that could be performed in order to fulfil these characteristics.

The maturity of an enterprise is determined based on its achieved capability levels of all processes in the assessment model. In order to determine the capability level of a process, each process attribute is evaluated with the use of a scale, which indicates the degree of achievement of components corresponding to a particular process attribute in terms of percentage intervals. The scale distinguishes the following achievement degrees: not achieved, partially achieved, largely achieved and fully achieved. Process attributes corresponding to a particular capability level should be at least partially achieved and process attributes of any lower levels should be fully achieved in order to achieve that capability level. The maturity level corresponding to the obtained capability levels for the processes is determined based on some derivation rules. For example, an enterprise achieves maturity level 2 in case the enterprise achieves capability level 2 for all processes corresponding to this maturity level.

4.2 Agile software development models

As most traditional software development models define maturity as the institutionalisation of well-defined processes, which is not usual in agile software development environments, the achievement of a high degree of maturity is not compatible with maintaining agility (Fontana et al., 2014). Hence, traditional software development models are not directly applicable to agile environments.

In the field of agile software development, two types of studies address the accomplishment of maturity while maintaining the benefits of agile methods (Fontana et al., 2018; Fontana et al., 2015). The first group of studies focuses on the adaptation of agile practices and principles in order be suitable for traditional software development models, such as CMMI-DEV. The second proposes novel maturity models for agile software development. In the context of this study, we focus on the second group of studies.

The agile adoption framework

Sidky et al. (2007) presented an agile adoption framework, which guides and assists organisations in agile adoption. The agile adoption framework comprises two components: the Sidky Agile Measurement Index (SAMI) and a four-stage assessment process. SAMI identifies the agile potential (i.e. the degree of agility that can be accomplished) of projects and organisations. The four-stage process assists in determining whether organisations are ready for agile adoption and what set of agile practices could be adopted based on their agile potential.

SAMI defines five agile levels, which represent degrees of agility of an organisation or project: 1) Collaborative 2) Evolutionary 3) Effective 4) Adaptive and 5) Encompassing. The agile levels of SAMI correspond to qualities identified from the agile manifesto. Agile levels consist of practices. The practices of each level are divided over a set of agile principles, which represents characteristics of an agile development process (e.g. human centric and technical excellence). A principle indicates the manner in which practices contribute to the achievement of an agile quality of a particular level. Each agile practice corresponds to required organisational characteristics (in the form of questions) for the successful adoption of that practice. For each organisational characteristic, a set of indicators is defined, which assesses a particular characteristic. The indicators corresponding to required organisational characteristics for an agile practice are used to determine to which extent an organisation is ready to adopt that practice.

The four-stage process contains the following stages: identification of discontinuing factors, project level assessment, organisational readiness assessment and reconciliation. In the first stage, a pre-assessment is conducted that identifies the presence of discontinuing factors, i.e., factors that could prevent the successful adoption of agile practices, in an organisation. In case the pre-assessment indicates that these factors are not present, an organisation could continue with the agile adoption initiative. In the second stage, SAMI is used to determine the highest level of agility that a project of an organisation can accomplish, which is referred to as the target level. In order to determine the target level, the assessment with SAMI focuses on limiting agile practices, i.e., practices that depend on the presence of factors (project characteristics) that are outside the control of a project or organisation for their adoption. The assessment identifies to which extent these required factors associated with limiting agile practices are present. The target level is the first agile level corresponding to a limiting agile practice of which one of its factors is identified to be absent.

In the third stage, the extent to which an organisation is ready to accomplish the target level is determined. In order to determine the organisation's readiness, the organisational characteristics corresponding to the agile practices up to the target level are assessed based on their presence. The highest level of agility that an organisation can accomplish depends on its readiness to adopt the practices up to the target level, which is referred to as the organisational readiness level. The organisational readiness level is the first agile level corresponding to an agile practice of which one of its required organisational characteristics is absent. In the final stage, the differences between the target level and organisational readiness level are reconciled in order to determine the set of agile practices that will be adopted in the project. For example, an organisation could determine to improve the presence of some organisational characteristics that prevent the accomplishment of the target level in case its organisational readiness level is lower than the target level.

The Scaled Agile Framework Maturity Model

Another framework that focuses on agile software development is the Scaled Agile Framework Maturity Model (SAFe MM), which was proposed by Turetken et al. (2017). SAFe MM guides organisations in the adoption of SAFe and assesses its adoption level. SAFe provides values, principles and practices in order to enable large organisations to scale agile and obtain business agility.

SAFe MM was developed in two main steps: the development and refinement of an initial model. In order to develop the initial version of SAFe MM, Turetken et al. (2017) adapted and extended SAMI with SAFe practices. After the development of the initial model, the initial model was refined through a Delphi study, i.e., a research approach, which aims to establish consensus on a particular subject by allowing experts to give their opinions and reconsider these opinions based on the opinions of other experts. In the Delphi study, a panel of seven SAFe and agile experts from industry reviewed the initial model. The conducted Delphi study consisted of two rounds. In the first round, the panel of experts assessed each practice in the model in order to determine whether it should remain unchanged, be changed (e.g. its description and position in the model) or removed. Moreover, the panel was asked to answer detailed questions regarding the completeness and alignment of the content of the model. In the second round, the panel of experts was requested to reach consensus on their proposed improvements based on the results of the first round.

The final version of SAFe MM consists of three types of practices: agile practices from SAMI, modified agile practices from SAMI and SAFe practices. Similar to SAMI, the achievement of each practice in SAFe MM is assessed with the use of indicators corresponding to characteristics of that practice. For the SAMI practices, the original indicators are used. In order to assess the achievement of a practice, all corresponding indicators are rated with a scale derived from ISO/IEC 15504. This scale has the following options: not achieved, partially achieved, largely achieved and fully achieved. In contrast to SAMI, the assessment results are not aggregated into a single maturity level.

4.3 Software security models

The Building Security in Maturity Model

A well-established, data-driven maturity model for software security is the Building Security in Maturity Model (BSIMM; Synopsys, 2022), which quantifies performed activities during software security initiatives (SSIs) of various organisations. These organisations form the BSIMM community. Based on the quantified software security activities of organisations that are part of the community, BSIMM aims to assists other organisations in evaluating and planning their own SSIs. As BSIMM focuses on activities that are actually performed during SSIs, the model evolves annually. SSIs are organisational-wide programs to coordinate the establishment, measurement and evolution of software security activities.

BSIMM organises the performed software security activities across the BSIMM community into a software security framework (SSF). The SSF distinguishes four domains: governance, intelligence, the secure software development lifecycle (SSDL) touchpoints and deployment. The domains consists of practice areas (e.g. attack models and security testing). Each practice area encompasses the corresponding security activities. The security activities of a practice area within a domain are divided across three levels based on their observation frequencies in BSIMM community. Level 1 activities are the most commonly performed activities, level 2 activities are performed less frequently and level 3 activities are rarely preformed across the community.

The results of an assessment with BSIMM are, among others, a scorecard, which reflects the current state of an assessed SSI of an organisation relative to the SSIs of the BSIMM community. The scorecard presents which of the software security activities observed across the community are performed during the SSI. Each software security activity in the scorecard is accompanied by its observation frequency. The scorecard enables organisations to understand the state of their SSIs, communicate the state with stakeholders (e.g. customers and executives), identify gaps against expectations, prioritise identified gaps and create improvement plans.

The Software Assurance Maturity Model

Another well-known maturity model that focuses on software security is the Software Assurance Maturity Model (SAMM), which guides organisations in assessing and improving their software security posture (i.e. the ability to develop and deploy secure software; OWASP, n.d.-a).

SAMM encompasses 15 security practices, which are divided over five business functions: governance, design, implementation, verification and operations. Each security practice consists of activities, which are separated in two streams. The streams of a security practice represent particular aspects of that practice with corresponding high-level objectives. The activities corresponding to a security practice are arranged in three maturity levels. Each maturity level consists of a specific objective, activity, and assessment question with corresponding answer options and quality criteria, which depend on the stream.

SAMM is usually applied through a sequence of six phases: preparation, assessment, target setting, planning, implementation and roll-out (OWASP, n.d.-b). In the preparation phase, the scope of the improvement initiative is defined (e.g. the entire organisation, a particular software system, project or team) and relevant stakeholders are identified. Subsequently, a maturity self-assessment is conducted by answering the questions corresponding to the maturity levels based on an evaluation of the current state of practices within the defined scope. The maturity self-assessment identifies the maturity of the scope regarding each security practice of SAMM. In the target setting phase, a target for the improvement initiative is set by determining which activities should be implemented in order to improve maturity of the scope. Moreover, the impact of the set target on the organisation is determined.

After the target is set, a roadmap for the implementation of the selected activities is created. In the roadmap, the activities are divided over a number of phases with a particular duration. In the implementation phase, the set of selected activities are implemented based on the created roadmap. Finally, the implemented improvements are made visible across the organisation through trainings and communication with management. Furthermore, the effectiveness of the improvements in terms of impact and usage are identified.

4.4 FLOSS development models

FLOSS projects, projects in which software is developed that is free to execute, study, modify and distribute, are characterised by the geographical decentralisation of the developers, the importance of the contributions to the projects and their reputations (Petrinja & Succi, 2012; The Free Software Foundation, 2023). Similar to the aspects of agile software projects, these aspects of FLOSS projects are not sufficiently incorporated in traditional software development models. Hence, several maturity models for FLOSS development are proposed.

QualiPSo Open Source Maturity Model

One of the maturity models for FLOSS development is the QualiPSo Open Source Maturity Model (OMM), which assists in assessing and improving the quality of the FLOSS development process (Petrinja & Succi, 2012). OMM consists of 25 components which represent aspects that are perceived as important for the quality of the FLOSS development process (e.g. project documentation and license management). In OMM, these components are referred to as TrustWorthy Elements (TWEs). Based on their perceived importance, the TWEs are divided into three maturity levels: 1) Basic 2) Intermediate 3) Advanced.

The TWEs are decomposed into smaller components based on the Goal-Question-Metric (GQM) approach. Each TWE comprises one or more goals related to that particular TWE. The defined goals in OMM address three types of aspects: the creation of elements (e.g. documentation, software components and processes), the management of these created elements and their improvement. Each goal in turn consists of practices, which represent activities which are commonly undertaken in FLOSS projects. For each practice, OMM defines several metrics that assist in determining to which extent the corresponding practice is fulfilled.

During the assessment of a FLOSS project, the metrics corresponding to the practices are assigned a value ranging from 1 to 4, which represents their degree of fulfilment. Metrics that are not applicable to the project are assigned a value of 0. For each practice, a rating is calculated based on the assigned values to the corresponding metrics. In order to determine whether a maturity level is obtained, the sum of all practice ratings is divided by the sum of the maximum achievable practice ratings for that corresponding level. For a level to be fulfilled, this calculated maturity rating should be at least 90%.

OMM distinguishes two types of assessments: a (internal) complete and (external) partial OMM assessment. In a complete OMM assessment, individuals that have access to all documents of a FLOSS project participate, while in a partial OMM assessment the assessors only have access to publicly available documents.

4.5 Comparison of maturity models

In Table 4.1, the discussed maturity models for software engineering in the previous sections are compared based on several design characteristics. From the comparison, we notice that most of the discussed maturity models have a prescriptive purpose of use. These maturity models do not only assist in assessing the current states (i.e. the as-is situations) of objects under consideration, but also propose measures to guide the improvement of these objects. As our aim is to develop a maturity model that guides ML teams in improving their development processes, our model should also have a prescriptive purpose of use.

All discussed maturity models, except SAMI, are grounded in knowledge and experience from industry. This often originates from the fact that maturity models are developed or adapted using the contributions of practitioners. Besides their grounding in knowledge and experience from industry, some of the models are also based on literature, such as existing models and standards (e.g. CMMI-DEV and ISO/IEC standards).

The maturity model that we will develop in this study focuses on maturity assessments at the team level. In contrast, most discussed models assist in determining the maturity of entire organisations. Assessments with the models are mostly performed by teams of assessors based on artifacts, such as process documentation and dashboards. In most cases, these teams consist of internal and/or external assessors. Internal assessors are individuals which are related to the object under consideration while external assessors are independent professionals in the domain of the maturity model.

Table 4.1: Comparison of maturity models for software engineering

	CMMI-DEV	SEMM	SAMI	SAFe MM	BSIMM	SAMM	OMM
Purpose	Descriptive	Prescriptive	Prescriptive	Prescriptive	Descriptive	Prescriptive	Prescriptive
Grounding	Industry	Literature and industry	Literature	Literature and industry	Industry (data)	Industry	Literature and industry
Analysis level	Organisation	Organisation	Organisation and project	Organisation	Organisation	Organisation	Project
Maturity levels	5	4	5	5	3	3	3
Assessment type	Third-party assessment	Third-party assisted assessment (Certification audit)	Internal or third-party assessment	Third-party assisted assessment	Third-party assessment	Internal assessment (self-assessment)	Internal or third-party assessment
Assessment base	Artifacts and affirmations[1]	Artifacts and affirmations[1]	Indicators	Indicators and workshops	Artifacts and interviews	Interviews or workshops	Artifacts
Assessors	Team of internal and external assessors	Team of internal and external assessors	Agile coach or internal employee(s)	Internal assessors accompanied by model developers	Team of two or three BSIMM assessors	One or more internal assessors	Team of assessors
Application	Thousands of organisations worldwide	Several organisations from the Spanish software industry	One organisation for which the benefits of the application is described in an academic study	One organisation as part of a case study to evaluate the model	130 organisations across different industries	Various organisations worldwide	Several FLOSS projects as part of academic studies

[1] A confirmation that a particular element (e.g. a best practice) of which the implementation is required by a maturity model, is actually implemented. This confirmation could be collected through different methods (e.g. interviews and questionnaires).

Chapter 5

Design approaches

Although numerous maturity models for different domains are presented in literature, methods and procedures for the development of such maturity models are less extensively covered. In the following sections, we discuss several design approaches for maturity models. We distinguishes two types of design approaches: approaches for general (i.e. fixed-level maturity models) and specific models. After discussing the design approaches, we identify the common phases in the development of maturity models.

5.1 Approaches for general models

One of the initial studies on the development of maturity models is the work of de Bruin et al. (2005). In this study, de Bruin et al. (2005) presented a development framework for maturity models in which the general phases of the development of these models are specified. In order to provide support for the presented framework, the framework was applied for the development of two maturity models in the domains of business process management and knowledge management.

The phases, which are included in the development framework, are as follows: scope, design, populate, test, deploy and maintain. In the scope phase, the focus of the maturity model, i.e., the domain that the model will target, is determined. Moreover, stakeholders are identified that will be involved in the development of the model. These stakeholders could be from academia, industry and the government. In the design phase, the design (i.e. structure) of the model, which should reflect the needs of the intended audience of the model, is determined. Maturity models mostly represent the maturity of an entity through a number of stages of which their requirements build upon each other. In addition, the design of the model involves several other decisions, such as the assessment method (e.g. self-assessment or third-party assisted) and the intended assessors.

In the populate phase, the content of the model is defined. The domain of the model is decomposed into domain components and sub-components. In order to decompose the domain into components, a combination of several research methods (e.g. literature review, interviews and the Delphi method) is usually employed. Besides the definition of the content, in the populate phase the assessment instrument, i.e., the instrument with which the assessment will be conducted, is created. After the populate phase, the model and the created assessment instrument are tested for validity and reliability. Subsequently, the model is deployed (i.e. made available for use) in real-world settings, and its generalisability is verified and improved. In order to gradually standardise the model and acquire acceptance, the deployment of the model is executed in two steps. The model is first deployed to stakeholders that were involved in its development and testing after which it is deployed to entities that were not part of these phases. The last phase of the framework focuses on the evolution and refinement of the model. As the domain and the understanding of the model evolve over time, the model is required to change in order to stay relevant. According to the framework, changes to the model should be tracked in the form of a repository.

Based on the work of de Bruin et al. (2005), Mettler (2011) proposed a phased approach for the development of maturity models. The phased approach consists of the following phases: scope definition, design of the model, design evaluation and evolution reflection. For each phase, Mettler (2011) identified design decision parameters, i.e., decisions that have to be taken during the development of maturity models. In the scope definition, the scope of the maturity model is defined. The scope definition involves the following design decisions: the extent of the model (i.e. focus on general or more-specific subject), the novelty of the targeted subject (e.g. focus on a mature or completely new subject), level of the analysis (e.g. group or organisation level), the intended audience and the distribution of the model (i.e. open or exclusive access). After the scope is defined, the maturity model is developed. In order to develop the model, the targeted notion of maturity should be defined (e.g. process-focused or object-focused definition), which influences the content of the model. For example, in case of a process-focused definition of maturity the content of the model should focus on process activities and practices while an object-focus definition requires the definition of functional and non-functional requirements. Besides defining the notion of maturity, the dimensionality of the model (i.e. one or multi-dimensional), the driver of the model design (i.e. theory-driven, practice-driven or a combination), the form of the model (e.g. software application), the assessment method and the intended users should be determined.

In the design evaluation phase, the model is verified and validated. Verification is the process of determining to which extent the model satisfies the intent of the designer accurately, while validation focuses on determining whether the model represents the real-world from the perspective of the intended users. The development process of the model could also be evaluated in this phase. Besides deciding on the subject of the evaluation, the timing of the evaluation should be determined, which influences the required research method. An evaluation could be performed before (ex-ante) or after the artifact (ex-post) is implemented in a real-world setting. In the last phase, an approach for the evolution of the model is conceived in order to incorporate the continuous change of the targeted subject and maintain the standardisation and acceptance of the model. In order to conceive an evolution approach, the subject (i.e. model content and assessment approach) and the frequency (i.e. non-recurring or continuous) of the evolution should be determined. Moreover, it should be determined who provides the incentive for model changes (i.e. internal or external). For example, model changes can be induced by the users of the model or the model designer.

Another study that addresses the development of maturity models was conducted by Becker et al. (2009). In this study, Becker et al. (2009) defined requirements for the development of these models based on guidelines for design science research proposed by Hevner et al. (2004). The researchers compared several maturity models in different domains on their fulfilment of these requirements. Based on this comparison, Becker et al. (2009) proposed a procedure model for the development of maturity models, which supports model designers in achieving the defined requirements. In order to demonstrate the applicability of the procedure model, the model was applied for the development of a maturity model in the domain of IT performance measurement.

The procedure model begins with the problem definition, in which the problem to be solved is defined and its relevance is identified. In order to define the problem, the domain and the intended audience of the model need to be determined. After the problem definition, existing maturity models are compared in order to determine the design strategy. Within the procedure model, the following design strategies are distinguished: the development of a completely new design, the improvement of an existing model, the combination of models into a new model and the application of structures or the content of existing models to a different domain.

The central phase of the procedure model is the iterative development of the maturity model. The development of the maturity model consists of the following steps: determination of the high-level structure of the model (i.e. one-dimensional or multi-dimensional), selection of a method for the design of each abstraction level (i.e. model dimensions or their attributes), design of the model according to the selected methods and testing of the model on its adequacy, consistency and comprehensiveness. Based on the results of the test, another iteration over development steps can be performed.

After the development of the model, the forms in which the model is transferred (i.e. transfer forms) to the intended users are conceived. Maturity models could, for example, be transferred to the targeted audience in the form of check-lists, reports, manuals and software applications. The design of the model transfer should incorporate possibilities for the evaluation of the model in order to allow users to provide feedback on the model (e.g. forms for change requests). In the next phase, the model is made accessible in the conceived forms. Finally, the model is evaluated in order to determine whether the model solves the defined problem and provides the aimed benefits. Based on the results of this evaluation, the development process may be reiterated, the transfer design may be modified while the maturity model remains unchanged or the maturity model may be rejected completely. If an unchanged maturity model is required to address the defined problem permanently, the model needs to be evaluated regularly in order to determine whether model modifications are necessary due to changing conditions, technical advancements and novel scientific insights.

In contrast to the previously discussed studies, Pöppelbuß and Röglinger (2011) focused on the maturity model development from the perspective of the models instead of their development process. They proposed a framework, which consists of design principles, i.e., principles of form and function, that maturity models should comply with in order to be employed in a useful manner. In order to define the design principles, Pöppelbuß and Röglinger (2011) reviewed literature related to maturity models. The framework serves as a kind of checklist, which assists researchers and practitioners in the development of novel maturity models and enables them to compare existing models. The defined design principles are categorised based on application purposes of maturity models. Within the framework, the following design principles are distinguished: basic principles, principles for a descriptive application purpose and principles for a prescriptive application purpose. Maturity models should comply with the basic principles regardless of their purpose. A descriptive model should comply with the basic principles and the principles for a prescriptive application purpose while a prescriptive model should satisfy the design principles of all categories.

In order to demonstrate the usefulness of the framework, three maturity models in the domain of business process management were evaluated on their fulfilment of the design principles. Pöppelbuß and Röglinger (2011) found that the evaluated models comply well with the basic design principles and the principles for a descriptive application purpose. The principles for a prescriptive application purpose, however, are not sufficiently fulfilled. Hence, Pöppelbuß and Röglinger (2011) concluded that especially these principles are helpful for the development of maturity models and contribute to the practicability of models when satisfied.

5.2 Approaches for specific models

Besides research on the development of general maturity models, a few studies address the development of specific types of models. Maier et al. (2012) proposed a roadmap for the development of maturity grids. They describe maturity grids as process maturity frameworks that specify the characteristics that any process should have to deliver a high performance and the capabilities that every organisation should possess in order to develop and deploy these high-performance processes. In maturity grids, process performance aspects are structured against a set of maturity levels. For each process performance aspect, a textual description of the characteristics at each maturity level is provided.

The roadmap was developed based on a review of 24 maturity grids, the feedback of two experts and the own expertise of Maier et al. (2012). In order to demonstrate the utility of the roadmap, the roadmap was used for the development of a maturity grid, which focuses on communication management within design engineering organisations. The roadmap comprises the following four phases: planning, development, evaluation and maintenance. Similar to Mettler (2011), Maier et al. (2012) defined a set of decision parameters for each phase. In the planning phase, the intended audience (i.e. all stakeholders involved in the assessment), the assessment purpose (i.e. raise awareness or benchmarking against best practice entity), the scope of the grid (i.e. general focus or domain-specific focus), and success criteria for model development and application are specified. After the planning phase, the grid is developed. In order to develop the grid, process areas and maturity levels should be defined. In addition, the textual descriptions of process area characteristics at each maturity level should be formulated. Moreover, the distribution method of the grid should be determined.

In the evaluation phase, the grid is tested in practice by organisations. These evaluations are performed in order to obtain feedback from participating organisations and iteratively refine the grid. These evaluations should be performed until no more significant improvement suggestions are obtained and the designer of the grid is satisfied with the evaluation results. In an evaluation, the grid is validated. The validation of the grid focuses on determining whether the intent of the designer corresponds with the understanding of its users. Furthermore, the obtained assessment results from applying the grid are tested for correctness. Besides validation, the grid is verified against the defined success criteria in the planning phase. Finally, as the domain of the grid broadens and deepens, the grid is continuously maintained in order ensure its accuracy and relevance. In case of major adjustments to the grid, the evaluation phase should be repeated. Besides the maintenance of the grid, the development process of the grid and obtained results (e.g. from assessments) should be adequately documented and properly communicated.

Another study that focuses on a specific type of maturity models was conduced by van Steenbergen et al. (2010). In this study, van Steenbergen et al. (2010) presented a method for the development of focus area maturity models. A focus area maturity model consists of focus areas related to a particular functional domain (e.g. enterprise architecture or software product management). Each focus area comprises a set of capabilities of progressive maturity. All capabilities of the focus area are structured relative to each other in a matrix, which defines an incremental maturity path. In the matrix, the focus areas are placed against the maturity levels. The capabilities, which are denoted by letters, are positioned at a particular maturity level in this matrix. The positions of the capabilities in the matrix indicate the order in which these capabilities should be acquired. The matrix describes which capabilities should be acquired after acquiring capabilities in the same focus area and other focus areas (i.e. intra and inter-process dependencies).

The presented method was developed based on a literature review on existing development methods and practical experience. Based on this literature review, van Steenbergen et al. (2010) derived generic phases in the development of maturity models. For each generic phase, they defined a set of development steps that are specific for focus area maturity models. The method comprises ten steps divided over four generic phases: scoping of functional domain, identifying focus areas, defining capabilities, identifying dependencies between capabilities, structuring capabilities in matrix, developing assessment instrument, defining improvement actions, implementing the model, iteratively refining the model and communicating design results. The steps are schematically represented in an activity diagram.

5.3 Common development phases

Similar to van Steenbergen et al. (2010), we identify common phases in the development of maturity models based on a comparison of the discussed process frameworks, i.e., frameworks that describe the process of developing maturity models. The comparison comprises development process frameworks for general maturity models as we focus on this type of maturity models in this study. The method of van Steenbergen et al. (2010) for the development of focus area maturity models is thus omitted from this comparison. We, however, incorporate the roadmap for the development of maturity grids proposed by Maier et al. (2012) due to the similarities between these grids and general maturity models.

The comparison of the process frameworks is represented in Table 5.1. We identify the following common development phases in the development process frameworks:

- **Scoping:** The definition of the scope of the maturity model. The scope definition encompasses specifications of the targeted domain and the intended audience.
- **Construction:** The construction of the maturity model, which often comprises the selection of design methods, the determination of the model dimensionality, the definition of model components, the design of the assessment instrument and the specification of the model users.
- **Validation:** The testing of the maturity model in practice on several criteria (e.g. completeness, usability and accuracy) before its implementation in real-world settings.
- **Deployment:** The publication of the maturity model to the targeted users in real-world settings.
- **Revision:** The change and refinement of the maturity model due to evolution of the model domain in order to ensure its accuracy and relevance. Modifications could be the result of evaluations, i.e., the study of maturity models in real-world settings.

As stated before, the design evaluation phase of the process framework of Mettler (2011) could be performed ex-ante or ex-post. In Table 5.1, the design evaluation phase is placed under the validation phase as we assume that the evaluation is performed ex-ante. In case of an ex-post evaluation, the evaluation is, however, performed after the implementation of the model in real-world settings and should be considered part of the revision phase. Moreover, it is important to note that model validation is part of the iterative development phase of the process framework of Becker et al. (2009).

The development process frameworks and the identified common phases provide a direction for the development of the maturity model for ML-based software systems. Due to time constraints, we do not implement our model in real-world settings. In this study, we, therefore, focus only on the scoping, construction and validation phases.

Table 5.1: Comparison of development process frameworks for maturity models

Generic phases	de Bruin et al. (2005)	Becker et al. (2009)	Mettler (2011)[2]	Maier et al. (2012)
Scoping	**Scoping:** The focus of the model and the development stakeholders are determined.	**Problem definition:** The problem to be solved is defined and its relevance is identified.	**Scope definition:** The scope of the model is defined, which comprises decisions on the extent of the model, the level of the analysis, the subject novelty, the intended audience and distribution of the model.	**Planning:** The specification of the intended audience, the assessment purpose, the scope of the grid and success criteria.
Construction	**Designing:** The design of the model (i.e. structure) is determined, which involves several design choices (e.g the assessment method and intended assessors). **Populating:** The content of the model and the assessment instrument are determined.	**Comparison of existing models:** Existing maturity models in the targeted domain are compared. **Determination of the design strategy:** The design strategy is determined based on the comparison of existing models in the previous phase. **Iterative development of the model:** The model is iteratively developed, which encompasses the following steps: determining model structure, selecting approaches, designing and testing of the model[1].	**Design of the model:** The model is developed, which encompasses the definition of the notion of maturity, the dimensionality of the model, design driver, form of the model, the assessment method and the intended users.	**Development:** The development of the grid, which involves the definition of process areas and maturity levels, the formulation of process area descriptions and the determination of the distribution method.
Validation	**Testing:** The model and the assessment instrument are tested for validity and reliability.		**Design evaluation:** the model is verified (i.e. determining to which extent designer's intent is satisfied) and validated (i.e. determine whether the model represents the real-world)	**Evaluation:** The validation (i.e. correspondence between developer's intent and understanding of users, and correctness of results) and verification of the grid against success criteria.
Deployment	**Deployment:** The model is made available for use, and its generalisability is verified and gradually improved.	**Conception of the transfer design:** The transfer of the model to the intended users is conceived. **Implementation of transfer design:** The model is made accessible according to the conceived transfer design.		
Revision	**Maintenance:** The model is changed and refined as a result of the evolution of the model domain and its understanding over time.	**Evaluation:** The model is evaluated in order to determine whether the model provides a solution for the defined problem and the aimed benefits. Evaluations could results in transfer and model modifications.	**Evolution reflection:** A model evolution approach is conceived in order to incorporate subject changes and ensure model standardisation and acceptance.	**Maintenance:** The grid is continuously maintained to keep the grid accurate and relevant. Moreover, the development process and obtained results are documented and communicated.

[1] The testing of the model could be considered equivalent to the validation of the model, which is part of the iterative development phase.
[2] The design evaluation phase is assumed to be conducted ex-ante. In case of an ex-post evaluation, the design evaluation phase is part of the revision phase.

Chapter 6

Model development

The identified scoping and model construction phases in Section 5.3 guide the development of our maturity model. In these phases, we follow the design decision parameters of de Bruin et al. (2005) and Mettler (2011). Furthermore, we adhere to some of the guidelines proposed by Becker et al. (2009). As Maier et al. (2012) address the development of maturity grids, we do not focus on their work during the development of our model.

In the following sections, we elaborate on the scoping and construction of our model.

6.1 The scoping of the model

As stated before, our maturity model focuses on the development process of ML-based software systems on a team level. Over the past years, the development of these systems has been increasingly covered in literature (e.g. Giray, 2021; Wan et al., 2021; Washizaki et al., 2019). As ML teams currently have little insights into their adoption of best practices and guidance for the improvement of their practice adoption is limited, our model aims to support and guide ML teams in assessing and improving their development processes based on a set of 45 best practices identified by Serban et al. (2020, 2021). Therefore, the model allows teams to obtain insights into the current situation (i.e. the as-is situation) of their development processes in terms of their best practice adoption and to adapt these processes based on this practice adoption.

We follow the categorisation of Serban et al. (2020, 2021) of the best practices into groups, which we refer to as practice domains. Our maturity model thus covers the following six domains of the development process of ML-based software systems: data, training, coding, deployment, team and governance. The practice domains with their corresponding descriptions are shown in Table 6.1.

Table 6.1: The practice domains with corresponding descriptions. This categorisation of practices is derived from Serban et al. (2020, 2021).

Practice domains	Description
Data	Practices related to the collection, preparation and cleaning of data, which is used for training the ML models.
Training	Practices associated with the training experiments, which involve the development, evaluation and monitoring of ML models.
Coding	Practices for writing, testing and integrating code. These practices are derived from traditional software engineering.
Deployment	Practices associated with the deployment of the ML models, which include the deploying of the ML models, and their monitoring and maintenance in production.
Team	Practices for collaboration and communication in the ML teams.
Governance	Practices that enforce responsible employment of ML, which include, among other things, privacy, transparency and fairness.

The model will be constructed based on adoption data of the best practices and existing maturity models for software engineering in literature. After the construction of the model, the model will be validated on several characteristics through a case study involving multiple ML teams. Based on the case study, we will determine the quality of the model in terms of the defined characteristics and propose adaptations for the model.

In an attempt to contribute scientifically to the field of ML and enable ML practitioners to develop, deploy and maintain ML-based software systems more robustly and responsibly, our model is publicly available[1] and licensed under the Creative Commons Attribution-ShareAlike 4.0 International License [2].

All the design decisions related to the scoping of the model are summarized in Table 6.2.

Table 6.2: The design decision parameters of de Bruin et al. (2005) and Mettler (2011) for the scoping phase accompanied by our decisions. These decisions define the scope of our model.

Decision parameters	Decision
Model subject	The development process of ML-based software systems
Analysis level	Team level
Subject novelty	Emerging
Stakeholders	Academia and practitioners
Intended audience	ML teams
Accessibility	Publicly available (i.e open)

[1]https://github.com/SE-ML/Maturity-model

[2]https://creativecommons.org/licenses/by-sa/4.0/

6.2 The construction of the model

6.2.1 The dataset

As stated in Section 6.1, our maturity model is developed based on the adoption of best practices by ML practitioners (especially teams). In order to determine the adoption of the practices, we use a dataset that comprises the responses from teams to the questionnaires conducted during the studies of Serban et al. (2020, 2021). The questionnaire that corresponds to the initial study of Serban et al. (2020) is referred to as the initial questionnaire, while the questionnaire that is related to the follow-up study of Serban et al. (2021) is referred to as the extended questionnaire.

Both questionnaires consist of three types of questions: the preliminary questions, practice adoption questions and perceived effect questions. The preliminary questions are mostly questions on the (demographic) factors of the teams, such as team size and team experience. The practice adoption questions are questions regarding the adoption of the practices by the teams. In order to measure the adoption of practices, Serban et al. (2020, 2021) used a Likert scale with four answers in their questionnaires that represent possible adoption degrees of the practices: not at all, partially, mostly and completely. For some practice adoption questions, Serban et al. (2020, 2021) added other possible answers to this Likert scale, such as not applicable. The perceived effects questions, the last type of question, focus on the perceived effects of the practices when adopted.

The initial questionnaire consists of 45 questions. The extended questionnaire is an extension of the initial questionnaire with 14 practice questions and one perceived effect question related to the identified practices for trustworthy ML in the follow-up study. The dataset contains in total 504 responses of which 378 responses correspond to the initial questionnaire and 126 responses correspond to the extended questionnaire.

6.2.2 The creation of a data pool

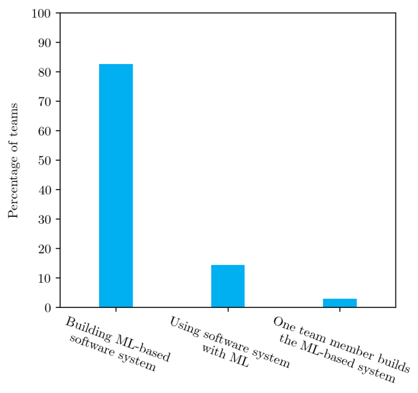
In order to be able to develop the maturity model based on adoption data of the practices, we create a consistent data pool by processing the dataset. In this regard, we follow BSIMM, which was discussed in Section 4.3. BSIMM is constructed based on a data pool of firms (i.e. the BSIMM community). In contrast to BSIMM, the data pool that we create consists of a number of teams instead of firms.

For the creation of the data pool, all questions that do not occur in both the initial and extended questionnaire are removed to ensure that all participants in our data pool have answered the same questions. Furthermore, the responses of participants that are not part of a team that builds ML-based software systems or use ML are discarded from the dataset. Moreover, we filter out all responses of teams that did not answer at least one practice adoption question with one of the four adoption degrees in the defined Likert scale. Therefore, our data pool contains exclusively teams that answered the selected practice adoption questions with adoption degrees.

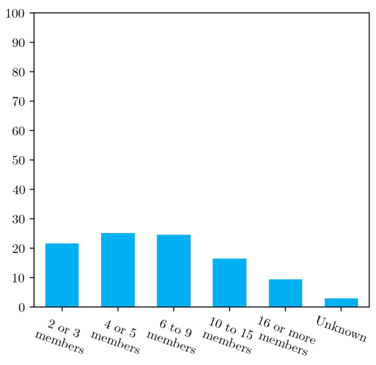
The resulting data pool, which is used for the development of our model, consists of 139 teams. Figure 6.1 illustrates the data pool based on several (demographic) factors. In the data pool, three types of teams are distinguished: teams that build ML-based software systems, teams in which one team member builds the systems and teams that use ML. From Figure 6.1a, we observe that most of the teams in our data pool (i.e. 82.7%) build ML-based software systems while in a small proportion of the teams (i.e. 2.9%) only one member builds the system. In Figure 6.1b, the distribution team sizes is presented. The largest percentage of teams of 25.2% consists of 4 to 5 members. This is almost equal to the percentage of teams (24.5%) with a size of 6 to 9 members.

The teams in our data pool are from all over the world as shown in Figure 6.1c. The largest proportions of teams in the data pool correspond to Europe, North America and Asia. For a small proportion of the teams (i.e. 0.7%), the geographical location is unknown. In Figure 6.1d, we present the proportion of teams per organisation type. Most teams work at a tech company, a university or in a non-commercial research lab. A small proportion of the teams of 10.8% work at a governmental organisation. As can be seen from Figure 6.1e, most of the teams in our data pool have between 1 to 5 years of experience. With a proportion of 35.3%, an experience between 2 to 5 years occurs the most under the teams followed by an experience between 1 to 2 years with a proportion of 30.2%.

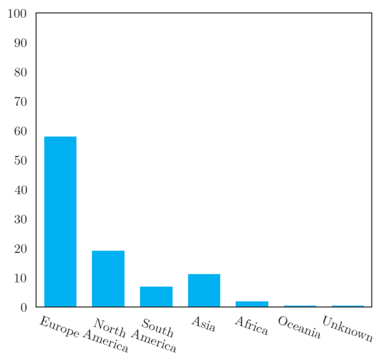
Figure 6.1f shows the used data sources by the teams in proportions. From this figure, we observe that the three most used data sources are tabular data, text and images. A proportion of 14.4% of the teams use other data sources than the specified sources, such as logs and sensor data.



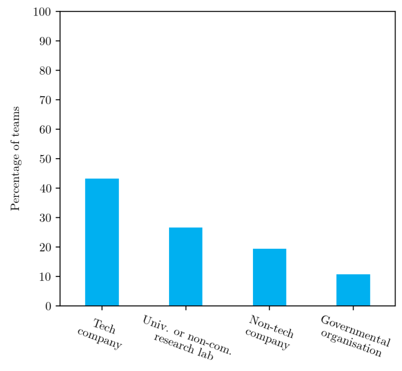
(a) Application of ML within the teams



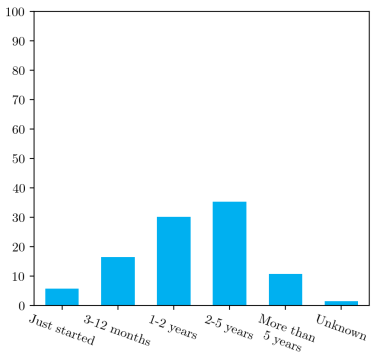
(b) Size of the teams



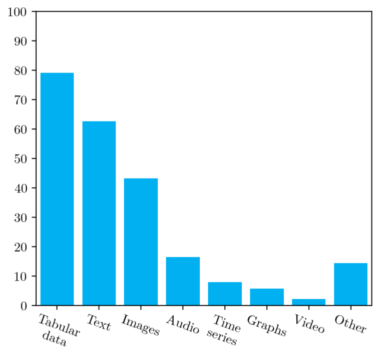
(c) Geographical locations of the teams



(d) The types of organisations



(e) Experience of the teams



(f) Used data sources by teams

Figure 6.1: Information about the teams of the created data pool used for the development of the maturity model. The data pool is visualised based on several (demographic) factors.

6.2.3 The adoption of the practices

Based on the created data pool, we determine the adoption of best practices by the teams in this pool. Within our maturity model, we define two scores that represent the adoption of a practice: the practice adoption and the adoption percentile. The practice adoption corresponds to the number of teams that have adopted a particular practice. We consider a practice adopted by a team if it has been assigned an adoption degree of at least 'mostly'. The practice adoption score could be considered similar to the observation frequency of a security activity in the data pool of BSIMM, which indicates the number of firms that perform that activity. The adoption percentile of a practice is the percentile rank of the corresponding practice adoption score compared to the practice adoption scores of all other practices. The percentile rank of the practice adoption score represents how often the corresponding practice is adopted compared to the other practices. This allows for the comparison between different practice adoption scores. For example, a practice with an adoption percentile of 90% indicates that 90% of the other practices have an equal or lower practice adoption score than this practice.

Table 6.3 presents all practices with their corresponding adoption scores ordered from high to low. As the practice adoption questions that do not occur in both the initial and extended questionnaire are discarded from the data, the adoption of some practices is not determined.

Table 6.3: The practices with their adoption scores arranged in descending order. Two adoption scores are distinguished: the practice adoption and the adoption percentile. The practice adoption score of a practice represents the number of teams in our data pool that have adopted that practice while the adoption percentile represents the percentile rank of its practice adoption score. Some practices do not have adoption scores due the manner in which the dataset is processed.

Nr.	Practice	Practice domain	Practice adoption	Adoption percentile
1	Capture the training objective in a metric that is easy to measure and understand.	Training	102	100.0
2	Share a clearly defined training objective within the team.	Training	97	96.4
3	Enable parallel training experiments.	Training	96	91.1
4	Continuously measure model quality and performance.	Training	96	91.1
5	Use versioning for data, model, configurations and training scripts.	Training	95	83.9
6	Use a collaborative development platform.	Team	95	83.9
7	Share status and outcomes of experiments within the team.	Training	90	78.6
8	Write reusable scripts for data cleaning and merging.	Data	85	75.0
9	Communicate, align, and collaborate with others.	Team	80	67.9
10	Work against a shared backlog.	Team	80	67.9
11	Continuously monitor the behaviour of deployed models.	Deployment	80	67.9
12	Ensure data labelling is performed in a strictly controlled process.	Data	79	60.7
13	Make data sets available on shared infrastructure (private or public).	Data	75	57.1
14	Enable automatic roll backs for production models.	Deployment	72	53.6
15	Automate model deployment.	Deployment	68	50.0
16	Peer review training scripts.	Training	66	46.4
17	Perform checks to detect skew between models.	Deployment	62	42.9
18	Use static analysis to check code quality.	Code	60	39.3
19	Log production predictions with the model's version and input data.	Deployment	56	35.7
20	Enable shadow deployment.	Deployment	54	30.4
21	Enforce fairness and privacy.	Governance	54	30.4
22	Test all feature extraction code.	Training	53	25.0
23	Automate hyper-parameter optimisation.	Training	50	21.4
24	Check that input data is complete, balanced and well distributed.	Data	48	17.9
25	Use sanity checks for all external data sources.	Data	47	14.3
26	Run automated regression tests.	Coding	42	10.7
27	Actively remove or archive features that are not used.	Training	37	7.1
28	Assign an owner to each feature and document its rationale.	Training	28	3.6
29	Test for social bias in training data.	Data	-	-
30	Prevent discriminatory data attributes used as model features.	Data	-	-
31	Use privacy-preserving machine learning techniques.	Data	-	-
32	Employ interpretable models when possible.	Training	-	-

Table 6.3: The practices with their adoption scores arranged in descending order. Two adoption scores are distinguished: the practice adoption and the adoption percentile. The practice adoption score of a practice represents the number of teams in our data pool that have adopted that practice while the adoption percentile represents the percentile rank of its practice adoption score. Some practices do not have adoption scores due the manner in which the dataset is processed. (Continued)

Nr.	Practice	Practice domain	Practice adoption	Adoption percentile
33	Automate feature generation and selection.	Training	-	-
34	Automate configuration of algorithms or model structure.	Training	-	-
35	Assess and manage subgroup bias.	Training	-	-
36	Use continuous integration.	Coding	-	-
37	Assure application security.	Coding	-	-
38	Provide audit trails.	Deployment	-	-
39	Decide trade-offs through defined team process.	Team	-	-
40	Establish responsible AI values.	Governance	-	-
41	Perform risk assessments.	Governance	-	-
42	Inform users on machine learning usage.	Governance	-	-
43	Explain results and decisions to users.	Governance	-	-
44	Provide safe channels to raise concerns.	Governance	-	-
45	Have your application audited.	Governance	-	-

6.2.4 The structure of the model

As stated in Section 2.2, maturity models represent the maturity of objects as a sequence of stages (i.e. maturity levels). According to de Bruin et al. (2005), stages of maturity models should have short names that indicate their purpose, and descriptions that expand their names and provide information on their content (e.g. their requirements and measures). De Bruin et al. (2005) distinguish two approaches for defining maturity stages: a top-down approach and a bottom-up approach. Both approaches address *what* represents maturity and *how* maturity is measured. In the top-down approach, the descriptions of the stages (i.e. *what*) are created firstly after which the content of the stages (i.e. *how*) is determined. In the bottom-up approach, the content (i.e. *how*) is determined first after which descriptions (i.e. *what*) are created to reflect the content.

De Bruin et al. (2005) also state that in the development of maturity models it is important to consider how maturity stages are presented to the audience of the model. In the linear stage approach, maturity is represented as a sequence of one-dimensional linear stages. An assessment with a linear stage model (i.e. a model that represents maturity as one-dimensional linear stages) results in an overall maturity stage for the entire object under consideration. In the stage-gate approach, the domain (i.e. the subject) of the model is divided into components and subcomponents in order to allow for separate assessments of discrete aspects of the object in addition to its overall assessment.

As the practices identified by Serban et al. (2020, 2021) serve as the starting point for the development of our maturity model, we use the bottom-up approach in order to define the maturity stages for the development process of ML-based software systems. After all, the identified practices form the content of the model (i.e. *how*). Based on these practices, the descriptions of the stages (i.e. *what*) are created.

In order to enable ML teams to assess their development processes in a comprehensive manner and to improve specific aspects of these processes, we use the stage-gate approach for the development of our model. Therefore, our maturity model distinguishes two representations (i.e. views), the domain and maturity representation, which are slightly similar to the improvement paths of CMMI-DEV. The domain representation focuses on the maturity of teams regarding the practice domains described in Table 6.1 represented by engineering capability (EC) levels. These practice domains, which consist of several practices, form the components of the domain of our model. As the number of practices is limited, we do not divide the practice domains further into subcomponents. In contrast to the domain representation, the maturity representation focuses on the overall maturity of teams represented by maturity levels. The maturity levels consist of practices of EC levels of particular practice domains. Both representations contain thus the same set of practices, but the practices in each representation are organised in a different manner. Figure 6.2 illustrates the structure of our model.

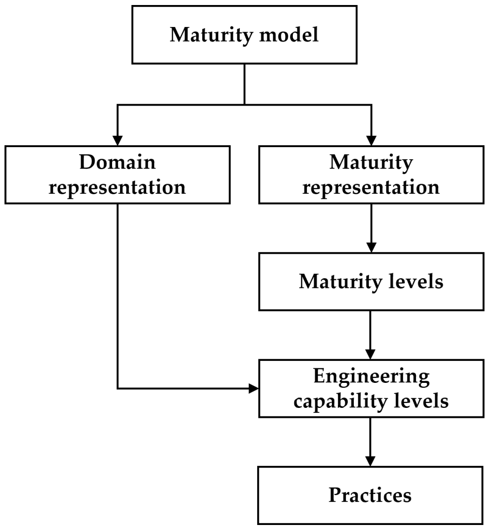


Figure 6.2: The structure of our maturity model. The model comprises two representations (i.e. views): the domain representation and maturity representation. The domain representation uses EC levels to describe the maturity stages of teams regarding the practice domains while the maturity representation uses maturity levels to indicate the maturity stages of the overall development processes of teams. Each maturity level is built of practices of EC levels of several practice domains.

6.2.5 The creation of the domain representation

The ordering approach

After determining the adoption of the practices (Section 6.2.3), the practices are organised into the domain representation. For each practice domain, the corresponding practices are divided across the different EC levels. Our maturity model distinguishes three EC levels with practices. In addition, the model defines an initial level, i.e., EC level 0. In contrast to the other three levels, this level does not consist of practices and is considered the starting point of the maturity of teams regarding the practice domains. In order to divide the practices of each domain across these levels, we devised an approach that consists of the following steps:

1) **Divide practices based on their adoption.** Practices with an adoption percentile greater than 67 are assigned to level 1, practices with an adoption percentile greater than 33 and less than or equal to 67 are placed in level 2, and practices with an adoption percentile equal to or less than 33 are placed in level 3. Practices that are more frequently adopted by teams in our data pool are thus placed in lower levels assuming that these practices are easier to adopt. This manner of dividing practices across levels is similar to the way in which security activities of practice areas are divided across levels based on their observation frequency in BSIMM. In BSIMM, security activities with a higher observation frequency are assigned to lower levels as these activities are more commonly executed than security activities with a lower observation frequency.

2) **Assign remaining practices based on their similarity.** Practices for which the adoption is not determined, are assigned to particular levels based on their similarity in content with other practices in these levels assuming that practices with similar content have the same level of complexity.

3) **Switch the positions of practices.** Switch the positions of practices with other practices in order to have a logical progression of practices in and between levels. For example, a practice could be switched from its position with another practice in case its adoption depends on the achievement of that other practice. Moreover, general practices could be followed by more specific practices assuming that general practices form the basis of specific practices and assist in their implementation.

Ordering the practices into levels

Table 6.4 presents the division of practices across the three different levels after conducting step 1 of our devised approach. In this table and the remainder of this chapter, we refer to the practices with their assigned numbers in Table 6.3.

After step 1, we perform steps 2 and 3 of our approach. In the practice domain "Data", we assign practices 29 (test for social bias), 30 (prevent the use of discriminatory attributes as features) and 31 (use privacy-preserving techniques) to level 3 of the domain as these practices are related to practices 24 (continuously check input data) and 25 (use sanity checks) of that level. After all, practices 24 and 25 focus on checking and processing the data, which is also the basis of practices 29, 30 and 31. Within this level, practice 29 is placed after practice 31 assuming that teams that work with personal data first take measures to preserve privacy (more general) before these teams test for social bias within that data (more specific). Practice 24 follows practices 25, 31 and 29, as the execution of these practices is encompassed by the content of practice 24.

Practices 33 (automate feature generation and selection) and 34 (automate configuration of model structures) corresponding to practice domain "Training" are assigned to level 3 since both of these practices focus on automating particular ML tasks similar to practice 23 (automate hyper-parameter optimisation). Furthermore, practices 32 (employ interpretable models when possible) and 35 (assess and manage subgroup bias) are added to level 3 as these practices are related to practice 28 (assign an owner to each feature and document its rationale). All these three practices encompass aspects of trustworthy ML. Practice 28 contributes to the explainability of the model by documenting the rationale of the feature. Similarly, practice 32 and 35 address explainability and fairness through the employment of interpretable models and the prevention of subgroup bias, respectively.

After practices 32, 33 and 35 are added to level 3, practice 22 (test all feature extraction code) is transferred from level 3 to level 2 as practice 22 is related to practice 16 (peer review training scripts) in that level. Both practices focus, after all, on reviewing code. Moreover, practices 23 and 34 are moved from level 3 to level 2. By moving these practices to level 2, all practices related to automating ML tasks and reviewing code are assigned to level 2 and all practices related to trustworthy ML are assigned to level 3. Although practice 33 focuses on automating feature generation and selection, this practice is added to level 3 instead of level 2 as practice 22 in level 2 assumes that features are manually engineered. For the same reason, practice 33 follows practice 27 (remove or archive unused features) and 28 in level 3. Practice 33 is thus placed after all practices for which it is assumed that features are manually

Table 6.4: The division of practices across the EC levels for each practice domain after the first step of our devised approach. In this step, the practices are organised in levels based on their adoption percentiles. The practices are referred to with their assigned numbers in Table 6.3. EC level 0 does not contain practices and is, therefore, not shown. Furthermore, practices without adoption scores are discarded.

DATA

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
8	85	75.0
Engineering capability level 2		
12	79	60.7
13	75	57.1
Engineering capability level 3		
24	48	17.9
25	47	14.3

TRAINING

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
1	102	100.0
2	97	96.4
3	96	91.1
4	96	91.1
5	95	83.9
7	90	78.6
Engineering capability level 2		
16	66	46.4
Engineering capability level 3		
22	53	25.0
23	50	21.4
27	37	7.1
28	28	3.6

CODING

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
-	-	-
Engineering capability level 2		
18	60	39.3
Engineering capability level 3		
26	42	10.7

DEPLOYMENT

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
11	80	67.9
Engineering capability level 2		
14	72	53.6
15	68	50.0
17	62	42.9
19	56	35.7
Engineering capability level 3		
20	54	30.4

TEAM

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
6	95	83.9
9	80	67.9
10	80	67.9
Engineering capability level 2		
-	-	-
Engineering capability level 3		
-	-	-

GOVERNANCE

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
-	-	-
Engineering capability level 2		
-	-	-
Engineering capability level 3		
21	54	30.4

engineered.

As a training objective should be shared first within the team before this objective can be captured in a metric, the positions of practice 1 (capture the training objective in a metric) and practice 2 (share a training objective within the team) in level 1 are switched. The adoption of practice 2 requires the accomplishment of practice 1. Furthermore, the positions of practice 27 and 28 in level 3 are switched assuming that the adoption of practice 28 could assist teams in the accomplishment of practice 27. After all, practice 28 could enhance the understanding of features that in turn could help teams to remove or archive unused features more easily (i.e. practice 27). Moreover, the positions of practice 32 and 35 are switched since we assume that practice 32 is more impactful in ensuring trustworthy ML than practice 35. Practice 32 requires changing the entire model for an interpretable model while practice 35 only requires the assessment of a model aspect (subgroup bias).

In the domain "Coding", practice 36 (use continuous integration) is added to level 3 assuming that this practice is the most complete and sophisticated practice regarding the writing, testing and integration of code. Practice 18 (use static analysis to check code quality), practice 26 (run automated regression tests) and practice 37 (assure application security) can all be carried out as part of continuous integration. Therefore, these practices should be adopted before practice 36. Practice 37 is added to level 3 due to its similarity to practice 26 as both practices focus on the identification of bugs (from the perspective of the security and functionality of the application, respectively). Practice 26 and 37 are transferred from level 3 to 2 since their adoption could contribute to the accomplishment of practice 36. After all, practice 26 and 37 could be considered part of practice 36, as explained above. Practice 18 is moved from level 2 to level 1. Practice 18 focuses on enhancing the code quality, which facilitates easier maintenance, testing or extension of the code. This in turn is useful for the adoption of the practices in level 2, which mainly focus on testing the code for bugs.

In the "Deployment" domain, practice 38 (provide audit trails) is assigned to level 3. Practice 38 addresses the social implications of machine learning and artificial intelligence (AI) with regards to respecting consumer rights. This calls for regulations on ML-based software systems and insight into model behaviour through audits. As this is a current open problem, the adoption of practice 38 is considered as complex and therefore this practice is placed in level 3.

Practice 20 (enable shadow deployment) is transferred from level 3 to 2 because it is related to practice 17 (perform checks to detect skew between models) as the deployment of shadow models could assists teams in preventing skew between the training and test data. Moreover, the positions of practices 14 (enable automatic roll backs for production model) and 15 (automate model deployment) are switched in order to have these practices in a chronological order (i.e. automatically rolling back models can be performed after the deployment of the models).

Practice 39 (decide trade-offs through defined team process) corresponding to the practice domain "Team" is added to level 3 of that domain because we assume that this practice is the most complex practice regarding the collaboration and communication within teams. It requires the definition and enforcement of a standardised team process for which the adoption of other practices in this domain could be useful. Furthermore, practice 10 (work against a shared backlog) is transferred from level 1 to level 2 assuming that the achievement of practice 10 depends (partially) on the adoption of practice 9 (communicate, align, and collaborate with others). After all, for teams to be able to share a backlog within the team and with external stakeholders (i.e. practice 10), these teams first need to communicate, collaborate and align internally and externally (i.e. practice 9).

As only the adoption of practice 21 (enforce fairness and privacy) in the practice domain "Governance" is determined, our devised approach cannot completely be used to organise the practices of this domain across the three levels. In the practice domain "Governance", we, therefore, arrange the practices based on their similarities and generality. We consider practices 21 and 40 (establish responsible AI values) as the most broad and general practices in this practice domain. As a result, practices 21 and 40 form the basis of all other practices in this domain and these practices are, therefore, assigned to level 1.

Practices 41 (perform risk assessments) and 45 (have your application audited) are clustered together as both practices focus on assessments of ML-based software systems. Similarly, practices 42 (inform users on machine learning usage), 43 (explain results and decisions to users) and 44 (provide safe channels to raise concerns) all ensure a transparent use of ML within software systems from the perspective of the users and are, therefore, grouped together. As we believe that practices 41 and 45 require more effort to adopt than practices 42, 43 and 44 , these practices are placed in level 3 and practices 42, 43 and 44 in level 2.

Table 6.5 presents the division of the practices across the three EC levels for each practice domain, i.e., the domain representation, after the completion of steps 2 and 3. As stated before, the practices are referred to with their assigned numbers in Table 6.3. In steps 2 and 3, 17 practices without adoption scores are assigned to one of the EC levels and the position of 16 practices are switched in or between levels, respectively. The domain representation with written out practices can be found in Table A.1 of Appendix A.

Table 6.5: The division of all practices across the EC levels for each practice domain based on our approach. In our maturity model, this division is referred to as the domain representation. EC level 0 does not contain practices and is, therefore, not shown. The division only presents the practices with their assigned numbers in Table 6.3. The complete domain representation, which consists of written out practices, can be found in Appendix A.

DATA

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
8	85	75.0
Engineering capability level 2		
12	79	60.7
13	75	57.1
Engineering capability level 3		
25	47	14.3
31	-	-
29	-	-
24	48	17.9
30	-	-

TRAINING

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
2	97	96.4
1	102	100.0
4	96	91.1
3	96	91.1
5	95	83.9
7	90	78.6
Engineering capability level 2		
16	66	46.4
22	53	25.0
23	50	21.4
34	-	-
Engineering capability level 3		
28	28	3.6
27	37	7.1
33	-	-
35	-	-
32	-	-

CODING

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
18	60	39.3
Engineering capability level 2		
26	42	10.7
37	-	-
Engineering capability level 3		
36	-	-

Table 6.5: The division of all practices across the EC levels for each practice domain based on our approach. In our maturity model, this division is referred to as the domain representation. EC level 0 does not contain practices and is, therefore, not shown. The division only presents the practices with their assigned numbers in Table 6.3. The complete domain representation, which consists of written out practices, can be found in Appendix A. (Continued)

DEPLOYMENT

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
11	80	67.9
Engineering capability level 2		
15	68	50.0
14	72	53.6
20	54	30.4
17	62	42.9
19	56	35.7
Engineering capability level 3		
38	-	-

TEAM

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
6	95	83.9
Engineering capability level 2		
9	80	67.9
10	80	67.9
Engineering capability level 3		
39	-	-

GOVERNANCE

Nr.	Practice adoption	Adoption percentile
Engineering capability level 1		
40	-	-
21	54	30.4
Engineering capability level 2		
42	-	-
43	-	-
44	-	-
Engineering capability level 3		
41	-	-
45	-	-

The engineering capability levels

After organising the practices across the EC levels, we create short names (i.e. labels) and descriptions for the four EC levels. Following the top-down approach described by de Bruin et al. (2005) , the short name and description of an EC level cover the content of the practices of all practice domains corresponding to that level. Table 6.6 presents the EC levels within our maturity model with corresponding names and descriptions.

Table 6.6: The EC levels with corresponding names and descriptions.

Engineering capability level	Description
0 - Initial	EC level 0 is defined as initial. At EC level level 0, practices related to ML tasks within the corresponding domain are adopted ad hoc or not adopted at all.
1 - Beginner	EC level 1 practices are defined as beginner practices. Beginner practices are considered to be the most fundamental practices within the ML development process. These practices form the prerequisites for all other practices within the corresponding domain.
2 - Intermediate	EC level 2 practices are defined as intermediate practices. Intermediate practices build upon the beginner practices. These intermediate practices are associated with automated, monitored and reviewed machine learning tasks. Most intermediate practices require intensive collaboration within ML teams, and the involvement of external stakeholders (stakeholders outside the team).
3 - Advanced	EC level 3 practices are defined as advanced practices. Advanced practices are considered to be the most sophisticated practices for the development of ML-based software systems. These advanced practices are related to clearly defined and planned ML tasks. The practices prevent negative impact of the use of ML on social, ethical, technical and financial aspects. Some advanced practices are often overlooked during the ML development process.

6.2.6 The creation of the maturity representation

From EC levels to the maturity representation

As stated in Section 6.2.4, the distinction of two representations in our maturity model slightly resembles the structure of CMMI-DEV that defines two improvement paths: the continuous and staged representations. In CMMI-DEV, the obtained results by using the continuous representation, i.e., the achieved capability levels for particular process areas, can be converted to an associated maturity level that represents the overall maturity of the processes of organisations (CMMI Product Team, 2010). Therefore, CMMI-DEV assigns sets of process areas to maturity levels and specifies for each maturity level which capability levels the assigned process areas should have to be equivalent to a particular maturity level. For example, for an organisation to accomplish maturity level 2, all process areas assigned to that maturity level should achieve capability level 2 or level 3.

As we slightly follow CMMI-DEV with the distinction of two improvement paths, our maturity model also enables ML teams to convert their obtained assessment results with the domain representation, i.e., EC levels for particular practice domains, to maturity levels (or vice-versa). Therefore, the levels of the maturity representation are composed of EC levels of particular practice domains (i.e. sets of EC levels of practice domains), which we refer to by their corresponding practice domain and EC level. For example, "Team EC level 1" refers to the practices of EC level 1 related to the practice domain "Team".

The maturity representation distinguishes six maturity levels of which the first level is an initial level. The EC levels of the practice domains are assigned to maturity levels based on the adoption, similarity and dependency of their corresponding practices. In case EC levels are assigned to maturity levels based on the adoption of their practices, we attempt to compose the levels of practices with a similar adoption in such a way that the higher the level, the lower the adoption of the practices of that level. After the EC levels of practice domains are assigned to a maturity level, the corresponding practices are ordered logically in and between levels.

Assigning EC levels and ordering their practices

As maturity level 1 is an initial level that serves as the starting point of the maturity of ML teams, this maturity level does not contain any practices. The practices corresponding to EC level 1 of practice domains "Training", "Team" and "Data" are assigned to maturity level 2 because the adoption percentiles of these practices are approximately similar ranging from 75.0 to 100.0. The practices within maturity level 2 are ordered such that these practices form a logical progression, which reflect our perspective on subsequent steps that ML teams should perform as part of their development processes.

Maturity level 3 is composed of "Data EC level 2", "Team EC level 2" and "Deployment EC level 1". Similar to maturity level 2, all practices of the assigned EC levels to maturity level 3 have approximately equal adoption percentiles ranging from 57.1 to 67.9. From maturity level 3, the practices assigned to a particular maturity level are ordered based on the ordering of practices of the previous level. In this case, the ordering of practices in maturity level 3 is based on the ordering of the practices in maturity level 2. In order to arrange the practices of a maturity level (from level 3) based on the previous level, the following approach is used:

1) Check for practice p in level n if there exists one or more practices in level $n+1$ that is/are related to that practice.

2) If so, this/these practice(s) is/are placed at the first available position in level $n+1$, otherwise continue to step 3.

3) Steps 1 and 2 are repeated for practice $p+1$ until all practices of level $n+1$ are assigned to a position in that level. If all practices of level n are checked, but some practices of level $n+1$ are not assigned to a position, continue to step 4.

4) Order the remaining practices of level $n+1$ based on aspects such as their assumed generality, complexity and priority relative to all other practices in that level.

In case of maturity levels 2 and 3, practice 4 (continuously measure model quality and performance) of maturity level 2 is, for example, related to practice 11 (continuously monitor the behaviour of deployed models) of maturity level 3 since both practices focus on monitoring the models. As practice 4 is the second practice that is related to a practice in maturity level 3 (in this case practice 11), practice 11 is placed as the second practice of maturity level 3.

As practices of EC level 1 corresponding the practice domains "Code" and "Governance", and practices of EC level 2 related to the practice domain "Deployment" have similar adoption percentiles and all their adoption percentiles are lower than the practices allocated to previous maturity levels, these practices are assigned to maturity level 4. Furthermore, maturity level 4 comprises "Training EC level 2" since some of its corresponding practices are related to the assigned practices of the practice domains "Coding" and "Deployment". For example, practices 23 (automate hyper-parameter optimisation) and 34 (automate configuration of model structures) both focus on automating ML tasks such as practices 14 (enable automatic roll backs for production model) and 15 (automate model deployment) of the practice domain "Deployment". In addition, practices 16 (peer review training scripts) and 22 (test all feature extraction code) focus on reviewing code as practice 18 (use static analysis to check code quality) of the practice domain "Coding".

Practice 11 (continuously monitor the behaviour of deployed models) of maturity level 3 is related to the practices corresponding to EC level 2 of the practice domain "Deployment" (i.e. practices 15, 14, 20, 17 and 19) assigned to maturity level 4. After all, all of these practices focus on the deployment of ML-based software systems. Following our described approach, the practices corresponding to EC level 2 of the practice domain "Deployment" are placed as the first practices of maturity level 4 since practice 11 is the first practice of maturity level 3 that is related to these practices. As practice 11 is the only practice that is related to practices of maturity level 4, the other practices of this maturity level are ordered based on other aspects. Practices 23 (automate hyper-parameter optimisation) and 34 (automate configuration of model structures) follow the practices related to EC level 2 of the practice domain "Deployment" assuming that teams first will automate tasks related to the deployment of ML-based software systems (more common) before automating tasks related to the training of these systems (less common). As we assume that practices on reviewing code have a higher priority than practices related to the responsible employment of ML, practices 18 (use static analysis to check code quality), 16 (peer review training scripts) and 22 (test all feature extraction code) are placed before practices 40 (establish responsible AI values) and 21 (enforce fairness and privacy).

"Data EC level 3" and "Coding EC level 2" are assigned to maturity level 5 based on the adoption percentiles of practices 25 (use sanity checks), 24 (continuously check input data) and 26 (run automated regression tests), which are similar. As the practices of EC level 2 of the practice domain "Governance" are related to some practices of EC level 3 of the practice domain "Data", these practices are also added to maturity level 5. After all, practices 31 (use privacy-preserving techniques), 29 (test for social bias) and 30 (prevent the use of discriminatory attributes as features) and the practices of EC level 2 of practice domain "Governance" all focus on ensuring trustworthy ML. Practices 31, 29 and 30 address the fair use of ML-based software systems, while the practices of EC level 2 of practice domain "Governance" ensure the transparent employment of these systems.

Except for practice 25 (use sanity checks) and 30 (prevent the use of discriminatory attributes as features), the order of the practices in maturity level 5 is mostly based on the order of the practices of maturity level 4. Practice 25 is placed as the first practice of level 5 because we assume that it is the most general practice of level 5. Practices 26 (run automated regression tests) and 37 (assure application security) of "Coding EC level 2" are placed after practice 25. These practices are related to practice 18 from maturity level 4, which is the first practice of maturity level 4 that is related to practices in level 5, as they all belong to the practice domain "Coding". The next two practices of maturity level 4 that are related to practices of maturity level 5 are practices 40 (establish responsible AI values) and 21 (enforce fairness and privacy) of the practice domain "Governance". These practices are related to all remaining practices of level 5 as they all focus on trustworthy ML. Therefore, all remaining practices of level 5 are placed after practice 37. However, practice 30 is placed as the last practice of level 5 since we assume that is the most rigorous practice to ensure the trustworthy employment of ML-based software systems. In contrast to the other practices of level 5 that focus on trustworthy ML, practice 30 is the only practice that directly requires the adaptation of the model for its adoption.

Maturity level 6 contains all remaining EC levels. The order of their corresponding practices is completely based on the order of practices of the previous level.

Table 6.7 shows the EC levels that each practice domain should be at order to be equivalent with a maturity level, which allows ML teams to convert their assessment results obtained by using the domain representation into maturity levels (or vice-versa). In addition, Table 6.8 presents the division of the practices into the maturity levels. This division, which is referred to in our maturity model as the maturity representation, with written out practices can be found in Table B.1 of Appendix B.

Table 6.7: The maturity levels represented in EC levels of practice domains. For each maturity level, it is indicated which EC levels the practice domains should be at in order for ML teams to achieve a particular maturity level.

	Maturity level 1	**Maturity level 2**	**Maturity level 3**	**Maturity level 4**	**Maturity level 5**	**Maturity level 6**
Data		EC level 1	EC level 2	EC level 2	EC level 3	EC level 3
Training		EC level 1	EC level 1	EC level 2	EC level 2	EC level 3
Coding				EC level 1	EC level 1	EC level 3
Deployment			EC level 1	EC level 2	EC level 2	EC level 3
Team		EC level 1	EC level 2	EC level 2	EC level 2	EC level 3
Governance				EC level 1	EC level 2	EC level 3

Table 6.8: The division of all practices across the maturity levels. In our maturity model, this division is referred to as the maturity representation. Each maturity level is composed of practices of capability levels corresponding to particular practice domains. Maturity level 1 is considered the starting point of the maturity of ML teams and is, therefore, not shown. The division only presents the practices with their assigned numbers in Table 6.3. The complete maturity representation, which consists of written out practices, can be found in Appendix B.

Nr.	Practice domain	Practice adoption	Adoption percentile
Maturity level 2			
2	Training	97	96.4
1	Training	102	100.0
6	Team	95	83.9
8	Data	85	75.0
4	Training	85	75.0
7	Training	90	78.6
5	Training	95	83.9
3	Training	96	91.1
Maturity level 3			
12	Data	79	60.7
11	Deployment	80	67.9
9	Team	80	67.9
10	Team	80	67.9
13	Data	75	57.1
Maturity level 4			
15	Deployment	68	50.0
14	Deployment	72	53.6
20	Deployment	54	30.4
17	Deployment	62	42.9
19	Deployment	56	35.7
23	Training	50	21.4
34	Training	-	-
18	Coding	60	39.9
16	Training	66	46.6
22	Training	53	25.0
40	Governance	-	-
21	Governance	54	30.4
Maturity level 5			
25	Data	47	14.7
26	Coding	42	10.7
37	Coding	-	-
31	Data	-	-
29	Data	-	-
24	Data	48	17.9
42	Governance	-	-
43	Governance	-	-
44	Governance	-	-
30	Data	-	-
Maturity level 6			
36	Coding	-	-
28	Training	28	3.6
27	Training	37	7.1
33	Training	-	-
35	Training	-	-
32	Training	-	-
39	Team	-	-
41	Governance	-	-
38	Deployment	-	-
45	Governance	-	-

The maturity levels

Similar to the EC levels, we create names and descriptions for the maturity levels after assigning EC levels of practice domains to the levels and ordering their corresponding practices. Each maturity level indicates a particular maturity state of the development processes of ML teams. The maturity levels with their names and descriptions are shown in Table 6.9.

Table 6.9: The maturity levels with corresponding names and descriptions.

Maturity level	Description
1 - Initial	A maturity level 1 development process is characterised as initial. At maturity level 1, ML-based software systems are developed arbitrarily. Within a maturity level 1 development process, ML tasks are executed ad hoc and chaotically. Maturity level 1 does not comprise practices.
2 - Basic	A maturity level 2 development process of a ML team is defined as a basic development process. A basic development process encompasses practices related to the most fundamental tasks for the development of a ML-based software system. At maturity level 2, the focus of a ML team is mostly on the execution of training experiments and their results.
3 - Progressed	A maturity level 3 development process is characterised as a progressed development process. A progressed development process builds upon a basic development process. While a basic development process focuses on training experiments and their outcomes, a progressed development process covers all essential steps (performed in a moderately basic manner) within the ML workflow. At maturity level 3, the development of ML-based software systems is conducted in a more organised manner, which establishes traceability of work items. Maturity level 3 contains thus mostly practices associated with collaborative and data management tasks.
4 - Optimised	A maturity level 4 development process is defined as an optimised development process. At maturity level 4, the development process is improved in terms of training (experiments), deployment and coding, which is mostly enforced with automation and checks. Besides, ML teams start to pay attention to fairness. An optimised development process comprises automation within training experiments, automated model deployment, and model and coding checks (e.g. peer reviewing code).
5 - Proactively optimised	A maturity level 5 development process is defined as a proactively optimised development process. A proactively optimised development process focuses on preventing negative impact on and of the built software system in terms of technical, social, ethical aspects, which assures a responsibly built and qualitative system. Hence, maturity level 5 encompasses practices related to (data) errors, bugs, cyber attacks and bias prevention.
6 - Perfected	A maturity level 6 development process is defined as a perfected development process. At maturity level 6, all defined domains (e.g. "Data" and "Coding") within the ML workflow are considered to be fully developed. A perfected development process contains practices related to defined and reviewing tasks.

6.2.7 The application of the model

In order to enable ML teams to independently apply our maturity model without the assistance of external stakeholders (e.g. the developers of the maturity model), assessments with our maturity model are conducted on a self-assessment basis. For a ML team to conduct an assessment of their development process, a team should choose which representation(s) to use: the domain representation, the maturity representation or both. During the assessment, a team should determine to which extent they adopt each practice of the selected representation(s). De Bruin et al. (2005) suggest to use Likert scales for assessment questions in order to improve reliability and response consistency. We, therefore, use the Likert scale of Serban et al. (2020, 2021) to reflect the degrees of adoption. As stated in Section 6.2.1, this Likert scale distinguishes four possible adoption degrees: not at all, partially, mostly and completely. Since some practices in our maturity model are not applicable to all development processes of ML teams, we add "not applicable" to the possible answers of the scale for these particular practices. These practices are practices 12, 25, 31, 29, 30, 22, 28, 27, 33, 35 and 21. Teams for which these practices are not applicable, do not have to take them into account during the assessment of their development processes. For example, the adoption of practices 30 (prevent the use of discriminatory attributes as features) and 31 (use privacy-preserving techniques) are irrelevant for ML teams that do not use personal data during their development processes.

In our maturity model, a particular level is accomplished when all practices corresponding to that level and all practices of any lower levels are adopted. A practice is considered to be adopted as a team adopts that practice "mostly" or "completely" within its development process. Practices that are considered "not applicable" by a team are not taken into account when determining the achieved level.

As stated in Section 5.1, according to Becker et al. (2009) part of the development of a maturity model is the conception of the transfer forms of the model, i.e., the forms in which the model is transferred to the users of the model (e.g. reports and software applications). In addition, de Bruin et al. (2005) state that during the development of a model it is necessary to create the instrument with which the assessment should be conducted. Moreover, Mettler (2011) identified the form of the model as one of the important design decision parameters for the development of a model. For our maturity model, we, therefore, created a report (i.e. framework report) and an interactive workbook in Excel, which support teams in applying the model. The report provides teams with all essential information in order to be able to comprehend and use the model. The interactive workbook provides teams with a straightforward way to assess their development processes, obtain their assessment results and plan the improvement of their processes.

In order to assist teams in planning the improvement of their development processes, teams could create a roadmap in the "process improvement" spreadsheet of the interactive workbook, which defines the improvement of their processes in several phases. From the assessment of a development process, a team can determine for which practices the adoption needs to be improved in order to reach their desired level(s). A team should distribute these practices over the different phases of the roadmap to define an incremental plan for their process improvement initiative. For each practice in a phase, a team should also specify the aimed adoption for that practice at the end of a phase.

The use of a roadmap in the interactive workbook of our maturity model that enables teams to plan their improvement initiatives is based on SAMM (OWASP, n.d.-b). As stated in Section 4.3, in SAMM a roadmap could be created for the implementation of activities that are required for the object under consideration (e.g. an organisation) to achieve its desired maturity regarding its software security posture. In the roadmap, these required activities are distributed over a number of phases with a particular duration.

In order to guide teams during the application of our maturity model, we created the application guide. The application guide defines the application of our model as a process of five consecutive steps: 1) plan and prepare the assessment 2) conduct the assessment 3) set maturity target(s) 4) plan process improvement 5) execute process improvement plan. Each step of the application guide is described by several components, which are based on the components of the steps of the quick start guide of SAMM (OWASP, n.d.-b): the purpose of the step, the required (sub)activities, useful resources and best practices for the execution of a step. The application guide can be found in Appendix C.

6.2.8 Consistent and objective assessments

As stated in Section 6.2.7, ML teams can assess their development processes with the use of our maturity model through a self-assessment. A disadvantage of a self-assessment is that obtained assessment results are to a certain extent subjective as assessors (in this case ML teams) could under- or overestimate their own capabilities. In order to reduce the subjectivity of and improve the consistency between assessments that are performed with our maturity model, we studied how other maturity models in the field of SE assure that their assessments are objective and consistent.

For example, assessments with CMMI-DEV, which are referred to as appraisals, should satisfy sets of requirements defined in the Appraisal Requirements for CMMI (ARC) document (SCAMPI Upgrade Team, 2011a). The ARC document defines three types of appraisal methods based on the set of requirements that a particular method satisfies: ARC Class A, B and C appraisal methods. The requirements in the ARC document ensure consistency between the different appraisals and appraisal methods. An example of an ARC Class A appraisal method is the Standard CMMI Appraisal Method for Process Improvement (SCAMPI) A (SCAMPI Upgrade Team, 2011b). SCAMPI A is well-defined and extensively described, which supports the consistency between different appraisals performed with this method. SCAMPI A comprises three phases. Each phase consists of several processes. These processes are described based on several elements, such as purpose, activities and outcome. The activities of the processes can, in turn, be decomposed into three components: required practices, parameters and limitations, and implementation guidance. All three types of appraisal method classes require the appraisal of organisations based on artifacts (e.g. organisational policies and meeting notes) and affirmations, which provide objective evidence as the basis for these appraisals.

Similar to SCAMPI A, the audit process of SEMM, which complies with the IOS/IEC 1702 standard, is well-defined (Garzás et al., 2013). During the auditing of organisations, artifacts and affirmations should be presented as evidence for the institutionalisation of processes. Moreover, the achievement of process attributes is determined with the use of a scale that represents the degree of achievement of components corresponding to a particular process attribute in terms of percentage intervals.

SAMM has a quick start guide, which describes the core steps required to apply the maturity model (OWASP, n.d.-b). As evident from Section 6.2.7, each step of this quick start guide consists of several components, such as its purpose and activities. In order to ensure the objectivity of assessments, each assessment question of SAMM is accompanied with quality criteria, which assist assessors in answering the corresponding question (OWASP, n.d.-c). In case none of the quality criteria is satisfied by the organisation under assessment, the associated question should be answered with "no". Otherwise, one of the other answer options should be selected.

BSIMM also describes how the model can be applied to assess and evolve SSIs of organisations. During an assessment with BSIMM, a team of assessors conducts interviews and studies artifacts in order to determine which software security activities are performed by an organisation.

In contrast to the other discussed maturity models, OMM does not define a process for its application. Nevertheless, OMM consists of several aspects that contribute to the objectivity of its assessments (Petrinja & Succi, 2012). For example, the main components (i.e. TWEs) of OMM are decomposed into smaller components based on the GQM approach accompanying the practices in the model with metrics that assist in the assessment of their fulfilment. Furthermore, the metrics corresponding to a particular practice are assigned a value ranging from 0 to 4, which represents their degree of fulfilment in percentage intervals. Assessors assign these values to the metrics based on available documents related to a FLOSS project under assessment. Based on a defined rating mechanism, the assigned values are aggregated into several scores (e.g. the level of a TWE and the overall maturity level).

All in all, most of the discussed maturity models provide detailed information on their application to support the consistency between different assessments. Therefore, we believe that the application guide that we created for our maturity model could contribute to the consistency between assessments performed with our model. In order to ensure the objectivity of assessments, most models require assessments to be based on objective evidence (e.g. artifacts). Furthermore, the models accompany their assessment questions with additional criteria or indicators (e.g. metrics), or make use of quantitative scales. In order to reduce the subjectivity of assessments of our own model, we, therefore, advice ML teams in our application guide to base their assessment decisions on artifacts that were created during their development processes (i.e. development artifacts).

In addition, we propose the practice cards, i.e., cards that provide detailed information on practices. A practice card describes a practice based on several components: a description of the practice, its purpose, its perceived effect, assessment criteria and related practices. The defined assessment criteria on a practice card of a particular practice aim to assist ML teams in determining to which degree that particular practice is adopted during an assessment. The assessment criteria are derived from grey and academic literature. In case the development process of a team satisfies all assessment criteria of a practice, that practice can be considered to be adopted completely. If none of the assessment criteria are satisfied, the practice should be considered to be not adopted at all. By defining assessment criteria for practices, we thus basically define the extreme ends of the used Likert scale in our maturity model.

In this study, practice cards are only made for a few practices, which serve as an example of how the subjectivity of assessments can be reduced. These practice cards with corresponding assessment criteria can be found in Appendix D.

6.2.9 The design decisions

In the previous sections, we discussed how our maturity model is constructed. Table 6.10 contains the decision parameters corresponding to this construction phase with our made decisions. The decision parameters are derived from Becker et al. (2009), de Bruin et al. (2005), and Mettler (2011).

Table 6.10: The design decision parameters related to the construction phase accompanied by our decisions. These decision parameters are based on Becker et al. (2009), de Bruin et al. (2005), and Mettler (2011).

Decision parameters	Decision
Grounding	Literature & industry (data)
Model structure	Two representations
Maturity definition	Process-focussed
Maturity levels	6
Assessment type	Self-assessment
Assessment base	Artifacts and assessment criteria
Assessors	ML teams
Model forms	Report and Excel workbook

Chapter 7

Model validation

After the development of our maturity model, we validated the model to determine to which extent the developed model assists ML teams in assessing and improving their development processes. In order to validate the model, we conducted a case study involving several ML teams.

7.1 Case study design

The case study was designed by following some of the guidelines of Runeson and Höst (2009). In the case study, we asked ML teams (i.e. the participants) to assess their development processes with our developed maturity model. In order to enable the teams to apply our maturity model in a straightforward manner, we provided the teams with the framework report and the interactive Excel workbook, which were created for our maturity model. The framework report aimed to assist the participating teams in familiarising themselves with the model while the interactive workbook aimed to help the teams to assess their development processes (by filling it in). Before distributing the report and the workbook, we conducted a pilot in which we asked a ML practitioner to conduct a fictional assessment with our maturity model in order to determine whether the model is understandable and whether it is clear how to apply the model with the workbook and whether the workbook functions correctly. Based on the feedback of the practitioner, we revised the materials (i.e. the framework report and workbook).

For the convenience of the participating teams, the teams were asked in the case study to select one of the model representations (i.e. domain or maturity representation) to conduct the assessment. After the teams selected a model representation, the teams were instructed to assign the most applicable adoption degree to each practice in the selected representation in the interactive workbook. Subsequently, the maturity of the teams was automatically determined based on their assigned adoption degrees and presented to them in the workbook.

After the teams conducted assessments with our maturity model, we interviewed the teams in order to obtain their opinion and feedback on the model. The conducted interviews were semi-structured, which allowed us to improvise and expand on the responses of the teams. The interviews were structured according to the funnel model (Runeson & Höst, 2009). Firstly, we presented the purpose of the interview and the case study to the participating teams. Subsequently, we asked the teams some introductory questions of which some were related to their demographics. The introduction was followed by the two main parts of the interview. The first main part of the interview focused on the application of the maturity model, while the second main part focused on its content. The interview questions of the main parts are based on the following criteria that we defined for the validation of our developed maturity model: the comprehensibility, ease of use, completeness, correctness, accuracy, usefulness and feasibility of the model. These validation criteria are derived from Salah et al. (2014) and the Technology Acceptance Model (TAM; Davis, 1989). The TAM specifies when individuals use and accept new technology and how the acceptance of new technology is influenced. According to the model, the intention of individuals to use new technology is affected by their perceived usefulness and ease of use of this technology. For the interviews, we created an interview guide, which can be found in Appendix E.

In order to determine the opinion and feedback of ML teams on our model more precisely, we also administered a questionnaire to the participating teams after the interviews. In our case study, we thus combined different data collection methods, which is also referred to as methodological triangulation. According to Runeson and Höst (2009), triangulation is required when a study mainly relies on qualitative data (as our case study).

In the questionnaire, we asked the teams to summarise their provided opinion and feedback during the interviews by rating the model on some of the defined validation criteria. For these rating questions, we used a Likert scale that ranges from 1 to 5. After each rating question, the teams were asked to motivate their answer (i.e. the selected rating). The questionnaire can be found in Appendix F.

7.2 Results

7.2.1 The participants

Four (ML) teams participated in our case study. Three of these teams (team B, C, D) are all involved in the development of ML-based software systems. One team (team A), however, only assists other teams in the development of the systems by supervising their quality and improving ML development standards. Although team A is not directly involved in the development of the systems, team A has a good overview of how these systems are developed by other teams in their company.

Table 7.1 presents the characteristics of the four participating teams. As shown in this table, three of the four teams work in a non-tech company, while only one team works in a tech company. The companies in which the four teams are working are active in three different industries. Teams A and B both work for companies that are part of the online travel agency industry. The companies that teams C and D work for, are active in the financial services and healthcare industry, respectively. All teams have at least four members.

The teams that participated in our case study are quite experienced. The team members of team B are among the most experienced of all teams with at least 8 years of experience. Team D consists both of experienced and less experienced members with their experience ranging from 1 to 20 years. Except for team A, the teams use different data sources during the development of ML-based software systems. Team B and D both use textual data. Besides the use of textual data, team D uses images. In contrast to the other teams, team C uses tabular data for the development of the systems.

Table 7.1: The characteristics of the teams that participated in the case study

	Organisation type	Industry	Team size	Team experience	Used data
Team A	Non-tech company	Online travel agency	4 ppl.	$\geq$ 4 years	N/A
Team B	Non-tech company	Online travel agency	6 ppl.	$\geq$ 8 years	Textual data
Team C	Non-tech company	Financial services	6 ppl.	$\geq$ 3 years	Tabular data
Team D	Tech company	Healthcare	5-15 ppl.	1-20 years	Images & textual data

7.2.2 The interviews

As stated in Section 7.1, the teams were asked to select one of the two representations for the assessment of their development processes with our maturity model. In the case study, all participating teams provided us with their opinion and feedback on the model based on its application with the domain representation. In order to be able to validate the model as thoroughly as possible, we, however, asked the teams to familiarise themselves with both representations. We present the results of the interviews per validation criteria or topic below.

Comprehensibility In general, most of the teams found the content of the maturity model easy to understand. However, the teams did not understand all aspects of the model directly. For example, team B had to look up the meaning of some practices in the SE4ML catalogue. Some teams did not understand one or more practices used in the model completely. Team A and C both, for example, did not understand practice 34 (automate configuration of model structures). Besides practice 34, team C did not understand practice 43 (explain results and decisions to users). This team found the information in the SE4ML catalogue too theoretical and stated that the practices would be easier to understand when all practices have a corresponding practice card. Team A, B and C did not directly understand how the maturity scores were determined based on their conducted assessments. Therefore, we had to explain the scoring method of our model to these teams during the interviews.

Team D stated the following during the interview about the comprehensibility of the model: *"I think in general it was pretty easy. First, I needed to get used to it and understand the document a bit. But once I got the hang of it, it was not that hard."*

Ease of use All teams were able to assess their development processes with the use of the interactive workbook of our maturity model in an easy manner. However, we found that the required macros for the workbook, which generate the content of some spreadsheets, only function correctly when the workbook is opened in Microsoft Excel. For team A and B, the workbook, therefore, did not work properly as these teams did not have access to Microsoft Excel and opened the workbook in Google Sheets. Furthermore, team C was not able to activate the macros in Microsoft Excel due to particular settings. Therefore, these three teams did not automatically obtain their maturity based on their selected adoption degrees. After these teams completed the assessment, we provided them with another workbook containing their maturity scores.

Team A, B and D found it difficult to distinguish between the different adoption degrees in the used Likert scale. For example, team A stated the following about the ease of use of the model: *"The hardest part was where to put partially and where to put mostly. Like how to differentiate between the two. It is a bit subjective, but in general it was easy"*.

In order to further improve the ease of use of the workbook, both team A and B suggested us to include links for the practices in the representations to their corresponding information pages in the SE4ML catalogue.

We asked three of the four teams (team B, C and D) to estimate how long it took them to familiarise themselves with the maturity model and to conduct the assessment with the workbook. Team B needed approximately one hour to understand the model and conduct the assessment, while teams C and D required between one to one and a half hours.

Completeness The teams found that our maturity model is complete and does not miss any major components or elements. Nevertheless, some of the teams made some minor suggestions for the addition of some elements to further refine the model. As stated before, team A and B both suggested us to include links for the practices in the model to their information pages in the SE4ML catalogue. Team C and D would like to have more information on the practices in order to improve their understandability. Team C, therefore, would like to have practice cards for all practices in the model.

As only for some practices the option "not applicable" is included in the used Likert scale, both team B and C were sometimes forced to select "not at all" in case a practice was not applicable to them. For example, team C stated the following: *"We did think of some practices like use privacy preserving machine learning techniques that they are not applicable to everybody. So then you fill in not at all, but that does not necessarily mean you are not mature. That means that it is not needed or something"*.

Correctness We asked the teams whether they think that the practices per domain in the domain representation are logically organised across the EC levels. Team A found that practice 34 (automate configuration of model structures) in level 2 of the practice domain "Training" is an advanced practice as the adoption of this practice requires the implementation of sophisticated tools. Furthermore, the team found that its difficulty is similar to that of practice 33 (automate feature generation and selection) in level 3. Therefore, team A would move practice 34 from level 2 to level 3. Moreover, team A stated that both practice 27 (remove or archive unused features) and 32 (employ interpretable models when possible) of level 3 are easier to adopt than all practices in that level and therefore would move these practices to another level.

In addition, the team found that the adoption of practices 42 (inform users on ML usage) and 44 (provide safe channels to raise concerns) in level 2 of the practice domain "Governance" would be the responsibility of organisations instead of ML teams. As their adoption would also require less effort than the adoption of practices 21 (enforce fairness and privacy) and 40 (establish responsible AI values) in level 1 according to team A, the team would switch their positions with practices 21 and 40.

Similar to team A, team C would move practices 21 and 40 from level 1 to level 3 as they consider these practices as difficult. Furthermore, they would change the position of practices from level 3 to 1 as the adoption of these practices is common within their industry. Lastly, the team stated that they would move practices 5 (use versioning for data, model, configurations and training scripts) and 8 (write reusable scripts for data cleaning and merging) from the practice domains "Training" and "Data", respectively, to the practice domain "Coding" as they think that these practices are more related to this domain.

Both team B and D did not make any specific suggestions for the change of the positions of the practices, but these teams commented on the division of the practices across EC levels in general. Team B, for example, believes that the division of the practices across levels depends on the context in which the maturity model is used. Team D made the following remark regarding the division of practices across levels: *"There are several points that are clearly beginner and some points that are clearly advanced. But there are also some points that can be interchangeably placed in other directions, but I will not argue against it"*.

Accuracy Two of the four teams (team B and C) found that their obtained maturity scores do not completely represent their actual maturity adequately. For example, team B stated the following about their obtained scores: *"I think in general we are doing pretty good. I know we are lacking a lot of things, but you will not have everything solved. That is not possible. So it does look a bit harsh"*. Furthermore, team C made the following remark about their obtained score regarding the practice domain "Governance": *"I would not expect us to be very low on governance compared to the average data science team"*.

As team A is not directly involved in the development of ML-based software systems, this team only commented on the accuracy of our model in a general manner. This team found that our scoring method in which all practices until a particular level should at least be mostly adopted for that level to be achieved, is strict. Furthermore, team A stated that our scoring method does not allow for an adequate comparison between teams as the scoring method assigns maturity scores without taking into account the number of adopted practices. For example, two different teams could have the same maturity level despite one team having adopted almost all practices until that level and the other team having adopted only a few. Therefore, team A advised us to introduce a metric that indicates the fraction of adopted practices per level in order to be able to compare the maturity of teams with the same level. Similar to team A, team C suggested to adapt the scoring method of our model and to also incorporate practices that are only adopted partially in the (final) maturity scores.

The fourth team (team D) did not understand our question about the accuracy of the model. Therefore, we were not able to determine whether their obtained scores represent their actual maturity.

Usefulness Team B found our maturity model especially helpful for obtaining insights into their development process in terms of their practice adoption. The team believes that these insights could help them reflect on the current state and the improvement of their process. Nevertheless, in this reflection the team would only consider practices that are relevant for them as this team treats the maturity model more as a guideline. Furthermore, team B stated that the two adoption scores (i.e. the practice adoption and adoption percentile) were interesting. However, these scores would not persuade the team to take action, because the scores lack information about the systems developed by the teams associated with these scores. Therefore, team B found that they could not determine whether the practices are relevant for them based on these scores.

For team C, the obtained insights about their development process from the model were interesting, but served more as a confirmation of the current state of their process as the team already had a good overview of their adopted practices. Nevertheless, they found that it was useful to be presented with their practice adoption in a structured manner. Team C made the following remark about the usefulness of the model: *"It's a good reality check and it's always nice to have some things concrete. But was it an eye opener? No... You have a feeling how your team is, right?"*.

Team D is both involved in ML product development projects and exploratory ML projects. According to the team, our maturity model would be especially useful for their ML product development projects. For these projects, team D stated that the model would provide sufficient insights into their development process in order to improve their process. This obtained information from the model can according to the team be used to communicate with third parties about their development process. Similar to team B, however, team D considers their obtained maturity scores as a rough indication of their maturity and would use the model more as a guideline for the improvement of their process.

As team A is not directly involved in the development of ML-based software systems, the question whether the model is considered useful is not asked to this team.

Feasibility We asked two of the four teams (team A and D) whether they think that ML teams would use our maturity model in practice to assess and improve their development processes. For the model to be used in practice, team A believes that some willingness is required from the teams themselves and that these teams should be supported by the leaders of their organisations. Similar to team A, team D stated that teams will only use the model in practice if its use is required by the management of their organisations. For the management of these organisations to adopt the model in practice, the team claimed that the management has to be convinced of the impact that the model has on the productivity of the teams in financial terms.

The representations We asked the teams whether they perceived the distinction between the domain representation and the maturity representation as useful. According to team A, it was useful to be presented with maturity scores related to both representations (i.e. EC levels and maturity levels). Nevertheless, the team found that it was confusing that they were allowed to choose one of the two representations in order to conduct the assessment. Therefore, team A advised us to no longer allow teams to choose between the two representations. According to the team, the model would be easier to use when teams would only be allowed to use the domain representation to conduct assessments. After the assessments, the teams could then be presented with their obtained maturity results with the use of the two representations and be allowed to choose with which representation to improve their processes.

Similar to team A, team C found the distinction between the two representations confusing and did not understand why teams were allowed to choose between them. This team would only present the final results of assessments with the use of these representations.

Team D preferred the domain representation over the maturity representation as this team found that the practices in this representation are divided in a more straightforward manner. After all, the tasks performed by this team during the development of ML-based software systems are also divided among the team members based on the same domains as in the domain representation.

The practice cards Three of the four teams (team A, B and D) did not study the practice cards. During the interviews, we, therefore, presented the practice cards to these teams and asked them for their opinion about these cards. Team A found the practice cards helpful to better understand the practices. However, team A questioned if the criteria on the practice cards capture all tasks related to a particular practice. If a team performs a task related to a practice that is not among its assessment criteria, that task will currently not count towards its adoption. The team, therefore, advised us to use the assessment criteria within our maturity model as examples of tasks related to practices instead of strict requirements that have to be satisfied. Similar to team A, team D questioned the comprehensiveness of the assessment criteria of the practice cards.

Team B and C found that the assessment criteria of the practice cards clarify what the practices entail. However, team B stated that the defined assessment criteria might not be applicable to every development process of teams. The team made the following remark about the use of assessment criteria within our model: *"I am thinking whenever you write assessment criteria for practices in general: how do you make sure that it is applicable to everyone? I do not think that you can say: you need to have A, B, C and D, then you are good"*.

7.2.3 The questionnaire

Table 7.2 presents the ratings that the participating ML teams in our case study gave to our maturity model on part of the defined evaluation criteria. For each evaluation criteria, the average and standard deviation of all corresponding ratings are determined. From Table 7.2, it can be seen that the ease of use and the correctness of our model were rated with an average score of 4.75 and 4.25, respectively, which are the two highest average scores. These scores indicate that our maturity model is perceived as easy to use and correct by the teams. We suspect that the average score corresponding to ease of use of our model can be explained by the fact that teams were provided with an interactive workbook, which enabled them to apply our model in a straightforward manner. The average score regarding the correctness of the model is supported by our findings from the interviews that indicate that teams only had observed minor issues and had suggested small additions to the model.

The completeness and usefulness of our model were both rated with an average score of 4.00 indicating that our model is complete and useful according to the teams. Team A, however, assigned the usefulness of our model a score of 3 out of 5, which is the lowest score corresponding to this criteria. The team motivated this rating in the questionnaire by stating that the usefulness of the model depends on the organisation of teams and their motivation for using the model. Furthermore, team A believes that teams should be supported by the leaders of their organisations to focus on their process improvement for the model to be useful, which was also mentioned by the team during the interview.

The comprehensibility and feasibility of our model were rated the lowest of all criteria with average scores of 3.75 and 3.50, respectively. Team C gave the comprehensibility of our model the lowest score of all teams (3 out of 5). This can be explained by the fact that it required some effort from team C to understand several practices used in our model. All other teams rated the comprehensibility with a score of 4 out of 5.

The feasibility of our model was assigned a score of 3 by both team A and C while the other two teams gave the feasibility a score of 4. This indicates that both team A and C believe that our maturity model is less likely to be adopted in practice by teams for the assessment and improvement of their development processes than team B and D. Team C stated in the questionnaire that they think that our maturity model would only be used in practice if it is required. The score given by team A can be explained by the fact that the team stated during the interview that leadership support and willingness from the teams themselves is required for the model to be used in practice.

Table 7.2: The ratings given to the maturity model on different validation criteria by the ML teams that participated in our case study. For each validation criteria, the average and standard deviation of all corresponding ratings is determined. The design of the table is based on M. Overeem et al. (2022).

	Comprehensibility	**Ease of use**	**Completeness**	**Correctness**	**Usefulness**	**Feasibility**
Team A	4	5	4	4	3	3
Team B	4	4	4	5	4	4
Team C	3	5	4	4	4	3
Team D	4	5	4	4	5	4
Average	3.75	4.75	4.00	4.25	4.00	3.50
Std. Dev.	0.43	0.43	0.00	0.43	0.71	0.50

7.2.4 Requirement satisfaction

In Section 3.4.2, we defined requirements with corresponding contribution arguments for our maturity model. Based on our findings from the case study, we can determine whether our developed model satisfies these requirements. The requirements for our maturity model are stated below. For each defined requirement, we discuss whether that requirement is satisfied.

R1: The maturity model should provide ML teams with insights into the current state of their development processes based on their adoption of best practices.

As described before, three of the four teams (team B, C and D) found our developed maturity model useful for obtaining insights into their development processes in terms of their practice adoption. Team B stated that their practice adoption determined by our model could help them to reflect on the current state of their development process. Team C stated that our model provided them with their current practice adoption in a concrete and structured manner. Nevertheless, their practice adoption determined by the model was only a confirmation for the team as they already knew which practices they had adopted. Team D found that the model clearly specifies which aspects to focus on during the development of ML-based software systems

As team A is not directly involved in the development of ML-based systems, the team does not adopt practices themselves. Therefore, the question whether the model provides insights in the current state of their development process in terms of their practice adoption is not applicable for them.

All in all, we consider requirement 1 (R1) as satisfied.

R2: The maturity model should guide ML teams to their desired state of their development processes.

In the "process improvement" spreadsheet of the workbook of our maturity model, teams can create a roadmap in order to plan the improvement of their development processes in several phases. In order to determine whether the model guides teams to their desired state of their development processes, a study has to be conducted in which teams have to create roadmaps and improve their processes according to these roadmaps. As our conducted case study was, however, only limited to assessments of development processes with our model, none of the participating teams created and executed roadmaps for process improvements. Therefore, we are not able to determine whether the model guides teams to their desired state of their processes. We thus consider the satisfaction of requirement 2 (R2) as inconclusive.

R3: The maturity model should be understandable and applicable with limited effort (i.e. at most three hours to understand the model and conduct an assessment) by ML teams.

As stated in Section 7.2.2, we asked three of the four teams (team B, C and D) how long it took them to understand the maturity model and conduct an assessment. All three teams spent approximately between one and one and a half hours to familiarise themselves with the model and conduct the assessment. Furthermore, all teams stated during the interviews that they were able to apply the model in a straightforward manner despite minor issues (e.g. macros of the workbook not functioning correctly). This is also supported by the average score of 4.75 that the teams gave regarding the ease of use of the model in the questionnaire, which is the highest average score of all rated criteria.

Regarding the comprehensibility of our model, most of the teams stated that in general they did understand the content of our model easily despite that some aspects (e.g. the scoring method) of the model were not directly understood. In comparison to most of the teams, team C had more difficulty understanding our model, especially the practices. This is also evident by the ratings given on the comprehensibility of the model in the questionnaire. Team A, B and D rated the comprehensibility with a score of 4, while team C gave the comprehensibility a score of 3 resulting in an average score of 3.75, which is one of the lowest average scores of all criteria. As only team C had issues with understanding the model, we consider that our model is understandable with limited effort.

Overall, we can conclude that our maturity model satisfies requirement 3 (R3). Nevertheless, the understandability of the model could be further improved as not all aspects of the model were directly clear to all teams and team C had more difficulty understanding the model than other teams.

7.2.5 Model refinements

During our conducted case study, several issues with our developed maturity model emerged. Some of these emerged issues can be addressed with minor adaptations to our model. As described before, three of the four teams did not understand how their maturity was determined based on their conducted assessments (i.e. the scoring method). In order to address this issue and improve the comprehensibility of our model, we suggest to include an explanation of the scoring method of our model in the "assessment report" spreadsheet, which presents the obtained maturity of teams, in the Excel workbook. Some of the teams also did not understand one or more practices in our model (directly). In order to assist teams in understanding the used practices in our model more easily, the practices in the workbook should be linked to their corresponding information pages as suggested by team A and B during the case studies. In addition, each practice could be accompanied with a practice card, which provides detailed information on a practice, as proposed by team C. Team A and D, however, questioned the comprehensiveness of the assessment criteria of the practice cards while team B questioned the applicability of the criteria on every development process.

Furthermore, we found that not all practices in our maturity model are applicable to development processes of all teams. We, therefore, suggest that the option "not applicable" should be included between the possible answer options for more practices than is currently done.

During the case study, we also discovered that two of the four teams found the distinction between the two representations confusing. In order to prevent confusion between the representations, the suggestion of team A could be followed to allow teams to only conduct assessments with the domain representation and to represent the maturity results with the use of the two representations. The decision of teams of which representation to use should then be made only after they have conducted an assessment and have been presented with their maturity results.

Besides the issues that can be addressed with minor adaptations to our model, several issues emerged during the case study that require more effort to address. We discovered that our created Excel workbook is not interoperable. Therefore, the workbook did not function properly for three of the four teams. In order to make our maturity model more easily accessible for teams, the report and workbook of our model could be combined into a web application, which would assist teams in understanding and applying the model easily. Moreover, three of the four teams perceived the selection between adoption degrees in the used Likert scale as difficult. Although the assessment criteria on the practice cards were intended to facilitate the choice between the different adoption degrees, the teams questioned the comprehensiveness and generality of the criteria, as described above. Possible solutions could be to adapt the assessment criteria, create general definitions for adoption degrees or use metrics. However, more research is required on facilitating objective selection between the adoption degrees. Lastly, team B and C found that their maturity scores do not completely correspond to their actual maturity. In addition, team A and C stated that partially adopted practices should be incorporated in the maturity scores. Therefore, more study is required on the accuracy and scoring method of our model.

Chapter 8

Discussion

8.1 The maturity model

We developed a maturity model that aims to support and guide ML teams in the assessment and improvement of their development processes. The developed maturity model consists of 45 practices for the development of ML-based software systems, which are identified in the studies of Serban et al. (2020, 2021). In the maturity model, these practices are arranged into two representations (i.e. views): the domain and maturity representation. The domain representation focuses on the maturity of teams regarding particular practice domains represented by engineering capability levels. The maturity representation describes, however, the overall maturity of the development processes of teams indicated by maturity levels. The domain representation can be used when a team aims to assess and improve its development process regarding specific practice domains. The maturity representation, however, is useful when a team wants to assess and improve its overall development process.

The practices in the domain representation are arranged into levels based, among other things, on their adoption by the 139 teams in our data pool. In order to create this data pool, we processed a dataset that contains the responses to the questionnaires administered during the studies of Serban et al. (2020, 2021). We, for example, removed all questions that do not occur in both questionnaires in order to ensure that all teams in the data pool have answered the same set of questions. Based on the data pool, we defined two scores that represent the adoption of the practices by the teams in the pool.

As the practices were ordered based, among others, on their adoption by the teams, the manner in which we processed the dataset has influenced the ordering of the practices. After all, another way of processing the dataset could have resulted in different adoption scores and another ordering of the practices. Furthermore, as we removed questions that do not occur in both questionnaires, the adoption of some practices is not determined. As a result, we ordered these practices based on their similarity with or dependency on other practices, and their assumed complexity. Our approach of ordering the practices can thus not be fully considered as data-driven. Therefore, this approach is not completely objective and different orderings are possible. Further validation of the model should focus on studying whether our ordering of the practices is acceptable or if the positions of practices should be changed. Ideally, we believe that the arrangement of the practices should be completely based on data.

The levels of the maturity representation consist of sets of EC levels of particular practice domains, which enables users to convert obtained EC levels with the domain representation into maturity levels and vice-versa. Similar to the domain representation, these sets of EC levels of particular practice domains are arranged into the maturity levels based on their adoption, similarity and dependency of their corresponding practices. As a result, this ordering is limited by the same lack of objectivity as the ordering of the practices in the domain representation. Furthermore, as maturity levels are composed of EC levels of practice domains, the choices made in the construction of the domain representation directly affect the composition of maturity levels. Moreover, the composition of maturity level is limited by the fact that practices cannot be individually assigned to maturity levels but should always be assigned together with their associated practices of the EC level of their domain.

Despite the limitations of our ordering approaches, we believe that these approaches could also be used for the development of maturity models with different subjects than our model. Our ordering approaches specify concretely how practices (or similar elements) can be arranged into levels, which is not stated in studies on design approaches for maturity models. These studies often only present several high-level guidelines for particular phases or steps in the development of maturity models. For our approaches to be used for the development of a model with a different subject than our model, domains of that subject with corresponding practices (or similar elements) should be identified. Furthermore, data is required on the adoption of these practices by the targeted audience of the model. Moreover, our model structure with two representations should be followed.

8.2 The model validation

In order to determine to which extent the developed maturity model assists teams in the assessment and improvement of their development processes, we validated the model through a case study involving four ML teams. In the case study, the participating teams were asked to assess their development processes with the model. After the assessment of their processes with the model, the teams were interviewed and asked to complete a questionnaire in order to obtain their opinion and feedback on the model.

With the case study, we found that, in general, the model is positively perceived by the four ML teams. The participating teams consider the model as easy to use and correct. Regarding these criteria, the teams only had minor comments. Some teams had difficulty distinguishing the different answer options in the used Likert scale. Furthermore, two teams suggested small modifications to the order of the practices in the domain representation. In the questionnaire, the teams rated these two criteria with an average score higher than 4.0 out of 5. The completeness and usefulness were rated with an average score of 4.0 out of 5. The teams perceive the model as complete and stated that it does not miss any major components or elements. Some teams would, however, like to be provided with more information about the practices. Most of the teams also find that the model gives them sufficient insights into the current state of their development processes. Nevertheless, two teams would only use the model as a guideline for assessment and improvement of these processes.

The comprehensibility and feasibility of the model were rated lower than the previously discussed criteria with an average score below 4.0 out of 5. During the interviews, most of the teams stated that the content of the model is in general easy to understand despite minor comprehensibility issues. For example, some teams did not understand a few practices and the scoring method directly. However, the lower average score of the comprehensibility can be explained by the fact that it took one of the four teams (team C) more effort to understand some of the practices. For the model to be used in practice, some of the teams indicated that willingness is required from the teams themselves and that leadership support is required, which explains the lower average score for the feasibility. Regarding the accuracy of the model, some teams found that their obtained maturity scores do not completely represent their actual maturity and that the scoring method needs adaptation to allow for better comparison between teams.

As only a limited number of teams (i.e. four teams) participated in our case study, our obtained findings should only be considered as an indication of how ML teams in general perceive the developed model. In order to obtain a more complete view on how ML teams perceive the model, more teams should have participated in our case study. Furthermore, two of the four participating teams were familiar with the studies of Serban et al. (2020, 2021), which could have biassed their opinion and feedback on the model, especially their understanding of our used practices. Moreover, with our case study, we were not able to validate the model in its entirety. For example, we asked teams to use one of the two representations for the assessment of their development processes. As all teams chose the domain representation, teams were not able to provide their opinion and feedback purposefully on model elements related to the maturity representation. Therefore, the opinion and feedback of teams on the maturity representation was limited and excluded from our findings. Finally, we only asked the teams to perform a self-assessment with our developed model. As a result, the teams did not create a roadmap based on their obtained insights from their assessments. We, therefore, could not determine whether allowing the teams to create a roadmap will support and guide these teams in improving their processes. Furthermore, as our model was not implemented in real-world settings, we were not able to determine whether the model is actually effective in improving the development processes of teams.

8.3 The knowledge questions

In Section 3.3, we defined knowledge questions. These knowledge questions were implicitly answered in the previous chapters of this thesis. In this section, we summarise the answers to the knowledge questions briefly.

KQ1: What are the current maturity models in the domain of SE?

We identified the following types of maturity models in the domain of SE: traditional software development models, agile software development models, software security models and FLOSS development models. For each category, we elaborated on one or two corresponding maturity models in Chapter 4. We focused especially on models of which the design or development is documented extensively. In total, we discussed seven maturity models in the domain of SE. These models are presented per model type in Table 8.1. It is important to note that there are more maturity models available in the domain of SE than discussed in this thesis and presented in the table.

One of the most applied models in the domain of SE is CMMI-DEV, which guides organisations in the improvement of their processes for the development of products and services. By comparing the identified models, we found that most models have a prescriptive purpose of use and are grounded in knowledge and experience from industry. Assessments with these models are mostly performed by teams of assessors, which base their assessment decisions on artifacts.

Table 8.1: The identified maturity models in the domain of SE per model type

Model type	Models
Traditional software development	CMMI-DEV and SEMM
Agile software development	SAMI and SAFe MM
Software security	BSIMM and SAMM
FLOSS development	OMM

KQ2: What approaches are available for the development of maturity models?

In Chapter 5, we discussed six approaches for the development of maturity models. We distinguish two types of development approaches: approaches for general and specific models. Table 8.2 presents the identified development approaches for each type of approach. Similar to maturity models in the domain of SE, there are probably more approaches than discussed in this thesis.

Most of the identified approaches describe the development of maturity models as a process that consists of several steps or phases. We refer to such approaches as development process frameworks. By comparing a set of process frameworks, we identified the following common phases in the development of maturity models: scoping, construction, validation, deployment and revision. These process frameworks and the identified common phases guided the development of our own maturity model. Due to time constraints, we focused in this thesis only on the scoping, construction and validation phases.

Table 8.2: The identified development approaches per approach type

Approach type	Development approaches
Approaches for general models	De Bruin et al. (2005)
	Mettler (2011)
	Becker et al. (2009)
	Pöppelbuß and Röglinger (2011)
Approaches for specific models	Maier et al. (2012)
	Van Steenbergen et al. (2010))

KQ3: How can the adoption of engineering best practices in the development processes of ML teams be operationalised?

The operationalisation of the adoption of engineering best practices in the development processes of ML teams is our developed maturity model. The development of our maturity model is extensively described in Chapter 6. As described before, our maturity model consists of 45 engineering best practices for the development of ML-based software systems. These practices are arranged into two representations: the domain and maturity representation. The domain representation focuses on the maturity of teams regarding the practice domains (e.g. data, training and coding) represented by EC levels while the maturity representation focuses on the overall maturity of the development process of teams represented by maturity levels. Including the initial levels, the model defines 4 EC levels and 6 maturity levels.

Our maturity model comes in the form of a report and an interactive Excel workbook, which aim to assist teams in applying the model. In order to apply the model and conduct an assessment, teams should select one of the two representations or both representations for the assessment. For each practice, the teams should determine to which degree a practice is adopted by them with the use of a Likert scale. The Likert scale distinguishes the following adoption degrees: not at all, partially, mostly and completely. For some practices, teams are also allowed to select the answer "not applicable" to indicate that a practice is not applicable to their development process. A particular level is accomplished when all practices corresponding to that level and all practices of any lower levels are assigned an adoption degree of at least "mostly". Based on the insights of assessments, teams could create a roadmap in order to plan their process improvement in several phases. The execution of assessments and the creation of roadmaps can be done in the interactive workbook of our model.

KQ4: How accurate and relevant is the created maturity model for the development processes of ML teams?

We validated our developed maturity model through a case study of which the results are presented and discussed in Chapter 7. In the case study, two teams stated regarding the accuracy of our model that their obtained maturity scores do not completely represent their actual maturity. Some teams also indicated that the scoring method used in the model should be adapted in order to allow for comparison between teams. After all, with our current scoring method, teams could have the same maturity scores despite one team having adopted almost all practices until a particular level and the other team having only adopted a few practices until that level.

Most of teams stated in the case study that our model provides them with sufficient insights into their development processes. Two teams, however, stated that they would only use the model as a guideline for the assessment and improvement of their processes. For example, one of these teams commented that they would only focus on practices that they perceive relevant for their development process. In the questionnaire, the teams rated the usefulness of the model with an average score of 4.00 out of 5.

Although most teams found the model useful, some teams indicated that ML teams would not use such models by themselves and that some leadership support is required for the models to be used in practice.

Overall, our maturity model is thus not completely accurate due to our devised scoring method. Nevertheless, we can conclude that the model is relevant as it provides teams with sufficient insights into their processes. For our model to be used in practice, ML teams should, however, be supported by the leaders of their organisations.

Chapter 9

Conclusion

In this thesis, the development and validation of a maturity model for ML-based software systems is extensively described. By developing a maturity model for ML-based software systems, we present ML teams with a framework that provides them with insights into the current state of their development processes in terms of their practice adoption. Based on these insights, we support teams in developing, deploying and maintaining ML-based software systems more robustly and responsibly. During the development of the model, we devised approaches to order the practices into levels. These approaches, among others, involve the use of adoption data corresponding to these practices. With the development of the maturity model, we thus demonstrate how engineering best practices for ML-based software systems in literature can be structured into a coherent and practical format that assists teams to actually operationalise these practices. The developed maturity model and the used ordering approaches could, therefore, serve as an example for the operationalisation of best practices and the development of maturity models in other domains.

By validating our developed maturity model, we determined how such a model is perceived by ML teams. Overall, the obtained findings indicate that in general ML teams have a positive impression of our model. Nevertheless, we found that the implementation of the model in practice needs leadership support and willingness of teams. Therefore, more research is required that addresses how to create support among practitioners and stakeholders (e.g. management) to apply such models in practice.

In future research, the developed maturity model could be revised based on our proposed refinements in Section 7.2.5. For example, the report and the Excel workbook corresponding to the model could be combined into a web application in order to improve its interoperability. Furthermore, the model could in future research be further validated by asking more teams to provide their opinion and feedback on the model. This validation study could, for example, be conducted after the proposed refinements are implemented in the model. Another validation study on the developed model should especially focus on elements of the model that are not covered in depth by our validation in order to ensure that the model is validated as completely as possible and to obtain a more complete view of the opinion and feedback of ML teams. Moreover, the adoption of practices that currently have no adoption scores could be determined in future research. In order to incorporate the adoption of these practices into the model, we suggest collecting new data on the adoption of all practices. Based on this new data pool, the ordering of the practices can be revised, which results in another version of the model. This reordering will improve the data-driven nature of the development of our model.

Finally, the model could in future research be implemented in real-world settings to study whether the model is actually effective in improving the development processes of teams (i.e. evaluation of the model). This study could entail an assessment of the processes and a creation of a roadmap with the model that teams should follow to improve their development processes. The teams should be monitored over a longer period of time to determine their progress and evaluate whether the model is effective.

References

About ML. (2021). About ml reference document. https://partnershiponai.org/paper/about-ml-reference-document/12/#Section-5

Akkiraju, R., Sinha, V., Xu, A., Mahmud, J., Gundecha, P., Liu, Z., Liu, X., & Schumacher, J. (2020). Characterizing machine learning processes: A maturity framework. In D. Fahland, C. Ghidini, J. Becker, & M. Dumas (Eds.), *Business process management* (pp. 17–31). Springer. https://doi.org/10.1007/978-3-030-58666-9_2

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., & Zimmermann, T. (2019). Software engineering for machine learning: A case study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Aston, B. (2022). 10 best product backlog tools for backlog management. https://theproductmanager.com/tools/best-product-backlog-tools/

Becker, J., Knackstedt, R., & Pöppelbuß, J. (2009). Developing maturity models for it management. *Business & Information Systems Engineering*, *1*, 213–222. https://doi.org/10.1007/s12599-009-0044-5

Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2017). The ML test score: A rubric for ML production readiness and technical debt reduction. *2017 IEEE International Conference on Big Data (Big Data)*, 1123–1132. https://doi.org/10.1109/BigData.2017.8258038

CMMI Product Team. (2010). CMMI for Development, Version 1.3. https://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, *13*(3), 319–340. https://doi.org/10.2307/249008

de Bruin, T., Rosemann, M., Freeze, R., & Kaulkarni, U. (2005). Understanding the main phases of developing a maturity assessment model. In D. Bunker, B. Campbell, & J. Underwood (Eds.), *Australasian Conference on Information Systems (ACIS)* (pp. 8–19). Australasian Chapter of the Association for Information Systems.

Fontana, R. M., Albuquerque, R., Luz, R., Moises, A. C., Malucelli, A., & Reinehr, S. (2018). Maturity models for agile software development: What are they? In X. Larrucea, I. Santamaria, R. V. O'Connor, & R. Messnarz (Eds.), *Systems, software and services process improvement* (pp. 3–14). Springer.

Fontana, R. M., Fontana, I. M., da Rosa Garbuio, P. A., Reinehr, S., & Malucelli, A. (2014). Processes versus people: How should agile software development maturity be defined? *Journal of Systems and Software*, *97*, 140–155. https://doi.org/10.1016/j.jss.2014.07.030

Fontana, R. M., Meyer, Jr., V., Reinehr, S., & Malucelli, A. (2015). Progressive outcomes: A framework for maturing in agile software development. *Journal of Systems and Software*, *102*, 88–108. https://doi.org/10.1016/j.jss.2014.12.032

Garzás, J., Pino, F. J., Piattini, M., & Fernández, C. M. (2013). A maturity model for the spanish software industry based on iso standards. *Computer Standards & Interfaces*, *35*(6), 616–628. https://doi.org/10.1016/j.csi.2013.04.002

Giray, G. (2021). A software engineering perspective on engineering machine learning systems: State of the art and challenges. *Journal of Systems and Software*, *180*, Article 111031. https://doi.org/https://doi.org/10.1016/j.jss.2021.111031

Hazelwood, K., Bird, S., Brooks, D., Chintala, S., Diril, U., Dzhulgakov, D., Fawzy, M., Jia, B., Jia, Y., Kalro, A., Law, J., Lee, K., Lu, J., Noordhuis, P., Smelyanskiy, M., Xiong, L., & Wang, X. (2018). Applied machine learning at Facebook: A datacenter infrastructure perspective. *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 620–629. https://doi.org/10.1109/HPCA.2018.00059

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105.

Humphrey, W. (1988). Characterizing the software process: A maturity framework. *IEEE Software*, *5*(2), 73–79. https://doi.org/10.1109/52.2014

Ishikawa, F., & Yoshioka, N. (2019). How do engineers perceive difficulties in engineering of machine-learning systems?: Questionnaire survey. *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, 2–9. https://doi.org/10.1109/CESSER-IP.2019.00009

Jain, A., Patel, H., Nagalapatti, L., Gupta, N., Mehta, S., Guttula, S., Mujumdar, S., Afzal, S., Mittal Sharma, R., & Munigala, V. (2020). Overview and importance of data quality for machine learning. https://www.slideshare.net/HimaPatel2/overview-and-importance-of-data-quality-for-machine-learning-tasks

JetBrains. (n.d.). CI/CD best practices. https://www.jetbrains.com/teamcity/ci-cd-guide/ci-cd-best-practices/

John, M. M., Olsson, H. H., & Bosch, J. (2021). Towards MLOps: A framework and maturity model. In M. T. Baldassarre, G. Scanniello, & A. Skavhaug (Eds.), *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (pp. 1–8). IEEE. https://doi.org/10.1109/SEAA53835.2021.00050

Liu, H., Eksmo, S., Risberg, J., & Hebig, R. (2020). Emerging and changing tasks in the development process for machine learning systems, 125–134. https://doi.org/10.1145/3379177.3388905

Lwakatare, L. E., Raj, A., Bosch, J., Olsson, H. H., & Crnkovic, I. (2019). A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In P. Kruchten, S. Fraser, & F. Coallier (Eds.), *Agile processes in software engineering and extreme programming* (pp. 227–243). Springer. https://doi.org/10.1109/ESEM.2019.8870157

Lwakatare, L. E., Raj, A., Crnkovic, I., Bosch, J., & Olsson, H. H. (2020). Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions. *Information and Software Technology*, *127*, Article 106368. https://doi.org/10.1016/j.infsof.2020.106368

Maier, A. M., Moultrie, J., & Clarkson, P. J. (2012). Assessing organizational capabilities: Reviewing and guiding the development of maturity grids. *IEEE Transactions on Engineering Management*, *59*(1), 138–159. https://doi.org/10.1109/TEM.2010.2077289

Mateo-Casalí, M. Á., Fraile, F., Boza, A., & Nazarenko, A. (2023). Maturity model for analysis of machine learning operations in industry. In F. P. García Márquez, I. Segovia Ramírez, P. J. Bernalte Sánchez, & A. Muñoz del Río (Eds.), *Iot and data science in engineering management* (pp. 321–328). Springer. https://doi.org/https://doi.org/10.1007/978-3-031-27915-7_57

Mettler, T. (2011). Maturity assessment models: A design science research approach. *International Journal of Society Systems Science*, *3*(1-2), 81–98. https://doi.org/10.1504/IJSSS.2011.038934

Nascimento, E. d. S., Ahmed, I., Oliveira, E., Palheta, M. P., Steinmacher, I., & Conte, T. (2019). Understanding development process of machine learning systems: Challenges and solutions. *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–6. https://doi.org/10.1109/ESEM.2019.8870157

Otto, L., Bley, K., & Harst, L. (2020). Designing and evaluating prescriptive maturity models: A design science-oriented approach. In W. Guédria, H. A. Proper, J. Verelst, S. Hacks, F. Timm, K. Sandkuhl, M. Fellmann, G. Serapiao, M. Payan, M. Komarov, S. Maltseva, S. Uskenbayeva, D. Nazarov, D. Ge, M. Helfert, & L. Ehrlinger (Eds.), *2020 IEEE 22nd Conference on Business Informatics (CBI)* (pp. 40–47). IEEE. https://doi.org/10.1109/CBI49978.2020.10056

Overeem, B. (2014). 10 best practices for managing the product backlog. https://medium.com/the-liberators/10-best-practices-for-managing-the-product-backlog-a3e5502cc9ff

Overeem, M., Mathijssen, M., & Jansen, S. (2022). API-m-FAMM: A focus area maturity model for API Management. *Information and Software Technology*, *147*, 106890. https://doi.org/10.1016/j.infsof.2022.106890

OWASP. (n.d.-a). About us. Retrieved November 10, 2023, from https://owaspsamm.org/about/

OWASP. (n.d.-b). Quick start guide for version 2.0. https://owaspsamm.org/guidance/quick-start-guide/

OWASP. (n.d.-c). SAMM assessment. https://owaspsamm.org/assessment/

Pei, K., Cao, Y., Yang, J., & Jana, S. (2019). Deepxplore: Automated whitebox testing of deep learning systems. *Communications of the ACM*, *62*(11), 137–145. https://doi.org/10.1145/3361566

Petrinja, E., & Succi, G. (2012). Assessing the open source development processes using OMM. *Advances in Software Engineering*, *2012*, 1–17. https://doi.org/10.1155/2012/235392

Polyzotis, N., Roy, S., Whang, S. E., & Zinkevich, M. (2018). Data lifecycle challenges in production machine learning: A survey. *ACM SIGMOD Record*, *47*(2), 17–28. https://doi.org/10.1145/3299887.3299891

Pöppelbuß, J., & Röglinger, M. (2011). What makes a useful maturity model? a framework of general design principles for maturity models and its demonstration in business process management. *ECIS 2011 Proceedings*, 1–13. https://aisel.aisnet.org/ecis2011/28/

Raeder, T., Stitelman, O., Dalessandro, B., Perlich, C., & Provost, F. (2012). Design principles of massive, robust prediction systems. *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1357–1365. https://doi.org/10.1145/2339530.2339740

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. https://doi.org/10.1007/s10664-008-9102-8

Salah, D., Paige, R., & Cairns, P. (2014). An evaluation template for expert review of maturity models. In A. Jedlitschka, P. Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, & M. Raatikainen (Eds.), *Product-focused software process improvement* (pp. 318–321). Springer. https://doi.org/10.1007/978-3-319-13835-0_31

SCAMPI Upgrade Team. (2011a). Appraisal Requirements for CMMI, Version 1.3. https://insights.sei.cmu.edu/documents/866/2011_005_001_15383.pdf

SCAMPI Upgrade Team. (2011b). Standard CMMI Appraisal Method for Process Improvement (SCAMPI) A, Version 1.3: Method definition document. https://insights.sei.cmu.edu/documents/1618/2011_002_001_15311.pdf

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2020). Adoption and effects of software engineering best practices in machine learning. *ESEM '20: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 1–12. https://doi.org/10.1145/3382494.3410681

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2021). Practices for engineering trustworthy machine learning applications. *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*, 97–100. https://doi.org/10.1109/WAIN52551.2021.00021

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2023a). Assign an owner to each feature and document its rationale. https://se-ml.github.io/best_practices/02-doc_features/

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2023b). Check that input data is complete, balanced and well distributed. https://se-ml.github.io/best_practices/01-input-data-complete/

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2023c). Establish responsible ai values. https://se-ml.github.io/best_practices/06-code_conduct/

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2023d). Provide audit trails. https://se-ml.github.io/best_practices/04-audit_trails/

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2023e). Use continuous integration. https://se-ml.github.io/best_practices/03-cont-int/

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2023f). Use sanity checks for all external data sources. https://se-ml.github.io/best_practices/01-sanity_check/

Serban, A., Van der Blom, K., Hoos, H., & Visser, J. (2023g). Work against a shared backlog. https://se-ml.github.io/best_practices/05-use_backlog/

Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, *5*, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

Sidky, A., Arthur, J., & Bohner, S. (2007). A disciplined approach to adopting agile practices: The agile adoption framework. *Innovations in Systems and Software Engineering*, *3*, 203–216. https://doi.org/10.1007/s11334-007-0026-z

Synopsys. (2022). Bsimm13: Foundations report 2022. https://www.synopsys.com/software-integrity/engage/bsimm-web/bsimm13-foundations

The Free Software Foundation. (2023). What is free software? https://www.gnu.org/philosophy/free-sw.html.en

Turetken, O., Stojanov, I., & Trienekens, J. J. M. (2017). Assessing the adoption level of scaled agile development: A maturity model for scaled agile framework. *Journal of Software: Evolution and Process*, *29*(6), Article e1796. https://doi.org/10.1002/smr.1796

van Steenbergen, M., Bos, R., Brinkkemper, S., van de Weerd, I., & Bekkers, W. (2010). The design of focus area maturity models. In J. L. Winter Robertand Zhao & S. Aier (Eds.), *Global perspectives on design science research* (pp. 317–332). Springer. https://doi.org/10.1007/978-3-642-13335-0_22

Visual Paradigm. (n.d.). What is deep in product backlog? https://www.visual-paradigm.com/scrum/what-is-deep-in-agile-product-backlog/

Vom Brocke, J., Hevner, A., & Maedche, A. (2020). Introduction to design science research. In J. Vom Brocke, A. Hevner, & A. Maedche (Eds.), *Design science research. cases* (pp. 1–13). Springer. https://doi.org/10.1007/978-3-030-46781-4_1

Wan, Z., Xia, X., Lo, D., & Murphy, G. C. (2021). How does machine learning change software development practices? *IEEE Transactions on Software Engineering*, *47*(9), 1857–1871. https://doi.org/10.1109/TSE.2019.2937083

Washizaki, H., Uchida, H., Khomh, F., & Guéhéneuc, Y.-G. (2019). Studying software engineering patterns for designing machine learning systems. *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, 49–495. https://doi.org/10.1109/IWESEP49350.2019.00017

Wieringa, R. J. (2014). *Design science methodology for information systems and software engineering*. Springer. https://doi.org/10.1007/978-3-662-43839-8

Zhang, H., Cruz, L., & van Deursen, A. (2022). Code smells for machine learning applications. *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*, 217–228. https://doi.org/10.1145/3522664.3528620

Zinkevich, M. (2021). Rules of machine learning: Best practices for ml engineering. https://developers.google.com/machine-learning/guides/rules-of-ml

Appendix A

The domain representation

Table A.1: The domain representation. The representation (i.e. view) specifies the maturity progression of teams regarding the practice domains represented by EC levels. The practices of each practice domain are divided across three EC levels. EC level 0, which serves as the initial level, does not consist of practices and is thus omitted in this table.

DATA

Practice	ID	Practice adoption	Adoption percentile
Engineering capability level 1 - Beginner			
Write reusable scripts for data cleaning and merging.	DAT1.1	85	75.0
Engineering capability level 2 - Intermediate			
Ensure data labelling is performed in a strictly controlled process.	DAT2.1	79	60.7
Make data sets available on shared infrastructure (private or public).	DAT2.2	75	57.1
Engineering capability level 3 - Advanced			
Use sanity checks for all external data sources.	DAT3.1	47	14.3
Use privacy-preserving machine learning techniques.	DAT3.2	-	-
Test for social bias in training data.	DAT3.3	-	-
Check that input data is complete, balanced and well distributed.	DAT3.4	48	17.9
Prevent discriminatory data attributes used as model features.	DAT3.5	-	-

Table A.1: The domain representation. The representation (i.e. view) specifies the maturity progression of teams regarding the practice domains represented by EC levels. The practices of each practice domain are divided across three EC levels. EC level 0, which serves as the initial level, does not consist of practices and is thus omitted in this table. (Continued)

TRAINING

Practice	ID	Practice adoption	Adoption percentile
Engineering capability level 1 - Beginner			
Share a clearly defined training objective within the team.	TRAIN1.1	97	96.4
Capture the training objective in a metric that is easy to measure and understand.	TRAIN1.2	102	100.0
Continuously measure model quality and performance.	TRAIN1.3	96	91.1
Enable parallel training experiments.	TRAIN1.4	96	91.1
Use versioning for data, model, configurations and training scripts.	TRAIN1.5	95	83.9
Share status and outcomes of experiments within the team.	TRAIN1.6	90	78.6
Engineering capability level 2 - Intermediate			
Peer review training scripts.	TRAIN2.1	66	46.4
Test all feature extraction code.	TRAIN2.2	53	25.0
Automate hyper-parameter optimisation.	TRAIN2.3	50	21.4
Automate configuration of algorithms or model structure.	TRAIN2.4	-	-
Engineering capability level 3 - Advanced			
Assign an owner to each feature and document its rationale.	TRAIN3.1	28	3.6
Actively remove or archive features that are not used.	TRAIN3.2	37	7.1
Automate feature generation and selection.	TRAIN3.3	-	-
Assess and manage subgroup bias.	TRAIN3.4	-	-
Employ interpretable models when possible.	TRAIN3.5	-	-

CODING

Practice	ID	Practice adoption	Adoption percentile
Engineering capability level 1 - Beginner			
Use static analysis to check code quality.	CODE1.1	60	39.3
Engineering capability level 2 - Intermediate			
Run automated regression tests.	CODE2.1	42	10.7
Assure application security.	CODE2.2	-	-
Engineering capability level 3 - Advanced			
Use continuous integration.	CODE3.1	-	-

Table A.1: The domain representation. The representation (i.e. view) specifies the maturity progression of teams regarding the practice domains represented by engineering capability levels. The practices of each practice domain are divided across three engineering capability levels. Engineering capability level 0, which serves as the initial level, does not consist of practices and is thus omitted in this table. (Continued)

DEPLOYMENT

Practice	ID	Practice adoption	Adoption percentile
Engineering capability level 1 - Beginner			
Continuously monitor the behaviour of deployed models.	DEPLOY1.1	80	67.9
Engineering capability level 2 - Intermediate			
Automate model deployment.	DEPLOY2.1	68	50
Enable automatic roll backs for production models.	DEPLOY2.2	72	53.6
Enable shadow deployment.	DEPLOY2.3	54	30.4
Perform checks to detect skew between models.	DEPLOY2.4	62	42.9
Log production predictions with the model's version and input data.	DEPLOY2.5	56	35.7
Engineering capability level 3 - Advanced			
Provide audit trails.	DEPLOY3.1	-	-

TEAM

Practice	ID	Practice adoption	Adoption percentile
Engineering capability level 1 - Beginner			
Use a collaborative development platform.	TEAM1.1	95	83.9
Engineering capability level 2 - Intermediate			
Communicate, align, and collaborate with others.	TEAM2.1	80	67.9
Work against a shared backlog.	TEAM2.2	80	67.9
Engineering capability level 3 - Advanced			
Decide trade-offs through defined team process.	TEAM3.1	-	-

GOVERNANCE

Practice	ID	Practice adoption	Adoption percentile
Engineering capability level 1 - Beginner			
Establish responsible AI values.	GOVERN1.1	-	-
Enforce fairness and privacy.	GOVERN1.2	54	30.4
Engineering capability level 2 - Intermediate			
Inform users on machine learning usage.	GOVERN2.1	-	-
Explain results and decisions to users.	GOVERN2.2	-	-
Provide safe channels to raise concerns.	GOVERN2.3	-	-
Engineering capability level 3 - Advanced			
Perform risk assessments.	GOVERN3.1	-	-
Have your application audited.	GOVERN3.2	-	-

Appendix B

The maturity representation

Table B.1: The maturity representation. The representation presents the overall maturity progression of the development processes of ML teams represented by maturity levels. Our maturity model defines six maturity levels. Each maturity level comprises a set of EC levels of particular practice domains (except maturity level 1). The overall maturity of teams is considered to start at maturity level 1 and, therefore, this level is omitted in this table.

Maturity representation

Practice	ID	Practice adoption	Adoption percentile
Maturity level 2 - Basic			
Share a clearly defined training objective within the team.	TRAIN1.1	97	96.4
Capture the training objective in a metric that is easy to measure and understand.	TRAIN1.2	102	100.0
Use a collaborative development platform.	TEAM1.1	95	83.9
Write reusable scripts for data cleaning and merging.	DAT1.1	85	75.0
Continuously measure model quality and performance.	TRAIN1.3	96	91.1
Share status and outcomes of experiments within the team.	TRAIN1.6	90	78.6
Use versioning for data, model, configurations and training scripts.	TRAIN1.5	95	83.9
Enable parallel training experiments.	TRAIN1.4	96	91.1
Maturity level 3 - Progressed			
Ensure data labelling is performed in a strictly controlled process.	DAT2.1	79	60.7
Continuously monitor the behaviour of deployed models.	DEPLOY1.1	80	67.9
Communicate, align, and collaborate with others.	TEAM2.1	80	67.9
Work against a shared backlog.	TEAM2.2	80	67.9
Make data sets available on shared infrastructure (private or public).	DAT2.2	75	57.1
Maturity level 4 - Optimised			
Automate model deployment.	DEPLOY2.1	68	50.0
Enable automatic roll backs for production models.	DEPLOY2.2	72	53.6
Enable shadow deployment.	DEPLOY2.3	54	30.4
Perform checks to detect skew between models.	DEPLOY2.4	62	42.9
Log production predictions with the model's version and input data.	DEPLOY2.5	56	35.7
Automate hyper-parameter optimisation.	TRAIN2.3	50	21.4
Automate configuration of algorithms or model structure.	TRAIN2.4	-	-
Use static analysis to check code quality.	CODE1.1	60	39.9
Peer review training scripts.	TRAIN2.1	66	46.6
Test all feature extraction code.	TRAIN2.2	53	25.0

Table B.1: The maturity representation. The representation presents the overall maturity progression of the development processes of ML teams represented by maturity levels. Our maturity model defines six maturity levels. Each maturity level comprises a set of EC levels of particular practice domains (except maturity level 1). The overall maturity of teams is considered to start at maturity level 1 and, therefore, this level is omitted in this table. (Continued)

Practice	**ID**	**Practice adoption**	**Adoption percentile**
Establish responsible AI values.	GOVERN1.1	-	-
Enforce fairness and privacy.	GOVERN1.2	54	30.4
Maturity level 5 - Proactively optimised			
Use sanity checks for all external data sources.	DAT3.1	47	14.7
Run automated regression tests.	CODE2.1	42	10.7
Assure application security.	CODE2.2	-	-
Use privacy-preserving machine learning techniques.	DAT3.2	-	-
Test for social bias in training data.	DAT3.3	-	-
Check that input data is complete, balanced and well distributed.	DAT3.4	48	17.9
Inform users on machine learning usage.	GOVERN2.1	-	-
Explain results and decisions to users.	GOVERN2.2	-	-
Provide safe channels to raise concerns.	GOVERN2.3	-	-
Prevent discriminatory data attributes used as model features.	DAT3.5	-	-
Maturity level 6 - Perfected			
Use continuous integration.	CODE3.1	-	-
Assign an owner to each feature and document its rationale.	TRAIN3.1	28	3.6
Actively remove or archive features that are not used.	TRAIN3.2	37	7.1
Automate feature generation and selection.	TRAIN3.3	-	-
Assess and manage subgroup bias.	TRAIN3.4	-	-
Employ interpretable models when possible.	TRAIN3.5	-	-
Decide trade-offs through defined team process.	TEAM3.1	-	-
Perform risk assessments.	GOVERN3.1	-	-
Provide audit trails.	DEPLOY3.1	-	-
Have your application audited.	GOVERN3.2	-	-

Appendix C

The application guide

Table C.1: The application guide. The application guide aims to support ML teams in applying our maturity model. The guide describes the application of the maturity model as a process of five steps. Each step is decomposed into a number of components, which provide more information on that step. These components are based on the components of the steps in the quick start guide of SAMM.

1	Plan and prepare the assessment
Purpose	Align the team on the assessment initiative
Activities	**1.1 Determine assessment scope** – Determine which aspects of the development process should be covered in the assessment. In case a team is involved in the development of several ML-based software systems, a team should determine which development process is going to be assessed (e.g. process related to particular project or software system). **1.2 Determine assessment objective(s)** – Determine as a team the objective(s) to be achieved as a result of the assessment. **1.3 Set assessment date** – Plan as a team the date on which the assessment will be conducted. **1.4 Identify development artifacts** – Identify which development artifacts could be useful for the assessment.
Resources	• Framework report
Practices	• Involve all members of the team in the assessment initiative. • Ensure that maturity assessment framework (i.e. maturity model) is understood within the team. • Evaluate the practices within maturity assessment framework before the assessment.

2	Conduct the assessment
Purpose	Determine and understand the current maturity of the development process
Activities	**2.1 Select framework representation(s)** – Determine which representation(s) to use for the assessment. The maturity framework consists of two representations: the domain and maturity representations. For an assessment, the domain representation, the maturity representation or both representations could be used. The domain representation focuses on the maturity regarding the practice domains while the maturity representation focuses on the overall maturity of the development process. **2.2 Evaluate practices and assign adoption degrees** – Determine the current adoption of the practices within the development process (e.g. based on development artifacts). For each practice, assign the extent to which the practice is adopted. Within the maturity framework, four adoption degrees are distinguished: not at all, partially, mostly and completely. Some practices within the framework (i.e. DAT2.1, DAT3.1, DAT3.2, DAT3.3, DAT3.5, TRAIN2.2, TRAIN3.1, TRAIN3.2, TRAIN3.3, TRAIN3.4 and GOVERN1.2) are not applicable to all development processes. Teams for which these practices are not applicable, should exclude these practices from the assessment. **2.3 Determine achieved maturity** – Determine the achieved level(s) based on the assignment of the practice degrees. For a particular level to be achieved, the adoption degree of all applicable practices under that particular level and all lower levels (if any) should be at least mostly. In case the interactive workbook is used for the assessment, the obtained level(s) is/are automatically generated and shown on the "Assessment report" spreadsheet.
Resources	• Interactive workbook • Practice cards • Development process artifacts (e.g. source code and notes)
Practices	• Ensure that all practices are understood in detail. • Assign adoption degrees to practices based on development artifacts as much as possible. • Reach consensus within the team on adoption degrees of practices.

Table C.1: The application guide. The application guide aims to support ML teams in applying our maturity model. The guide describes the application of the maturity model as a process of five steps. Each step is decomposed into a number of components, which provide more information on that step. These components are based on the components of the steps in the quick start guide of SAMM. (Continued)

3	**Set maturity target(s)**
Purpose	Define targets regarding the maturity of the development process in order to guide its improvement.
Activities	3.1 **Determine desired level(s)** – Determine which level(s) the team desires to achieve. In case the domain representation is used, the desired engineering capability level for one or more domains should be defined. In case the maturity representation is used, the desired overall maturity of the development process should be defined. If both representations are used for an assessment, desired engineering capability levels, the overall maturity level or a combination of both can be defined. 3.2 **Identify required practices for set target(s)** – identify which practices need to be adopted or of which practices the adoption degree needs to be improved in order to achieve set target(s).
Resources	• Interactive workbook
Practices	• Ensure that set maturity target(s) is/are achievable and reasonable.

4	**Plan process improvement**
Purpose	Create a roadmap which supports the accomplishment of the set maturity target(s).
Activities	4.1 **Determine number of phases and their duration** – Determine in how many phases the set maturity target(s) should be achieved. Besides, the duration of each phase should be specified. 4.2 **Set adoption degree targets for identified practices** – Determine for all identified practices the adoption degrees the team desires to achieve at the end of the improvement initiative. 4.3 **Identify required improvement activities for achieving adoption degree targets** – Determine what is needed in order to accomplish the desired adoption degrees of the identified practices in activity 3.2 4.4 **Define the phases** – Distribute the identified practices over the phases. Determine for each phase the set of practices to work on (a practice could be worked on during multiple phases). Besides, determine for each practice per phase the adoption degree that needs to be achieved at the end of that phase.
Resources	• Interactive workbook
Practices	• Ensure that set adoption degrees of are achievable and reasonable. • Balance the number of practices over the phases. • Focus first on practices of lower levels and then on higher level practices. • Take dependencies between practices into account. • Take the adoption scores (i.e. practice adoption and adoption percentile) into account.

5	**Execute process improvement plan**
Purpose	Improve process according to created roadmap.
Activities	5.1 **Implement improvements** – Improve the adoption of the identified practices according to the created roadmap.
Resources	-
Practices	-

Appendix D

The practice cards

Table D.1: The practice cards. The practice cards provide detailed information on practices, such as their purposes and perceived effects. Furthermore, the practice cards contain assessment criteria. The practice cards aims to reduce the subjectivity of assessments by assessment criteria, which assist ML teams in determining their degree of adoption of the practices. If all assessment criteria of a particular practice are met, the practice should be considered to be completely adopted. In case none of the defined criteria are satisfied, that practice should be considered to be not adopted at all.

DAT3.1	**Use sanity checks for all external data sources**
Description	Data from external sources needs to be checked on correctness and completeness in order to verify its quality. Errors in data could introduce development issues or could lead to inaccurate ML models.
Purpose	Prevent the processing of incorrect or incomplete data.
Effect	-
Assessment criteria	• Columns are selected explicitly and their data types are checked (Serban et al., 2023f; Zhang et al., 2022). • The data is checked for missing values (Jain et al., 2020; Serban et al., 2023f). • Correlations between columns (i.e. features) are checked. • The data is checked for class imbalance (Jain et al., 2020). • The data is checked for noisy labels (e.g. incorrect labelled instances; Jain et al., 2020). • The data is checked for duplicates (Jain et al., 2020)
Related practices	DAT1.1, DAT3.4

DAT3.4	**Check that input data is complete, balanced and well distributed**
Description	The (input) data for ML models continuously evolves. Therefore, the distributions, the completeness and the balance of the data need to be checked continuously in order to prevent issues from its evolution.
Purpose	Prevent the processing of incorrect or incomplete data.
Effect	-
Assessment criteria (Serban et al., 2023b)	• The cardinality of features are continuously checked (i.e. whether features still have the right number of unique values). • The distribution of the (input) data is continuously checked if it is shifted (prevents under- and overrepresentations). • Data dependencies (e.g. dependencies between features) are all known (i.e. no hidden dependencies). • A dashboard (or an another visualisation approach) is used to monitor the quality of the data. • An alert system is present to inform the team about unusual events.
Related practices	DAT3.1, DAT3.3, DEPLOY2.4

Table D.1: The practice cards. The practice cards provide detailed information on practices, such as their purposes and perceived effects. Furthermore, the practice cards contain assessment criteria. The practice cards aims to reduce the subjectivity of assessments by assessment criteria, which assist ML teams in determining their degree of adoption of the practices. If all assessment criteria of a particular practice are met, the practice should be considered to be completely adopted. In case none of the defined criteria are satisfied, that practice should be considered to be not adopted at all. (Continued)

TRAIN3.1	**Assign an owner to each feature and document its rationale**
Description	In case a ML-based software system incorporates many data attributes (i.e. features), it could be hard to understand all features and have a good overview of them. Each feature should, therefore, be assigned to a team member, which serves as the owner of that feature, and be documented.
Purpose	Easier comprehension of the created features which in turn improves their maintainablity.
Effect	Quality
Assessment criteria	• Each feature is assigned to an owner (Serban et al., 2023a). • For each feature, a detailed description, its origin and expected benefit (i.e. rationale) are documented (Serban et al., 2023a; Zinkevich, 2021). • In case a feature owner leaves, information and feature ownership is transferred to another member of the team (Serban et al., 2023a; Zinkevich, 2021).
Related practices	TRAIN3.2

CODE3.1	**Use continuous integration**
Description	In order to detect potential issues early due to changes in code, code changes can be integrated frequently into a shared repository and verified with automated software builds and tests.
Purpose	Detect potential issues (e.g. bugs, errors, code quality issues) due to code changes as early as possible.
Effect	Agility and quality
Assessment criteria	• Code quality tests are implemented into the continuous integration (CI) pipeline (Serban et al., 2023e). • Software security tests are implemented into the CI pipeline (JetBrains, n.d.). • Automated regression tests are implemented into the CI pipeline (Serban et al., 2023e). • A branching strategy is defined and shared across the team (Shahin et al., 2017). • Building and test time are kept as small as possible, for example by decomposing large changes into smaller ones, making commits as early and frequently as possible (e.g. once per day) or prioritising tests (JetBrains, n.d.; Shahin et al., 2017). • Build failures are fixed as early as possible and are considered to be the responsibility of the entire team (JetBrains, n.d.). • Team members are aware of the state of the system and made code changes, for example by close communication and collaboration among team members (JetBrains, n.d.; Shahin et al., 2017).
Related practices	CODE1.1, CODE2.1

Table D.1: The practice cards. The practice cards provide detailed information on practices, such as their purposes and perceived effects. Furthermore, the practice cards contain assessment criteria. The practice cards aims to reduce the subjectivity of assessments by assessment criteria, which assist ML teams in determining their degree of adoption of the practices. If all assessment criteria of a particular practice are met, the practice should be considered to be completely adopted. In case none of the defined criteria are satisfied, that practice should be considered to be not adopted at all. (Continued)

DEPLOY3.1	**Provide audit trails**
Description	As the usage of ML could have impact on social and ethical aspects, there are calls for regulation and auditing ML-based software systems. In order to make ML model behaviour traceable, auditable and regulatable, teams should provide audit trails. An audit trail is a collection of records related to the development process and the behaviour of the model.
Purpose	Enable and support audits of developed software systems by making the behaviour of ML models traceable.
Effect	-
Assessment criteria	• A strategy for auditing is defined and implemented in the development process (e.g. a sequence of steps for documentation are defined; Serban et al., 2023d). • Audit trails should cover each stage of the development process (trails about the data and the model are included as well; About ML, 2021; Serban et al., 2023d). • It is defined and documented how a system could be used and misused and whom could be potentially impacted (About ML, 2021). • Some audit reports are generated automatically (Serban et al., 2023d). • Audit trails are shared internally and externally (About ML, 2021).
Related practices	DEPLOY2.5, TRAIN3.5, GOVERN3.2

TEAM2.2	**Work against a shared backlog**
Description	The backlog is a ordered list of several tasks (i.e. work items) related to the project and the system (e.g. features and bugs). Although the entire team takes part in creating the backlog, the backlog is maintained by the product owner. In case a team wants to work on particular tasks, these tasks are removed from the backlog and put on a planning board. The backlog needs to be shared within the team and with external stakeholders.
Purpose	Support communication about work items and their content, priority and status within the team and with external stakeholders.
Effect	Effectiveness and traceability
Assessment criteria	• The creation of the backlog is considered a team effort (B. Overeem, 2014). • The backlog meets the DEEP (i.e. detailed, emergent, estimated and prioritised) acronmy (B. Overeem, 2014; Visual Paradigm, n.d.). • The workitems on the backlog are ordered based on more aspects than priority and value (e.g. risk and dependency; B. Overeem, 2014). • The backlog is kept manageable as only work items are added that will be executed (B. Overeem, 2014). • Backlog management is supported by a tool. (e.g. Microsoft planner and Teamwork; Aston, 2022; B. Overeem, 2014; Serban et al., 2023g). • The backlog is shared within with external stakeholders (B. Overeem, 2014; Serban et al., 2023g). • The backlog is easy accessible for each team member (B. Overeem, 2014).
Related practices	TEAM2.1

Table D.1: The practice cards. The practice cards provide detailed information on practices, such as their purposes and perceived effects. Furthermore, the practice cards contain assessment criteria. The practice cards aims to reduce the subjectivity of assessments by assessment criteria, which assist ML teams in determining their degree of adoption of the practices. If all assessment criteria of a particular practice are met, the practice should be considered to be completely adopted. In case none of the defined criteria are satisfied, that practice should be considered to be not adopted at all. (Continued)

GOVERN1.1	**Establish responsible AI values**
Description	In order to prevent negative impact of the use of ML and ensure responsible use, the team and external stakeholders should establish and all share the same values regarding responsible use of ML regarding aspects as privacy, fairness and security.
Purpose	Operate according to the same shared values regarding responsbile AI.
Effect	
Assessment criteria (Serban et al., 2023c)	• The team and all external stakeholders adhere to a particular framework for responsible AI (i.e. code of conduct). • The responsible AI framework and corresponding objectives are suitable for/tailored to the situation of team or the entire organisation.
Related practices	GOVERN1.2

Appendix E

The interview guide

Introduction

First of all, thank you for making some time for this interview. My name is Ceyhan Deve and I am a master's student ICT in Business and the Public Sector at the Leiden university. During this interview, I would like to get your opinion and feedback on the initial version of the maturity model that I created for my master's thesis. The created maturity model aims to support and guide ML teams in improving their development processes.

This interview consists of two main parts. The first part is about the application of the maturity model, while the second part focuses on the content of the model. At the end of this interview, I would like you to fill in a short questionnaire in which you are asked to give your final opinion about the model by rating the model on a set of criteria.

Your answers during this interview and questionnaire will be used to suggest refinements for the maturity model. Your answers will be anonymised and can not be traced back to you. In order to be able to analyse the answers in detail after the interview, I would like to record the interview. Is that okay with you? The recording will not be shared with anyone and will be deleted after the research is finished. Do you have any questions until now?

General

1. Can you tell me something about the organisation and the department you are in?
2. What are your role(s) within the team?
3. How large is your team?
4. How much experience does your team have in the development of ML-based software systems?
5. What kind of ML-based software system(s) do you develop as a team?
6. What kind of data does your team use for the development of the software systems?

The application of the maturity model

7. How easy was it to understand the content of the maturity model?
8. Was it easy to asses the maturity of your development process with the use of maturity model?
9. Are there any components of the model which you did not use during the assessment?
10. Do you think that your obtained assessment results represent the maturity of your development process adequately?
11. How useful do you find the maturity model for the assessment and improve of your development process?
12. Are you going to use the obtained insights into your process during the assessment to improve your development process?

The content of the maturity model

13. Do you think the practice domains covers all relevant aspects of the development process of ML-based software systems?
14. Are you familiar with all practices within the maturity model?
15. Do you perceive the distinction between the domain and maturity representation within the maturity model as useful?
16. Within the domain representation, do you think the practices per domain are logically and correctly assigned to the defined levels?
17. Do you find that the defined maturity levels within the maturity representation describe a logical progression of the maturity of development process of ML-based software systems?
18. Do you think each maturity level within the maturity representation is logically and correctly composed of practice domain levels?
19. Do you think the labels and the descriptions of the defined levels within the model represent the content of each level (i.e. practices within each level) correctly?
20. Are there any elements that you miss in the maturity model?

These were all my questions. Do you have any other suggestions or remarks?

In order to conclude this case study, would (one of) you please fill in the questionnaire?

Thank you for participating!

Appendix F

The questionnaire

Q1. On a scale of 1-to-5, how complete do you think the maturity model is? 1 being very incomplete and 5 being very complete.

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Q2. Could you motivate your answer to the previous question?

Q3. On a scale of 1-to-5, how easy was it to understand the maturity model? 1 being very difficult and 5 being very easy.

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Q4. Could you motivate your answer to the previous question?

Q5. On a scale of 1-to-5, how easy was it to use the maturity model? 1 being very difficult and 5 being very easy.

○ 1

○ 2

○ 3

○ 4

○ 5

Q6. Could you motivate your answer to the previous question?

Q7. On a scale of 1-to-5, how correct do you think the content of maturity model is? 1 being very incorrect and 5 being very correct.

○ 1

○ 2

○ 3

○ 4

○ 5

Q8. Could you motivate your answer to the previous question?

Q9. On a scale of 1-to-5, how useful do you think the maturity model is for the assessment and improvement of the development processes of ML teams? 1 being very useless and 5 being very useful.

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Q10. Could you motivate your answer to the previous question?

Q11. On a scale of 1-to-5, how likely do you think is it that ML teams would use the maturity model in practice for the assessment and improvement of their development processes? 1 being very unlikely and 5 being very likely.

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

Q12. Could you motivate your answer to the previous question?