# The machine learning maturity model for teams

## Framework report

**The machine learning maturity model for teams**
**Framwork report**

Created by Ceyhan Deve,
Master's student ICT in Business and The Public Sector at
Leiden Institute of Advanced Computer Science (LIACS), a
department of The Leiden University

This report serves as the documentation for the machine learning maturity model for teams. The machine learning maturity model for teams is created as part of a master's thesis in the programme ICT in Business and the Public Sector at the Leiden University.

# Contents

# Introduction

## The model

The machine learning maturity model for teams is a prescriptive model for the assessment of the development processes of machine learning (ML) teams, teams that develop ML-based software systems. The maturity model aims to support and guide teams in the assessment and improvement of their development processes.

The maturity model is created based on a set of 45 best practices for the engineering of ML-based software systems. Most of these practices (i.e. 29 of the 45 practices) are identified during a study conducted by Serban et al. (2020), aimed at determining how ML teams develop, deploy and maintain ML-based software systems. In this study, Serban et al.(2020) identified these practices based on a review of grey and academic literature, after which a survey among ML practitioners were conducted in order to determine the adoption of the practices and validate their perceived effects.

The rest of the practices are identified in a follow-up study by Serban et al. (2021), which build upon their previous study. This follow-up study focuses on bridging the gap between guidelines for the development of trustworthy ML and actionable practices for ML practitioners. Serban et al. (2021), therefore, identified practices for developing trustworthy ML based on a review of white and grey literature. As their previous work, the researchers conducted a survey in order to measure the adoption of the defined practices. This survey is an extension of the conducted survey in the initial study with questions related to the defined trustworthy ML practices. The extended survey measures, therefore, both the adoption of the practices of the initial study and the follow-up study.

We used academic literature and data about the adoption and perceived effects of the practices collected with the surveys of Serban et al. (2020, 2021) to create the maturity model.

In order to provide ML teams with a more comprehensive assessment of their development processes, the maturity model has two representations: the domain representation and the maturity representation. The domain representation focuses on the maturity of ML teams on a domain level represented by engineering capability levels.

The maturity representation, which builds upon the domain representation, focuses on the overall maturity of teams indicated by maturity levels. The two representations enable ML teams to improve their development processes on a domain level or as a whole.

The maturity model consists of the following main components:

- The domain and maturity representations
- The framework application guide
- The practice cards

# The survey data

## The initial data

The initial data is the (merged) responses to the questionnaires (i.e. the initial and extended questionnaire) used during the studies of Serban et al. (2020, 2021). The initial questionnaire consists of three sections: the preliminary questions, the practice adoption questions (i.e. questions on practice adoption) and the effect questions (i.e. questions on perceived effects of practices). The preliminary questions in the questionnaire are mostly related to demographic factors of the respondents, such as team size and experience. The practice adoption questions measure the degree to which practices are adopted on a 4-point Likert scale ranging from 'Not at all' to 'Completely'. In total, the initial questionnaire contains 45 questions. The extended questionnaire is an extension of the initial survey with 14 practice questions related to the identified practices in the follow-up study (Serban et al, 2021) and one effect question related to trustworthy ML.

The initial data consist in total of 504 responses of which 378 related to the initial questionnaire and 126 related to the extended questionnaire.

## The processing of the data

In order to create a consistent data pool of teams, we processed the initial data. We filtered out instances in the data related to respondents that are not part of a team that uses ML. Besides, we removed all questions that do not occur in both the initial and extended questionnaire in order to ensure that all respondents in the our data pool answered the same questions. Furthermore, we discarded respondents which did not answer all practice adoption questions in the remaining questions.

The created data pool consists of 139 teams that build ML applications, that use ML or of which one member builds a ML application.

With the use of the data pool, we calculated the adoption of the practices by teams in the pool and ranked these practices, among other things, based on their calculated adoption.

Within the framework, two adoption scores are distinguished, which indicate the adoption of practices: the practice adoption and the adoption percentile. The practice adoption of a particular practice indicates the number of teams within the pool that adopted the practice while the adoption percentile of a practice is the percentile rank of its practice adoption. A practice is considered to be adopted when a team adopts that practice 'Mostly' or 'Completely'.

In the remainder of this chapter, we visualise the data pool based on demographic factors.

**Application of machine learning within teams**

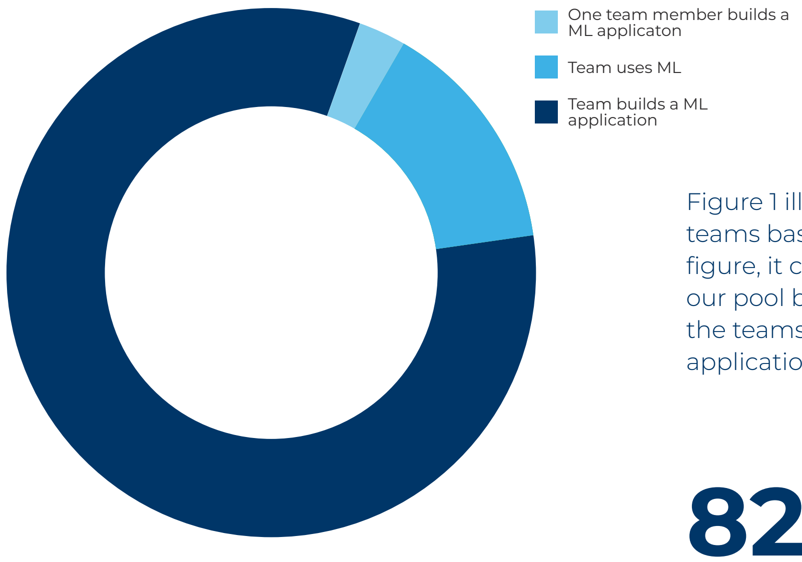


**Figure 1**

Figure 1 illustrates the composition of the data pool of teams based on the application type of ML. From this figure, it can be seen that most of the teams within our pool build a ML application. A small proportion of the teams have only one member who builds the ML application.

**82.7%**

**Of the teams build a ML application**

**43.2%**

## Of the teams work at a tech company

Figure 2 shows the proportions of teams within the data pool per organisation type. Most of the teams work at tech companies and in research (i.e. at a university or non-commercial research labs.

**Organisation type of the ML teams**

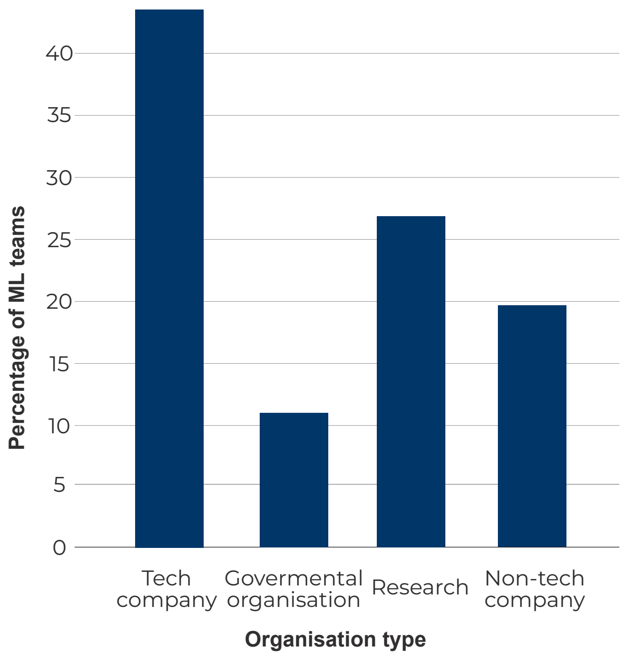


Figure 2

**Geographical locations of ML teams**

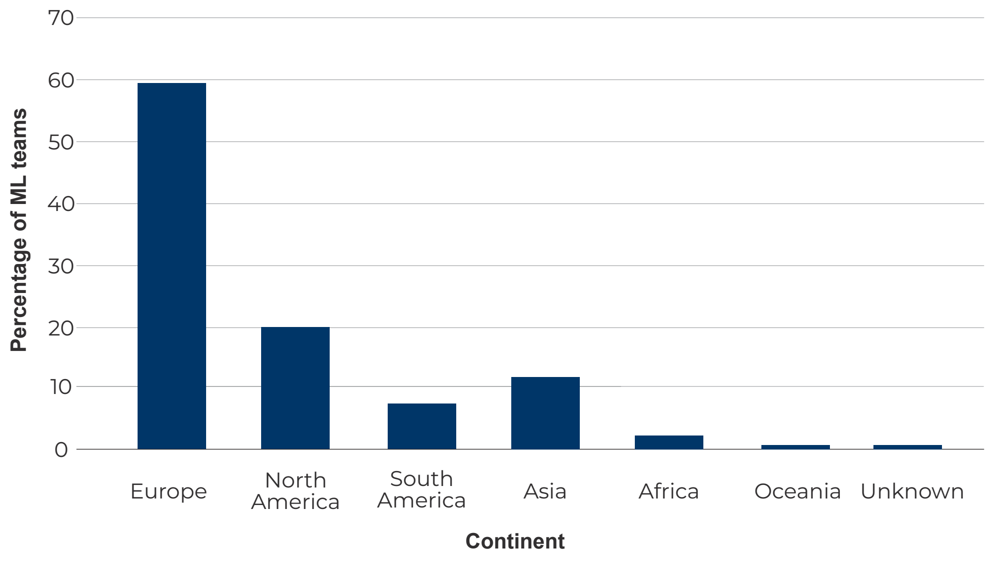


Figure 3

The teams in the data pool are from all over the world. Figure 3 shows the proportions of teams per continent. As evident from this figure, most of the teams are from Europe, North America or Asia. For a small proportion of the teams, the geographical location is unknown.

## Most of the teams are from

**Europe**

Figure 4 shows the proportions of teams per years of experience. With a percentage of 35,3%, a 2 to 5 years experience occur the most under the teams. This is almost equal to a 1 to 2 year(s) of experience with 30,2%.

**Experience of the ML teams**

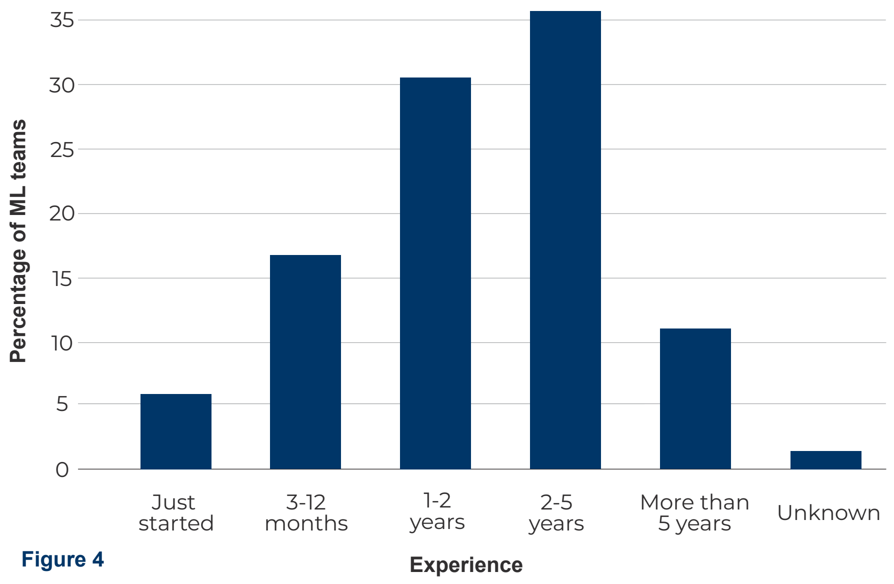


**Figure 4**

## Most of the teams have

# 2 to 5 years experience

**Size of the ML teams**

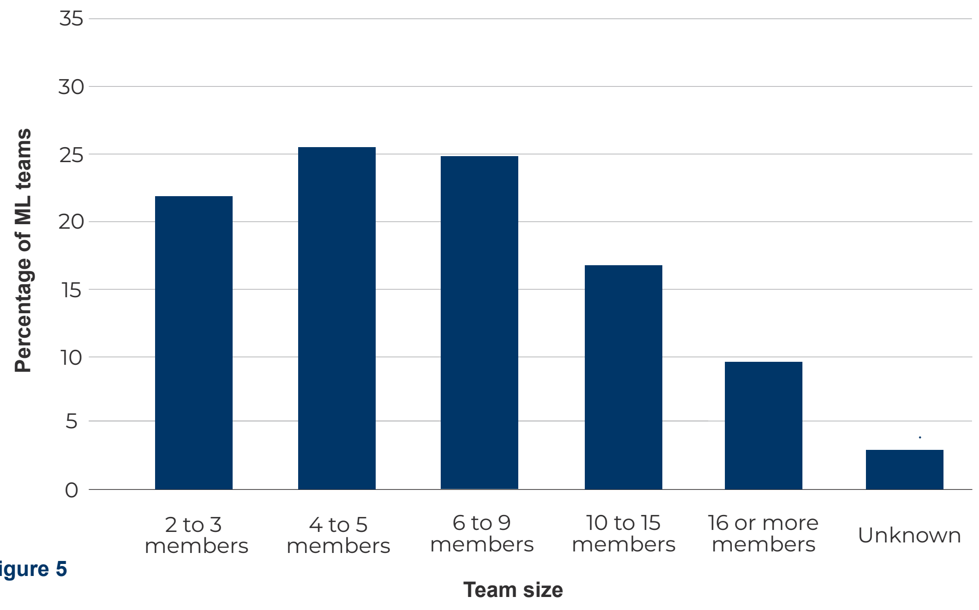


**Figure 5**

Figure 5 shows the proportions of teams per team size category. From this figure, it can be seen that the proportion of teams with a team size of 6 to 9 members are almost equal to the proportion of teams with a team size of 4 to 5 member, which is the largest proportion with 25,2%.

## Most of the teams have

# 4 to 5 members

**Data sources used by ML teams**

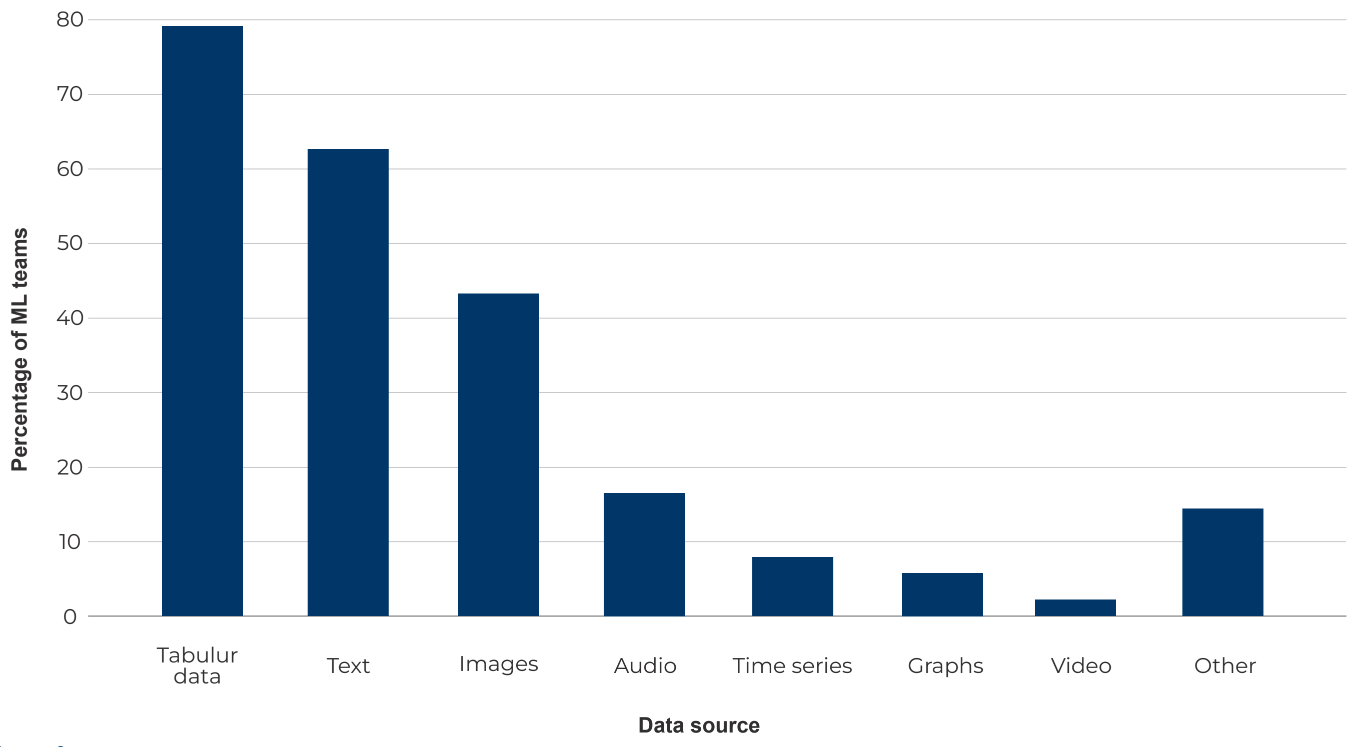


**Figure 6**

Figure 6 illustrates the usage of each data source by the teams in percentages (a team could use one or more data source). The three most used data sources are tabular data, text and images.

A relatively large proportion of the teams use other data sources than the specified data sources, such as sensor data.

# The framework

## The practice domains

The maturity model covers six domains of the ML development process: data, training, coding, deployment, team and governance. Each domain consists of a set of practices for the engineering of ML-based software systems.

The domains with their corresponding descriptions are shown below.

### 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 maintanance 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 domain representation

The domain representation of the maturity model represents the division of practices on a domain level. The practices of each domain are organised across three different levels, which are referred to as the capability engineering (EC) levels. In addition, we define an initial level (i.e. EC level 0), which does not contain any practices. The maturity of teams regarding the domains starts at this level. We eleborate on the EC levels on page 13.

| 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 | - | - |

| 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 |
| 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

| Practices | 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 | - | - |

## DEPLOYMENT

| Practices | 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 | 50,0 |
| 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 |
| **Engineering capability level  3 - Advanced** | | | |
| Provide audit trails. | DEPLOY3.1 | - | - |

## TEAM

| Practices | 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 | - | - |

**Figure 7: The domain representation.** *The practices of each domain are organised across three EC levels. EC level 0 does not consists of practices and is not shown in this figure.*

## The engineering capability levels

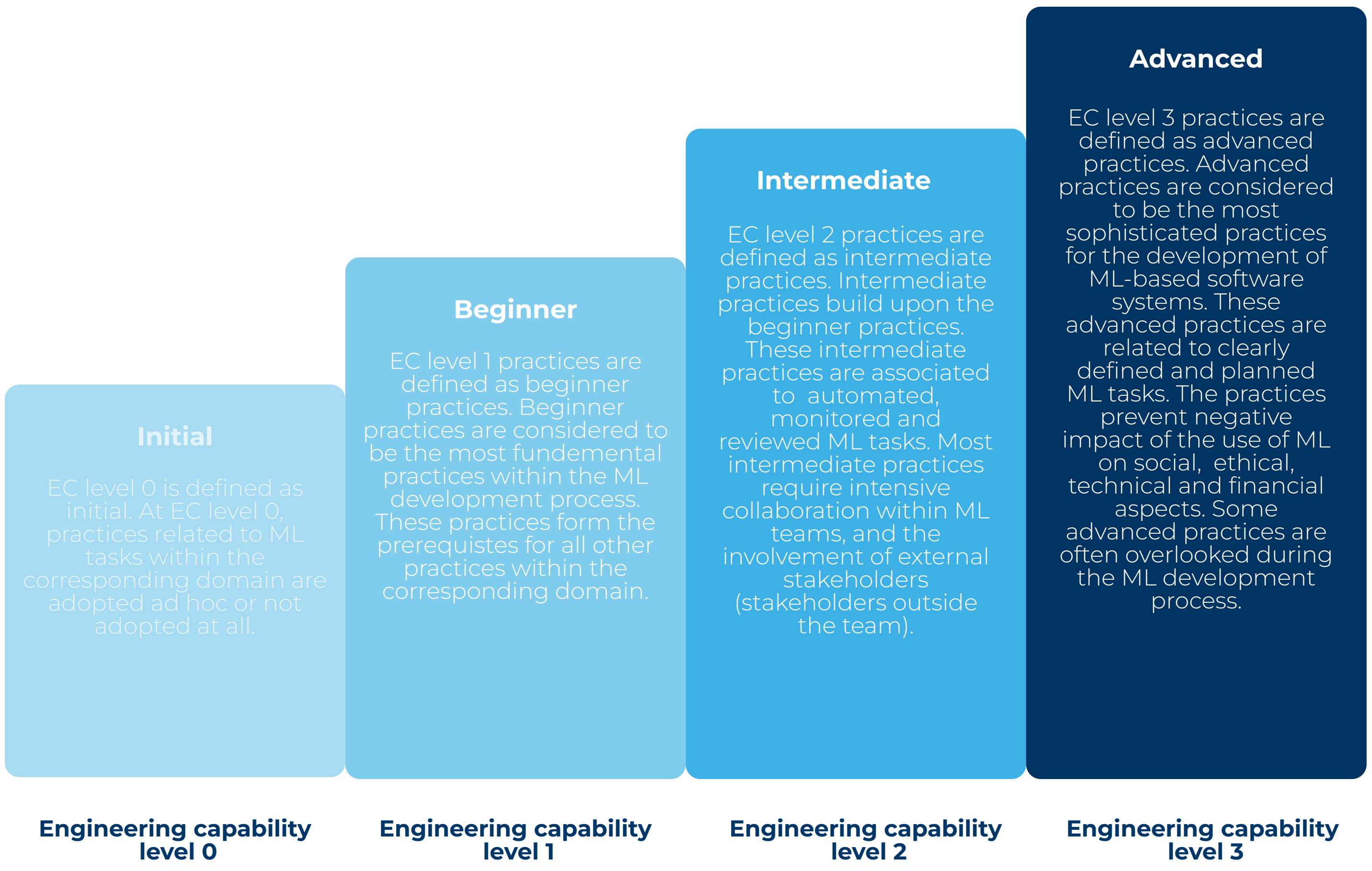
The EC levels indicate the maturity of ML teams regarding the six practice domains, which are shown on page 9. Within the framework, the following four EC levels are distinguished:

0. Initial
1. Beginner
2. Intermediate
3. Advanced

The maturity of a team regarding the domains starts at initial (i.e. level 0). For a ML team to achieve a particular level for a domain, a ML team needs to adopt all practices of that level of the corresponding domain that are applicable to their process  and all applicable practices of  any lower levels of that domain.

For example, a ML team achieves EC level 2 for the ‘Data’ domain when the team adopts all applicable practices of EC levels 1 and 2 related to ‘data’. Based on the adoption of practices by a ML team, a team achieves one of the three EC levels for each practice domain.

The four EC levels with their corresponding descriptions are represented in Figure 8.



**Initial**

EC level 0 is defined as initial. At EC level 0, practices related to ML tasks within the corresponding domain are adopted ad hoc or not adopted at all.

**Beginner**

EC level 1 practices are defined as beginner practices. Beginner practices are considered to be the most fundemental practices within the ML development process. These practices form the prerequistes for all other practices within the corresponding domain.

**Intermediate**

EC level 2 practices are defined as intermediate practices. Intermediate practices build upon the beginner practices. These intermediate practices are associated to  automated, monitored and reviewed ML tasks. Most intermediate practices require intensive collaboration within ML teams, and the involvement of external stakeholders (stakeholders outside the team).

**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.

Engineering capability level 0
Engineering capability level 1
Engineering capability level 2
Engineering capability level 3

**Figure 8: The engineering capability levels.** *The engineering capability levels indicate the maturity of machine learning teams regarding the practice domains.*

## The maturity representation

The maturity representation of the maturity model represents the division of the practices into maturity levels. These maturity levels consist of EC levels of the practice domains. The maturity levels indicate the overall maturity of the development processes of ML teams. Figure 9 shows the maturity representation.

On page 17, we elaborate on the maturity levels of the maturity representation.

### 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 |
| 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 | 39.3 |
| Peer review training scripts. | TRAIN2.1 | 66 | 46.4 |
| Test all feature extraction code. | TRAIN2.2 | 53 | 25,0 |
| 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 | 14.3 |
| 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 | - | - |

**Figure 9: The maturity representation.** *The practices are organised across maturity levels. Each maturity level indicate a particular overall maturity of the development process of a team. Maturity level 1 does not contain any practices and is thus not shown in this figure.*

## From engineering capability levels to maturity levels

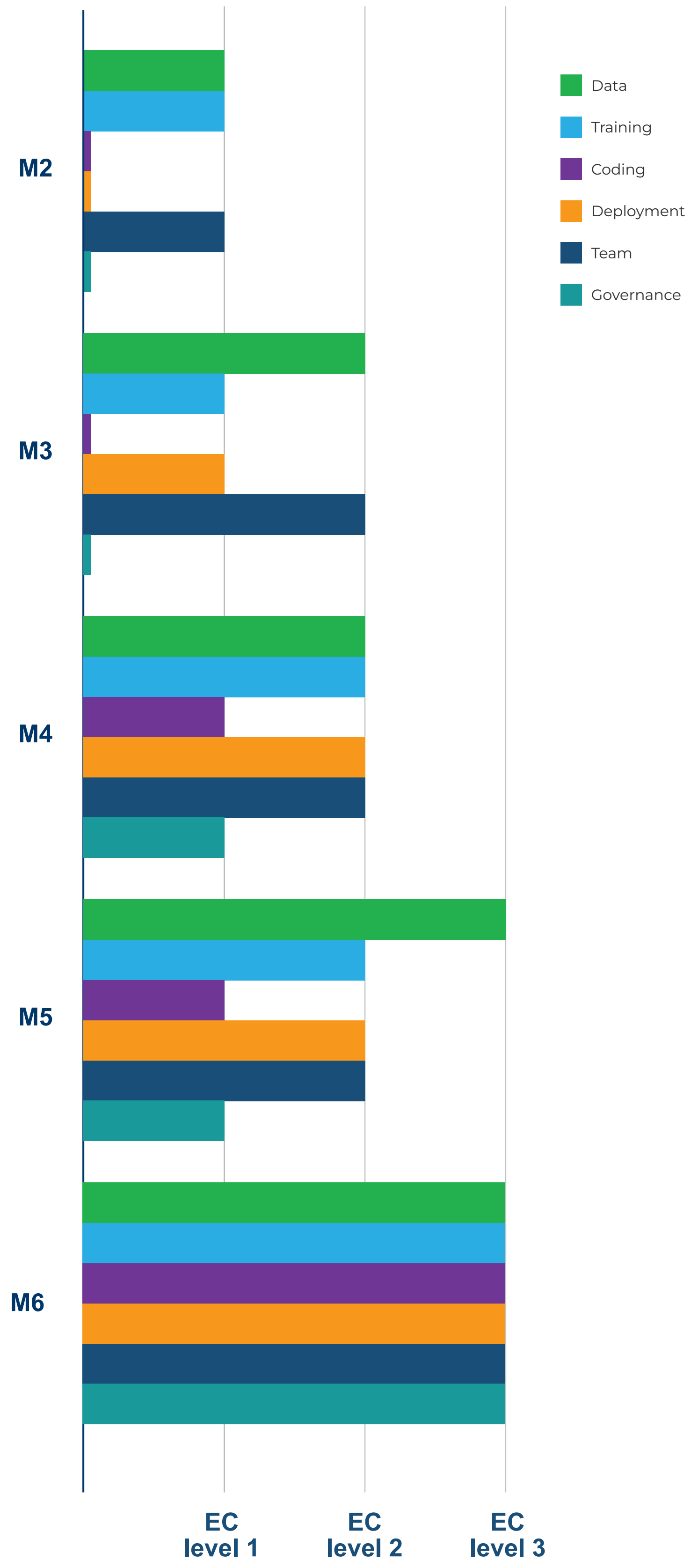


**Figure 10: The maturity levels represented in practice domain levels.** *As the overall maturity of the development process of a ML team increases, the practice domain levels changes.*

As described on page 14, maturity levels are made of EC levels of practice domains. An EC level of a particular domain is referred to as a practice domain level (e.g. team EC level 1). Each maturity level consists thus of a particular set of practice domain levels. Figure 10 illustrates for each maturity level which practice domain levels should be achieved. For example, a team should achieve EC level 1 for the data, training and team domain in order to have a maturity level of 2.

From Figure 10, it can be seen how the practice domain levels evolve when the overall maturity level of a team increases. EC levels of the practice domains could thus be converted to maturity levels or vice versa.

## The maturity levels

The maturity levels, which comprise practice domain levels, describe particular overall maturity states of the development process of a ML team. Within the maturity representation of the maturity model, six maturity levels are defined which range from 1 to 6:

1. Initial
2. Basic
3. Progressed
4. Optimised
5. Proactively optimised
6. Perfected

The maturity of development processes of ML teams begins at level 1 (initial). As with the domain representation, a team should adopt all practices of a particular maturity level that are applicable to their development process and all applicable practices of any lower levels in order to achieve that level of maturity for their development process.

The maturity levels are described below.

### 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 its 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 of 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. A 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 assure a responsible built and qualitative system. Hence, maturity level 5 encompasses practices related to (data) error, bug, cyberattack 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. coding and data) within the ML workflow are considered to be fully developed. A perfected development process contains practices related to defined and reviewing tasks.

# The assessment of development processes

## The assessment

The maturity model is a self-assessment framework. For an assessment of a development process, the domain representation, the maturity representation or both representations could be used. In order to assess a development process, ML teams should determine to which extent each practice is adopted. We use the same Likert scale as in the questionnaires of Serban et al (2020, 2021) to reflect the degrees of adoption. Within the framework, the following four adoption degrees are thus distinguished: not at all, partially, mostly and completely. As some practices are not applicable to all development processes, teams for which these practices are not applicable, do not have to take these particular practices into account during an assessment. These practices are DAT2.1, DAT3.1, DAT3.2, DAT3.3, DAT3.5, TRAIN2.2, TRAIN3.1, TRAIN3.2, TRAIN3.3, TRAIN3.4 and GOVERN1.2.

As stated before, a particular level within the selected representation(s) is accomplished when a team adopts all applicable practices of that level and any lower levels. A practice is considered to be adopted when that practice is 'mostly' or 'completely' adopted within the assessed development process.

In order to provide teams with guidance during the assessments, the created model consists of the framework application guide, which describes the process of applying the framework, and some practice cards, which provide more details on particular practices.

For the assessment of development processes, teams could use the interactive a workbook, which supports the assessment process. The workbook provide teams with a straightforward way to assess a process (i.e. assign adoption degrees to practices), obtain their assessment results and plan the improvement of the process.

## The framework application guide

The framework application guide defines the use of the framework as a process of five consecutive steps: plan and prepare assessment, conduct the assessment, set maturity target(s), plan process improvement and execute process improvement plan. Each step consists of several components, which are based on the components of the quick start guide of SAMM (OWASP, n.d.), a prescriptive maturity model for the assessment and improvement of the software security posture of organisations. These components are: the purpose of the process step, activities that needs to be executed during a step, useful resources that could be used and recommendations on the execution of the step (i.e. practices).

The framework application guide could be found in appendix A.

## The practice cards

For some practices, we created practice cards, which could support the assessment of development process. A practice card describes a practice based on several components: a description of the practice, its purpose, its perceived effect on the development process, assessment criteria and related practices.

The defined assessment criteria on practice cards provide guidance in determining the extent to which a practice is adopted. For a practice to be completely adopted, at least all defined assessment criteria should be met. In case none of the criteria are met, a practice is considered to be not adopted at all. With the definition of assessment criteria on the practice cards, we thus define the extreme ends of the Likert scale for the corresponding practices.

The practices cards are presented in appendix B.

# References

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

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

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* [Conference session]. KDD 2020, Virtual event, United States. 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/

OWASP (n.d.). *Software Assurance Maturity Model.* https://owaspsamm.org/

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

Serban, A., van der Blom, K., Hoos, H., & Visser, J. (n.d.-a). *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. (n.d.-b). *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. (n.d.-c). *Establish responsible AI values.* https://se-ml.github.io/best_practices/06-code_conduct/

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

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

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

Serban, A., van der Blom, K., Hoos, H., & Visser, J. (2020). Adoption and effects of software engineering best practices in machine learning. Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 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

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

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

Zhang, H., Cruz, L., & van Deursen, A. (2022). *Code smells for machine learning applications*. arXiv. https://doi.org/10.48550/arXiv.2203.13746

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

## Attributions

# Appendices

## A The framework application guide

### 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 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).<br><br>**1.2 Determine assessment objective(s)** – Determine as a team the objective(s) to be achieved as a result of the assessment<br><br>**1.3 Set assessment date** – Plan as a team the date on which the assessment will be conducted.<br><br>**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.<br>• Ensure that maturity assessment framework is understood within the team.<br>• 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.<br><br>**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.<br><br>**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 assessment sheets are used for the assessment, the obtained maturity level(s) is/are automatically generated and shown on the 'Assessment report' sheet. |
| **Resources** | • Interactive workbook<br>• Practice cards<br>• Development process artifacts (e.g. source code and notes) |
| **Practices** | • Ensure that all practices are understood in detail.<br>• Assign adoption degrees to practices based on development artifacts as much as possible.<br>• Reach consensus within the team on adoption degrees of practices. |

## 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 maturity level(s)** – Determine which maturity level(s) the team desires to achieve. In case the domain representation is used, the desired EC 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 EC levels, the overall maturity level or a combination of both can be defined.<br>**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.<br>**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.<br>**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.<br>**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.<br>• Balance the number of practices over the phases.<br>• Focus first on practices of lower levels and then on higher level practices.<br>• Take dependencies between practices into account.<br>• 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** | - |

## B The practice cards

| **DAT3.1** | **Use sanity checks for all external data sources** |
|---|---|
| Description | Data from external sources needs to be checked on correctness and completness 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., n.d.-f; Zhang et al., 2022).<br>· The data is checked for missing values (Jain et al., 2020; Serban et al., n.d.-f).<br>· Correlations between columns (i.e. features) are checked.<br>· The data is checked for class imbalance (Jain et al., 2020).<br>· The data is checked for noisy labels (e.g. incorrect labelled instances; Jain et al., 2020).<br>· 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 continously evolves. Therefore, the distributions, the completness and the balance of the data need to be checked continously in order to prevent issues from its evolution. |
| Purpose | Prevent the processing of incorrect or incomplete data. |
| Effect | - |
| Assessment criteria (Serban et al., n..d.-b). | · The cardinality of features are continously checked (i.e. whether features still have the right number of unique values).<br>· The distribution of the (input) data is continously checked if it is shifted (prevents under- and overrepresentations)<br>· Data dependencies (e.g. dependencies between features) are all known (i.e. no hidden dependencies).<br>· A dashboard (or an another visualisation approach) is used to monitor the quality of the data.<br>· An alert system is present to inform the team about unusual events. |
| Related practices | DAT3.1, DAT3.3, DEPLOY2.4 |

| **TRAIN3.1** | **Assign an owner to each feature and document its rationale** |
|---|---|
| Description | In case of large ML-based software systems with many data attributes (i.e. features), it could be hard to understand these features and have a good overview of them. Each feature should be assigned to a team member, which serves as the owner of that feature, and be documented. |
| Purpose | Easier comprehension of the developed features which in turn improves their maintainablity. |
| Effect | Quality |
| Assessment criteria | · Each feature is assigned to an owner (Serban et al., n.d-a).<br>· For each feature, a detailed description, its origin and expected benefit (i.e. rationale) are documented (Serban et al., n.d-a; Zinkevich, 2021).<br>· In case a feature owner leaves, information and feature ownership is transfered to another member of the team (Serban et al., n.d-a; 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 frequenty 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 continous integration (CI) pipeline (Serban et al., n..d.-e).<br>· Software security tests are implemented into the CI pipeline (JetBrains, n.d.).<br>· Automated regression tests are implemented into the CI pipeline (Serban et al., n..d.-e).<br>· A branching strategy is defined and shared across the team (Shahin et al., 2017).<br>· Building and test time are kept as small as possible, for example by decomposing large changes into smaller ones, making commits as early and frequenty as possible (e.g. once per day) or prioritizing tests (JetBrains, n.d; Shahin et al., 2017).<br>· Build failures are fixed as early as possible and are considered to be the responsibility of the entire team (JetBrains, n.d.).<br>· 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 |

| **DEPLOY3.1** | **Provide audit trails** |
|---|---|
| Description | As the usage of ML could have impact on social and ethical aspects, there are calls for regulating 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., n.d.-d).<br>· 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., n.d.-d).<br>· It is defined and documented how a system could be used and misused and whom could be potentially impacted (About ML, 2021).<br>· Some audit reports are generated automatically (Serban et al., n.d.-d).<br>· 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 an 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 (Overeem, 2014).<br>· The backlog meets the DEEP (i.e. detailed, emergent, estimated and prioritised) Acronmy (Overeem, 2014; Visual Paradigm, n.d.).<br>· The workitems on the backlog are ordered based on more aspects than priority and value (e.g. risk and dependency; Overeem, 2014).<br>· The backlog is kept managable as only work items are added that will be executed (Overeem, 2014).<br>· Backlog managment is supported by a tool (e.g. Microsoft planner and Teamwork; Overeem, 2014; Serban et al., n.d.-g; Aston, 2022).<br>· The backlog is shared with external stakeholders (Overeem, 2014; Serban et al., n.d.-g)<br>· The backlog is easy accessible for each team member (Overeem, 2014). |
| Related practices | TEAM2.1 |

| **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., n.d.-c) | · The team and all external stakeholders adhere to a particular framework for responsible AI (i.e. code of conduct).<br>· The responsible AI framework and corresponding objectives are suitable for/tailored to the situation of the team or the entire organisation. |
| Related practices | GOVERN1.2 |