Software Quality Journal (2022) 30:307–335
https://doi.org/10.1007/s11219-021-09557-y

Construction of a quality model for machine learning systems

Julien Siebert[1] · Lisa Joeckel[1] · Jens Heidrich[1] · Adam Trendowicz[1] · Koji Nakamichi[2] · Kyoko Ohashi[2] · Isao Namba[2] · Rieko Yamamoto[2] · Mikio Aoyama[3]

Accepted: 14 April 2021 / Published online: 25 June 2021

Abstract
Nowadays, systems containing components based on machine learning (ML) methods are becoming more widespread. In order to ensure the intended behavior of a software system, there are standards that define necessary qualities of the system and its components (such as ISO/IEC 25010). Due to the different nature of ML, we have to re-interpret existing qualities for ML systems or add new ones (such as trustworthiness). We have to be very precise about which quality property is relevant for which entity of interest (such as completeness of training data or correctness of trained model), and how to objectively evaluate adherence to quality requirements. In this article, we present how to systematically construct quality models for ML systems based on an industrial use case. This quality model enables practitioners to specify and assess qualities for ML systems objectively. In addition to the overall construction process described, the main outcomes include a meta-model for specifying quality models for ML systems, reference elements regarding relevant views, entities, quality properties, and measures for ML systems based on existing research, an example instantiation of a quality model for a concrete industrial use case, and lessons learned from applying the construction process. We found that it is crucial to follow a systematic process in order to come up with measurable quality properties that can be evaluated in practice. In the future, we want to learn how the term quality differs between different types of ML systems and come up with reference quality models for evaluating qualities of ML systems.

Keywords Machine learning · Quality requirements · Software quality · Quality evaluation · Quality model

1 Introduction

The digital transformation enables digital products and services that are based on data or on models derived from data. This enables innovative solutions such as automated translation, automated driving, or predictive maintenance. At the core of such products lie data-driven software components. A data-driven software component is a piece of software that

✉ Julien Siebert
julien.siebert@iese.fraunhofer.de

Extended author information available on the last page of the article

solves a given task (e.g., image segmentation, sentiment analysis, classification, etc.), using methods from data science, such as machine learning (ML), data mining, natural language processing, signal processing, statistics, etc. The functionality of data-driven software components (or at least part of it) is not entirely defined by the programmer in the classical way (by programming it directly), but is derived (i.e., learned) from data.

At the core of a data-driven software component lies the notion of a model (or several models, coupled together in a pipeline). Developing and operating data-driven software components raises new challenges in comparison to "classical" software engineering (Arpteg et al., 2018; de Souza Nascimento et al., 2019; Kästner & Kang, 2020; Kumeno, 2020; Lwakatare et al., 2020; Sculley et al., 2015; Wan et al., 2019; Zhang & Tsai, 2002; Zhang et al., 2019). In short, the behavior of such components is first and foremost fundamentally different from traditional (i.e., not data-driven) software: the relationship between the input and the outcome of the software component is usually non-linear. This means that a small change in the inputs can lead to strong discrepancies in the outputs (see, for example, the problem of adversarial examples (Kurakin et al., 2016)). This input-output relationship is also only defined for a subset of the data, which leads to uncertainty in outcomes for previously unseen data (or when the modeled system changes, i.e., if there is concept drift). Different algorithms can be used to solve a given task (for example, neural networks, support vector machines, and decision trees can all solve classification tasks). The knowledge of the algorithm(s) used to build a data-driven software component alone is not sufficient to understand how the component will behave. First, it is necessary to also consider the data used at both training time and inference time (i.e., runtime). Second, common development principles from software engineering, such as encapsulation and modularity, have to be rethought. For example, changing the application context of a data-driven component (its intended scope), the type of model used, or the internal parameters, usually implies retraining the component. This is also referred to as the CACE principle: changing anything changes everything (Sculley et al., 2015). Third, the development and integration of data-driven software components are a multi-disciplinary approach: it requires knowledge about the application domain, knowledge about how to construct models, and finally, knowledge about software engineering. Fourth, quality assurance, and specifically testing, works differently than in traditional software. This is because data-driven methods (such as ML, for instance) target problems where the expected solution is inherently difficult to formalize, and where test oracles are not directly available (Belani et al., 2019; Bosch et al., 2018; Horkoff, 2019; Zhang et al., 2020).

In order to ensure the intended quality of a software system, there are standards that define necessary quality properties of the system and its components. For instance, ISO/IEC 25010 (ISO/IEC, 2011) defines quality models for software and systems, i.e., a hierarchy of quality properties of interest and how to quantify and assess them. Due to the different nature of data-driven software components, these quality models cannot be applied directly as they are. Some have to be adjusted in their definition (e.g., reusability of trained models) and some need to be added (e.g., trustworthiness, fairness). We also have to be very precise about which quality is relevant for which part of the overall system. For instance, in a data-driven software system, the algorithms executing the model play a far less significant role than the type of the model, or the data used for training and testing (Sculley et al., 2015). To develop meaningful quality models, it is necessary to understand the application context of the use case and what kind of data-driven method is used.

In this article, we focus on ML systems and present how to systematically construct quality models for those systems based on an industrial use case. This is an extended version of an article (Siebert et al., 2020) that goes into more details regarding the construction

process itself and the structure of ML quality models. First, we discuss related work and summarize the gaps that we would like to close with our contribution. Second, we give an overview of the proposed process for constructing quality models for ML systems. Each step of the process is detailed in the subsequent sections: In Sect. 4, we describe the structure of the quality model to be constructed. In Sect. 5, we illustrate the industrial use case and application context for which we are developing the quality model. In Sect. 6, we analyze the development process of ML models regarding what qualities could be of potential interest in different development stages. In Sect. 7, we define different views one can take on an ML system and relevant entities, which will have to be evaluated for a specific use case and application context. In Sect. 8, we systematically work out a list of reference entities, relevant quality properties, and how to potentially measure them for a concrete industrial use case. In Sect. 9, we show how to instantiate concrete quality models based on the reference list for a concrete example, which will enable practitioners to specify and assess quality requirements for such kinds of ML systems. In Sect. 10, we discuss the usefulness of the identified qualities based on an evaluation performed together with experts from industry. In Sect. 11, we present major lessons learned from following the quality model construction process for ML systems. Finally, we conclude the paper with a brief summary and an outlook on future research.

2 Related work

To build a quality model, it is first necessary to define the usage scenarios (i.e., How will the system be used? By whom? What are the expectations in terms of quality? etc.). This naturally goes hand in hand with the definition of the relevant quality properties and the entities to be measured. It is also necessary to define how to measure these properties and, finally, to decide on the basis of these measurements what to do to improve certain properties of quality.

In the literature, some quite generic quality models for software and systems can be found, such as ISO/IEC 25010 (ISO/IEC, 2011) or ISO/IEC 8000 (ISO/TS, 2011). These standards propose different definitions of properties grouped into several categories with a decomposition structure (e.g., Product Quality is decomposed into eight properties, such as Functional Suitability, which is decomposed into sub-properties, such as Functional Correctness). With the advance and widespread adoption of data science methods, new and more specific quality proposals (such as the EU Ethics Guidelines for Trustworthy (HLEG, 2019), the German DIN SPEC 92, 001 (SPEC 92001-01), or the Japanese QA4AI consortium (Hamada et al., 2020) as well as certification guidelines (Marselis & Shaukat, 2018; Marselis et al., 2017) have emerged.

Some of the new quality properties are rather generic: they cover not only machine learning but also other disciplines from artificial intelligence (AI) or statistics. These include:

- Transparency and accountability (e.g., reproducibility, interpretability and explainability, auditability, minimization, and reporting of negative impact)
- Diversity, non-discrimination, fairness, as well as societal and environmental well-being (e.g., avoidance of unfair bias, accessibility and universal design, stakeholder participation, sustainability, and environmental friendliness)

- Security, safety, data protection (e.g., respect for privacy, quality and integrity of data, access to data, and ability to cope with erroneous, noisy, unknown, and adversarial input data)
- Technical robustness, reliability, dependability (e.g., correctness of output, estimation of model uncertainty, robustness against harmful inputs, errors, or unexpected situations)
- Human agency and oversight, legal and ethical aspects (e.g., possibility of human agency and human oversight, respect for fundamental rights)

Some quality properties are more specific to interactive and embodied AI (like assistants or robots), such as intelligent behavior and personality (Hamada et al., 2020; Marselis & Shaukat, 2018; Marselis et al., 2017). The quality properties are applied to entities. These entities can represent processes, products, impacted users, or external objects.

Although quality properties are being (re)defined for ML components, the literature from the field of requirements engineering shows many challenges when developing and implementing quality models in the context of ML systems (Belani et al., 2019; Bosch et al., 2018; Horkoff, 2019; Ismail et al., 2019; Vogelsang & Borg, 2019). On the one side, data scientists bring new types of requirements (e.g., in terms of data quality). On the other side, other stakeholders (the users of such systems) may have wrong expectations concerning what such a system can or cannot do, and might not be able to determine which quality properties are relevant. In between, requirements engineers also have to understand how different implementation choices may impact the quality of the system (Horkoff, 2019).

Here, it is interesting to take a short detour to the field of modeling and simulation theory (Calder et al., 2018; Edmonds, 2002; Edmonds et al., 2019; Epstein, 2008). Indeed, as already stated, at the core of an ML component lies the notion of a model. A model is by definition a re-presentation (i.e., a simplification) of some part of reality (i.e., the system under study). It is used to answer a given question about that part of reality (i.e., a model has a purpose). To define usage scenarios and elicit requirements, it is necessary to understand the intended purpose of the model. One example of a model purpose can be that the system under study should be described. This is done, for example, by inspecting the characteristics of the data representing the system (e.g., statistical distribution, topological structure). Another purpose, which is very often present in data science, is the making of a prediction. Other examples can be found in the literature in the field of modeling and simulation. These go far beyond the classical 4: describe, predict, understand, control. For example, Epstein lists 17 reasons for building models (among others: predict, explain, guide data collection, train practitioners) (Epstein, 2008); Edmonds et al. detail seven reasons for building models (prediction, explanation, description, theoretical exploration, illustration, analogy, and social learning) (Edmonds et al., 2019). In parallel, research into data science processes shows that different data science scenarios have emerged (Martinez-Plumed et al., 2020). We see here that ML-based services bring new usage and business scenarios. However, our understanding of how quality is impacted by these scenarios is not complete yet.

Process models related to data-driven methods (such as knowledge engineering, data mining, ML, etc.) have been around for decades (Mariscal et al., 2010; Martinez-Plumed et al., 2020). In recent years, more case studies and literature reviews have been conducted to assess the challenges perceived by developers of ML components, as well as their processes and best practices (see, for instance, the review (Lorenzoni et al., 2021)). We see that there is a consensus on the definition of tasks, roles, and how the process for developing and operating ML systems should be organized. However, it is less clear how the

process impacts quality. Implementing quality improvement actions requires a good understanding of the process: which steps are performed, which people/roles are involved, which entities are affected, etc.

ML components usually consist of several sub-components organized in pipelines: e.g., data preparation (e.g., resizing or cropping images, cleaning text), features engineering, training and evaluating the models. Describing the entities composing the systems and organizing these entities into different views helps to refine which quality properties may be relevant and which quality measures can be used. For example, in Zhang et al. (2020), the authors define a set of quality properties, such as correctness (i.e., goodness of fit), robustness, efficiency, etc. They also relate these quality properties to different views/entities: data, learning program, and framework (e.g., Weka, TensorFlow). In (Nakajima, 2018), the authors distinguish between three main qualities, namely service quality, product quality, and platform quality. They also describe different views/entities of the system: the training dataset, the neural network, the hyper parameters, "the inference in vivo" (corresponding to the decision outputted by the component at runtime), and the machine learning platform. DIN-SPEC 92001 also provides a description in terms of views/entities: data, model, platform, and environment. As a last example, the authors in Hamada et al. (2020) provide five main qualities related to views/entities, namely data integrity, model robustness, system quality, process agility, and customer expectation, including a total of 49 quality sub-properties. Several frameworks have been proposed to model quality of ML systems. For example, the MLQ framework (Ishikawa, 2018) proposes to build argumentations for assessing the quality of ML systems using the following concepts: ML algorithm, dataset, ML component, and ML system. The evAIa method (Poth et al., 2020) is based on four steps: a product risk assessment, a questionnaire on the AI approaches used, recommendations for QA methods to mitigate risks, and a transparency report. The questionnaire contains 47 questions covering 17 topics and classified into three main domains (features and data, implementation, infrastructure).

In the literature, we see that a consensus exists around what qualities need to be measured. However, the naming of the quality properties and the naming of the entities (or their classification) has not yet stabilized.

We also see that, because the field of data science is large, the importance of certain quality properties and measures for quantifying them depends on the concrete context and use case, and they have to be addressed in different tasks of the process model used. For instance, the availability of a ground truth is one important factor (see Fig. 1): (a) if the full ground truth exists (as in the case of reinforcement learning, for example), then test oracles exist. Consequently, the quality mainly depends on the test oracle itself, and the quality can be safely measured using the available ground truth. (b) If only a partial ground truth

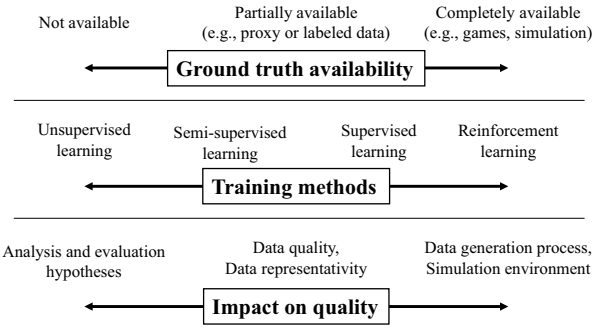


Fig. 1 The availability of ground truth data (labels) has a direct impact on the analysis or training methods used as well as on the definition of quality measures and their assessment

exists (as in the case of semi-supervised or supervised learning), data quality and its representativeness have to be analyzed carefully. (c) If no ground truth exists (as in the case of unsupervised learning), the assumptions made by the learning algorithms and those made during the model evaluation play a significant role with respect to quality. The type of tasks that are performed (such as regression, classification, clustering, outlier detection, dimensionality reduction, etc.) also has an impact on the quality assessment. Each type of task is accompanied by corresponding quality measures. For example, for classification tasks, the goodness of fit can be measured by accuracy, precision, recall, f-score, etc. (Hossin & Sulaiman, 2015), but for clustering, other measures are needed (Emmons et al., 2016). The measures chosen will depend on the use case. For example, in the case of binary classification tasks, the cost of a false-positive may not be the same as the cost of a false-negative. Some measures might not be compatible with one another, as is the case, for example, for fairness measures (Barocas & Boyd, 2017; Kleinberg et al., 2016).

The literature provides a solid basis of relevant quality properties, entities to be measured, process models, etc. However, we see different gaps that have not been addressed so far: (1) there is a lack of unique and clear definitions of views on ML systems (e.g., what is the definition of a platform view in (de Souza Nascimento et al., 2019), or should hyper-parameters be included as a separate view). (2) Existing quality models are often too abstract to be of value for practitioners (e.g., in terms of proposed measures) and require guidelines for tailoring to be applicable (Wagner et al., 2015). (3) The combination of and the relationship between quality properties and related measures have not been sufficiently investigated yet, and it is not clear whether they can be satisfied altogether. (4) Comprehensive development guidelines for quality-aware ML systems, which would bring together the different quality models, processes, and views, are largely missing or not made explicit.

In the remainder of this article, we will contribute mainly to closing the first two gaps. However, our overall research goal is aimed at coming up with comprehensive development guidelines for quality-aware ML systems.

3 Quality model construction process

This section describes the process we followed to construct a quality model for ML systems based on our previous work in the field (Goeb et al., 2015). It consists of six steps, which we will describe sequentially, but which are performed iteratively in practice. An overview of all steps and illustrations of the major outcomes for each step are presented in Fig. 2.

1. Define quality meta-model: First of all, we described the features of our ML quality model; that is, the basic structure we want to use for documenting all quality properties of interest and the measures/metrics for quantifying those properties. This resulted in a quality meta-model as our common understanding for specifying quality models.
2. Define use case and application context: Previous research in the field of quality modeling concluded that the concept of "quality" highly depends on the application context and concrete use case. This includes, e.g., the criticality of the system under development. For this reason, we tried to describe the context and use case as clearly as possible based on a real industrial case. Even though we assume that many of the quality properties derived for this use case in the next steps could be generalized and will also apply to similar systems, the use case and application context give us a good basis for

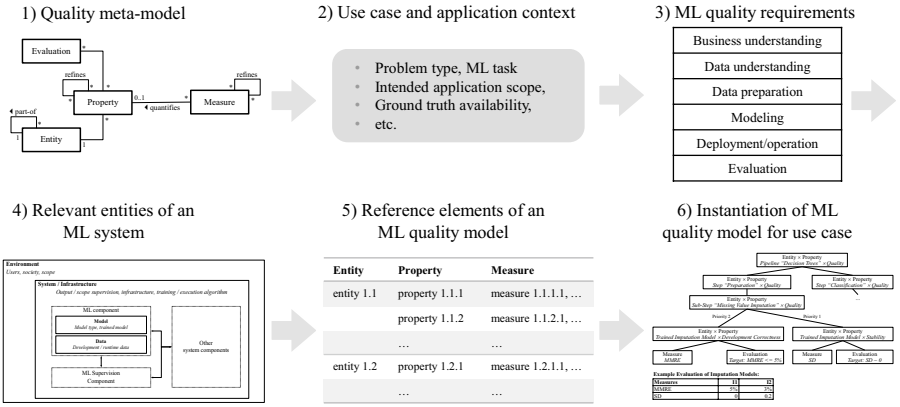


Fig. 2 Overview of the quality model construction process

filling our quality model with the most relevant properties instead of trying to come up with a complete list of all potentially interesting properties.

3. Identify relevant ML quality requirements: Developing an ML model can be split into different stages related to understanding the problem to solve, gathering the required data, and building the ML model itself. From the different stages, we may derive different relevant quality requirements regarding the inputs and outputs of the ML development process. The concrete activities performed depend on the concrete problem to be solved and the ML model to be constructed. This is derived from the application context and the use case.
4. Identity relevant entities of an ML system: In an ML system, the ML model itself is only one entity of many. To build a comprehensive quality model for ML systems, it is important to analyze all relevant entities that could come into play, such as the data, the model itself, the infrastructure on which the model is executed, the execution environment of the overall system, and so on. Again, this highly depends on the application context and the use case.
5. Identify reference elements of an ML quality model: Based on the identified quality requirements from the ML development process and the relevant entities of an ML system, we can create a table of reference elements to be used in an ML quality model. This intermediate step before building the quality model itself is required in order to get a simple overview of relevant entities and their quality properties as well as typical measures for quantifying the properties. For this purpose, we collected existing measures from the literature and from our practical experience in building ML models.
6. Instantiate quality model for use case: In the last step, we built the quality model itself based on the meta-model defined in the first step. For this purpose, reference elements from the previous step were instantiated for the concrete use case. This also included the definition of evaluation rules for quickly identifying quality issues for the concrete use case.

Even though the construction process we followed was only applied for a particular industrial use case and application context, we believe that the steps can be generalized and applied to other cases and contexts as well. In the following sections, we will provide more details for each of the steps listed above.

4 Step 1: Define quality meta-model

In practice, the quality of ML systems is typically considered implicitly, which causes a number of problems. The most basic and common one is an inconsistent understanding of quality by developers and users of ML systems. Additionally, quality criteria considered by system developers differ from the quality requirements of a system's users. Furthermore, data scientists tend to evaluate the fulfillment of quality criteria and their mutual dependencies rather implicitly, which makes it difficult to comprehend the decisions they made regarding quality.

In this section, we propose a solution to this situation. Motivated by quality management in traditional software engineering, we introduce a quality meta-model as a common ground for consistently defining, communicating, evaluating, and controlling the quality of ML systems. There are a number of practical advantages to an explicit quality meta-model. Most of all, it creates a basis for considering the quality of an ML system systematically. It forces one to ask questions about what the quality of a system means, what aspects of quality are most and which are less relevant, how to objectively measure the achievement of quality, what the quality acceptance criteria are, and what the tradeoffs among multiple quality criteria are.

We base the proposed quality meta-model (see Fig. 3) on previous work in traditional software engineering (Nistala et al., 2019), in particular on the Quamoco quality meta-model (Goeb et al., 2015).

The central element of a quality model is a property of an entity, that is, an attribute that characterizes the entity and is related to the quality of the entity. Entity refers to the concrete things that are important for the quality of an ML system (such as data for training models) and its properties represent certain characteristics of these things (such as completeness or consistency). The concept of a property is general and can be used on different levels of abstraction. Entities and their properties may be abstract or specific. The basic difference between the two is that in contrast to specific properties of entities, generic ones are rather difficult to quantify and evaluate. Specific properties of specific entities can be associated relatively easily with measures. An example of an abstract property of an entity could be *quality of a model*; in this case, neither quality nor model is exactly specified and thus difficult to evaluate. An example of a specific property of an entity is *correctness of a classification model*, which can be measured, for example, using F-Score.

To clearly describe system quality from an abstract level down to concrete measurements, abstract properties and entities have to be broken down—respectively through the "refine" and "part-of" relations in the meta-model—into more specific, measurable properties and entities. The concepts defined in the quality meta-model allow for modeling different hierarchies of quality to express divergent views on quality. In the example quality

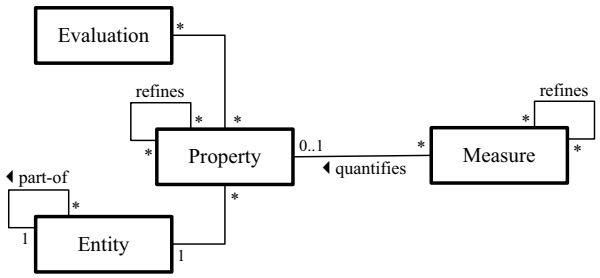


Fig. 3 Quality meta-model for ML systems

model, which we will introduce in Sect. 9, we exemplify two main hierarchies: processing pipeline and quality hierarchy.

Both properties and entities need to be concrete enough to be measured, particularly in the leaves of a quality model. For this purpose, a quality meta-model defines the concept of measure for a property of an entity. A measure is a concrete description of how a specific property (of a specific entity) should be quantified in a specific context. For example, *correctness of a classification model* can be measured using precision, recall, or F-score measures.

Quality evaluation comprises four basic elements: measurement, evaluation, aggregation, and interpretation. *Measurement* consists of the collection of measurement data for the factors specified at the lowest level of the quality model's hierarchy according to the measures defined in the quality model. *Evaluation* involves assessing the fulfillment of quality preferences associated with the factor. *Aggregation* comprises the synthesis of assessments obtained on individual child factors in a bottom-up manner throughout the quality model hierarchy into an overall assessment of a system under assessment. Finally, *interpretation* is the translation of the potentially abstract quality assessments into evaluations that are understandable (intuitive) for human decision makers.

Before a quality assessment can be executed, it first needs to be operationalized. The measurement step may require defining additional measures to ensure that measurement data collected for the same measure across different products are comparable. The evaluation step requires defining preference functions to model the decision criteria with respect to the measures defined for the factors. In the simplest case, a preference can be modeled as an acceptance threshold (*target*), that is, a specific boundary value that determines which values of a measure are preferred (acceptable) and which are not. Aggregation requires defining the aggregation operator to synthesize the assessments of the individual properties of entities across the quality model hierarchy into the total assessment. This may include defining preferences regarding the relative importance of child nodes defined for the same parent node in the quality model hierarchy to account for potential decision trade-offs between child nodes. The relative importance of properties of entities can be quantified through numerical weights. Finally, interpretation requires defining an interpretation model that will translate rough assessments into an understandable evaluation that decision makers can interpret properly, for instance to derive appropriate improvement actions. Users of the quality model and of the quality assessment method can (and should) perform operationalization prior to quality assessment in order to make the approach fit their specific context (e.g., they should adjust preference functions and weighting to reflect their specific preferences regarding the importance of individual properties of entities).

5 Step 2: Define use case and application context

Different elicitation techniques can be used in order to define the relevant use cases and the application context of the system under study. As part of this project, reference materials were made available by Fujitsu and workshops were held by a focus group. The industrial use case can be described as follows. The Accounting Center of Fujitsu receives purchase order requests (POR) in digital form that needs to be categorized for further treatment. POR are semi structured text documents. This task was traditionally done by human operators and is now performed by an ML component. The corresponding ML task is classification, and new POR need to be classified into 3 predefined categories. The ML component

is trained with a ground truth dataset consisting of labeled examples of past POR that have been categorized by human operators. In order to deal with wrong classifications, a monitoring system was implemented in conjunction with a correction engine based on expert rules. The ML component was re-trained when too many categorization failures were detected. The studied system thus includes at least 3 main components: the ML classification component itself, a monitoring system, and a correction engine.

The goal of this system is to reduce operating cost and time at an acceptable level of classification accuracy (comparable to humans). During the interviews, other related quality issues were mentioned. First, the development and operation of ML components were seen as complex and associated with high cost and risks. Several areas of expertise came into play when developing and operating this system. This led to communication and coordination problems. Furthermore, when wrong classifications occurred, finding the root cause of such a failure was not trivial. The principal quality aspects deemed relevant for this use case were the functional correctness of the system (in terms of classification accuracy) but also the development and operation costs (in terms of time).

6 Step 3: Identify relevant ML quality requirements

Many factors can influence the quality of a software system (code, hardware, development process, usage scenarios, etc.). One of the goals when designing a quality model is to cover all relevant quality properties. Since software engineering (whether ML-based or not) can become quite complex, it is necessary to have a systematic approach that helps to define and refine which quality properties have to be modeled. In this section, we propose the use of a process model (CRISP-DM) as a basis for listing potential entities and their properties that can influence the quality of an ML system.

The CRISP-DM model (short for CRoss Industry Standard Process for Data Mining) (Shearer, 2000) is an open standard describing the different phases encountered in data analysis projects. This model proposes six phases, namely, business understanding, data understanding, data preparation, modeling, evaluation, deployment. It is currently thought to be the de-facto standard for projects developing ML components, according to several polls (for instance Shearer, 2000), and although extensions have been proposed to this model (Kurgan & Muslek, 2006; Mariscal et al., 2010; Martinez-Plumed et al., 2020; IBM, nd; Microsoft, 2019), recent case studies like Amershi et al. (2019), Lwakatare et al. (2020), and Martinez-Plumed et al. (2020) show that the proposed six phases are generic enough. The CRISP-DM model is sometimes described as a waterfall model (Mariscal et al., 2010). However, for our purpose, abstracting from the sequential aspect and only considering the phases and the corresponding activities help to cover the different aspects influencing the quality of the ML system.

Business understanding Whether ML-based or not, a software component always meets certain requirements: It is developed in a certain business context; there are objectives to be achieved; the component is supposed to bring added value; it has a cost and may involve risks. Stakeholders, expected usage scenarios, and key performance indicators (KPIs) are usually defined during this phase. Compared to more "classical" software components, ML components bring some specific requirements with them. For example, a business objective must be transformed into a clear analysis objective (e.g., grouping customers into different segments based on their purchase history). The way to evaluate its success needs to

be established (e.g., a new web shop feature improves the click rate by x percentage points, as measured in an A/B test). The business context imposes assumptions and constraints on how components are designed and built (e.g., the ML component must run on an embedded device, so time and memory complexity must be limited), on data availability (i.e., what data has been collected or can be collected and labeled), and on requirements for privacy, safety, security, fairness, etc. It also affects choices regarding how to select and evaluate models (e.g., the ML model should not have a false alarm rate greater than x%), how to handle changes in concepts and distribution, and how to perform new training (e.g., run data should be sufficiently similar to training data—up to a certain notion of similarity, an alarm should be triggered when the model extrapolates).

Data understanding The data understanding phase can be seen as a requirement engineering phase specifically directed towards the data (it usually goes hand in hand with the business understanding phase). Indeed, the type of analysis method and the corresponding evaluation measures that can be used depend on several data-related factors (besides the analysis objective): the type of data available (e.g., unstructured data like text or images vs. structured data like tabular data), whether some ground truth is available (see the discussion in the previous section), the quality of the data (e.g., its resolution, its representativity; whether noise, outliers, or missing values are present, etc.), and how the data is gathered.

Data preparation Together with the modeling phase, the data preparation phase belongs to the concretization phase of the ML component. Here, several artifacts are implemented in order to transform the raw data into data that can be fed into analysis methods. The way the data is prepared (e.g., labeling, removal, or imputation of outliers or missing data, features engineering, features selection) have a cost and an impact on the quality of the analysis results. Several data preparation algorithms use internal models and posit some hypotheses about the data (e.g., the data follows a Gaussian distribution), which might not coincide with reality. Additionally, adjustments need to be made depending on privacy requirements (e.g., anonymization of sensitive features) or fairness requirements (e.g., preferential sampling). Data used for model building and model evaluation might also originate from different sources (e.g., simulation vs. real data), be distributed differently, and require different data preparation steps.

Modeling The modeling phase is probably the tip of the iceberg when it comes to developing ML components. This is where methods such as ML are applied to form and evaluate the artifacts that make up the component. As previously mentioned, this phase is strongly linked to the data preparation phase. In general, an ML component is composed of several sub-components from these two phases. The quality of the ML model is impacted by several aspects: the type of task to be solved (e.g., classification, clustering, regression, anomaly detection, dimensionality reduction, etc.), the type of model (neural network, decision tree, etc.), the data used for building (i.e., training), and evaluating the developed artifacts, as well as the manner in which the data is separated for training and validation, together with requirements on runtime complexity or safety constraints. Since the way the component is created is experimental, the way these experiments are managed (using hyperparameter search, cross-validation, independent train-test split) also plays a role in terms of quality. The modeling phase also contains an evaluation part that aims at evaluating the trained component with regard to the available data. It does so by measuring performance

measures (such as precision, recall, etc. for classification tasks), performing sensitivity analysis, or testing against adversarial examples.

Deployment (and operation) The original paper left the operating phase out. For our purpose, we include it. The quality at runtime is impacted by the data that flows into the ML component (e.g., the runtime data should have similar characteristics as the training and evaluation data to make reliable assumptions on the model performance). Therefore, monitoring the dataflow, the application context, and the ML component's behavior (e.g., by estimating the output uncertainty of the ML model) is needed in order to evaluate its quality. The second point is that operation requirements (such as real-time constraints or the runtime architecture) may differ from those during development. Indeed, an ML component may require large amounts of data as well as specific libraries and hardware architectures (such as GPUs) to be built (trained). However, once developed, this component may have to be deployed in a totally different environment (e.g., in a mobile device using a different programming language). Here, the way the deployment is managed (for example, by using exchange formats such as ONNX or PMML, by using versioning and automated CI/CD tools, etc.) also impacts the quality of the ML system. The deployment phase may also introduce evaluation methods such as integration testing or runtime monitoring.

Evaluation This was originally defined as the 5th phase, and its goal is to answer whether the ML component properly achieves the business objectives. Note that the modeling phase and possibly the deployment phase also contain an evaluation part. Here, however, the objective is to evaluate the system from a business perspective. In other words, does the system meet the KPIs defined above? This could be done, for example, through controlled experiments such as A/B tests. This allows creating feedback loops in order to select which version of an ML component gives the best results (from a business perspective). The first thing to notice is that in order to perform such an evaluation, the goals, the measures, and the methodology of the experiment have to be defined. These experiments may involve external software components (such as traffic redirection) whose internal quality may impact the way an ML component is evaluated.

A process model helps to systematically go through all the lifecycle phases of a system and list quality requirements that may be relevant for a given use case. To measure quality, it is also necessary to detail which entities play a role and how one can measure their corresponding quality properties. In doing so, the list of quality requirements aids to identify the quality properties of these entities.

7 Step 4: Identity relevant entities of an ML system

In the previous section, we looked at the process of developing and maintaining ML components in order to see which aspects can influence the quality of the system and which quality requirements are needed. In order to build a quality model with a systematic structure, we now propose different "views" that help to categorize quality properties and their corresponding quality measures together with the entities to be measured. This also helps to cover as many relevant quality properties as possible, as we look at influences on quality from a second perspective.

The views we propose are: model view, data view, system view, infrastructure view, and environment view (see Fig. 4 for an illustrative overview). Note that a given quality model may or may not use all the views, as the relevant ones are selected according to the use case.

Model view The model view is concerned with quality properties belonging to the artifacts that are trained using data in order to perform a given task (e.g., classification, regression, dimensionality reduction, etc.). An ML component usually consists of several subcomponents organized in a directed acyclic graph (also called a pipeline) (Amershi et al., 2019). The specificity of such a component is first the way it is built. We have to distinguish between the development phase (where the training and the evaluation of the pipeline are done) and the operation phase (where the artifacts created in the previous phase are deployed and used in production, i.e., at runtime), because these two phases may be implemented with different technologies (e.g., R/Python for learning, Web/Java on the application side) or have different quality demands (e.g., using a large quantity of data at training time, operating under short latency at runtime, etc.).

In the model view, we have made a distinction between what we call a *model type* (e.g., decision tree, neural network, etc.) and a *trained model* (e.g., a specific instance of a neural network trained on a specific dataset using a specific *training algorithm*). Again, the goal of this distinction is to separate quality properties related to a specific entity instance from those related to the entity type. For example, the *appropriateness* of a given model applies to a *model type* (like the family of decision trees), whereas the *goodness of fit* applies to a specific trained instance. Note that we also separate the model from its *training algorithm* (i.e., the algorithm that takes *training data* and a *model type* as input and outputs a *trained model*) and its *execution algorithm* (i.e., the algorithm that takes *runtime data* and a *trained model* as input and outputs a decision, for instance a classification of inputted *runtime data*). The argumentation is that the training and execution algorithms are pieces of "classical" software whose quality properties can be described and measured using existing standards.

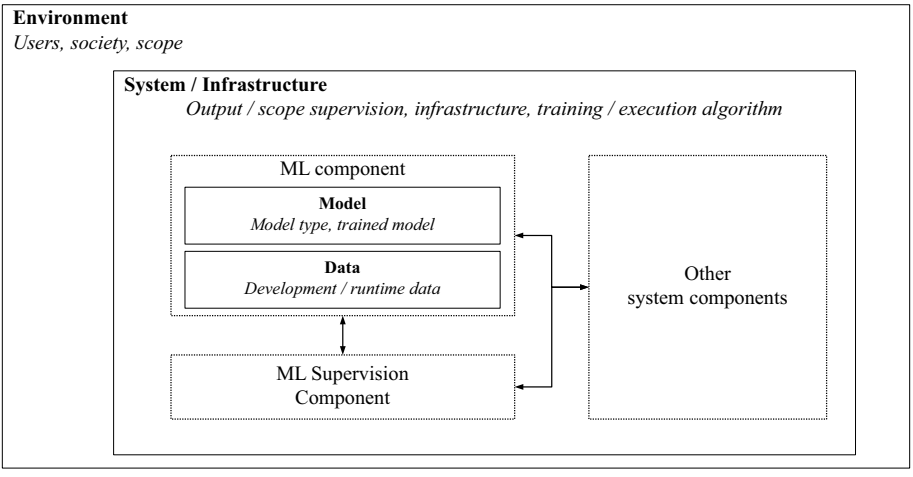


Fig. 4 Overview of the different views on the software system and the entities that influence the system's quality

Data view The data view is concerned with the quality properties related to the data. The term data here describes the data that is used as input for the ML component. We further distinguish between *development data*, i.e., data used during the development phase to train the component, and *runtime data*, i.e., the dataset used during the operation phase (i.e., the data that flows into the *trained model* when deployed and used in production). We make this distinction first of all because data used during model training and during application differs in nature. Training data typically has a "batch" nature, that is, it consists of a large set of data collected over a certain period, on which models are trained and validated. Runtime data, on the other hand, typically has a "stream" nature, that is, it represents data that changes continuously over time. The nature of data determines its specific quality properties as well as the applicable preparation and training algorithms. Combining batch and stream processing poses a significant challenge in the development of AI systems (Marz & Warren, 2015), e.g., because not all processing algorithms used during development to train models on batch data are easily applicable (transferable) to runtime environments where trained models are applied on stream data. Furthermore, training and runtime data can be associated with different physical objects, be stored in different databases, and be preprocessed or accessed differently during the development and operation phases. Therefore, different quality properties apply either to each dataset separately or to both (for example, by comparing the representativeness of the *development data* with regard to the *runtime data*). We pushed the distinction even further concerning the *development data*. Indeed, the process of training an ML component requires splitting the *development data* into different subsets: the so-called training, validation, and test subsets. The *training subset* is used to determine the model parameters during training. The *validation subset* is used for hyper-parameter tuning (e.g., the maximum depth of a decision tree). Finally, to provide an unbiased evaluation of the *trained model*, a *test subset* is used. Note that the *test subset* is supposed to be independent of the training and validation subsets. The way the *training*, *validation*, and *test subsets* are chosen has an impact on the quality of the evaluation of the trained model.

System view First, an ML component is usually organized in a pipeline of tasks. Developing such a pipeline is by its very nature experimental. A given pipeline may be trained several times with different model types, training algorithms, or datasets in order to find the best combinations (also known as the Combined Model Selection and Hyperparameter optimization (CASH) problem (Hutter et al., 2018). The way the search is done and the way the sub-components are connected has an impact on quality (see, for example, the problem of data leakage (Kaufman et al., 2011)). Second, once a model has been developed (i.e., trained), it needs to be deployed. As already stated above, the development and the runtime environment may be of a different nature. Therefore, a specific architecture needs to be put in place in order to export the trained models (e.g., using serialization, exchange formats like ONNX, PMML, or using model management tools) (Marz & Warren, 2015). Finally, the deployed component is itself part of a larger system, i.e., it consumes data from one or several sources and interacts with other ML-based or classical components. Since a decision outputted by an ML component is always subject to uncertainty, and since wrong decisions might impact the system's overall quality, considering the flow of information from the system input through all components to the system output is important for understanding the impact of a given ML component's quality on the overall system behavior. Typical quality properties related to the system view include, among others, data dependencies and feedback loops (Sculley et al., 2015). For example,

the input data can be monitored in order to detect either data drift or anomalous datapoints. The output of the component can also be monitored in order to detect and correct wrong decisions. This monitoring also has its own quality properties, which may be relevant for the use case at stake (e.g., monitoring effectiveness and efficiency).

Infrastructure view What we call the infrastructure view is closely related to the system view. However, here the view is more focused on the quality properties related to how the system is concretely implemented (e.g., hardware, training libraries). We decided to separate these two views in order to highlight some specificities of ML components. For example, the efficiency of the *training* and *execution algorithms* is a property that belongs to this view. The same applies to the suitability of the infrastructure either for training or for executing the components. For example, current trained deep learning models used for natural language processing are several gigabytes in size, and require several days (or weeks) of training on dedicated hardware machines (GPU clusters). The trained model cannot be executed on embedded devices due to computational and storage limitations.

Environment view The *environment* consists of elements that (1) are external to the system under consideration and (2) interact either directly or indirectly with the system. This includes the users. For ML systems, several environmental aspects may have a direct influence on the quality. These include, for example, aspects causing quality deficits in the data. This is strongly related to the notion of concept drift. Since an ML component is built for and tested in a given context of use (or target application scope), its quality will decrease when this context changes (Kläs & Vollmer, 2018). A self-adapting component dependent on the environment also raises further quality-related challenges (see, for example, the problems faced by the Microsoft chatbot Tay). Vice versa, an ML component can also have an impact on its environment, e.g., in terms of resource usage or societal discrimination (HILEG, 2019).

8 Step 5: Identify reference elements of an ML quality model

In this step, we created a table of reference elements to be used in an ML quality model. We used the quality requirements and the views defined in the previous sections to select pertinent entities for the use case. From that point on, we identified quality properties of interest for all entities. For each property, we either give examples of concrete measures for objectively evaluating the quality, or, if this is not possible, define examples for items one would have to check in order to address the respective quality. This was done based on existing literature in the field and our practical experience of building ML models.

Table 1 presents the list of reference elements from which we built a concrete quality model later on. The elements were designed to be specific enough to address the described use case appropriately (including supervision- and classification-related quality properties), but also contains generic elements to allow it to be applied to other (similar) use cases (such as most properties related to data and model).

Table 1 Overview of the derived reference elements of the quality model

View	Entity	Property	Example quality measures and checklists
Model	Model type	Appropriateness: Degree to which the *model type* is appropriate for the current task (e.g., classification, etc.) and can deal with the current data type (e.g., numerical, categorical)	Prerequisites for model type
	Trained model	Development correctness (goodness of fit): ability of the model to perform the current task measured on the development dataset	Precision, Recall, F-score, etc. for training
		Runtime correctness (goodness of fit): same as above measured on the runtime dataset	Precision, Recall, F-score, etc. at runtime
		Relevance (bias-variance tradeoff): degree to which the model achieves a good bias-variance tradeoff (neither underfitting nor overfitting the data)	Variance of cross-validation goodness of fit
		Robustness: ability of the model to handle noise or data with missing values and still make correct predictions	Equalized Loss of Accuracy (ELA)
		Stability: degree to which a trained model generates repeatable results when trained on different subsets of the training dataset	Leave-one-out cross-validation stability
		Fairness: ability of the model to output fair decisions	Equalized odds
		Interpretability: degree to which the trained model can be interpreted by humans	Complexity measures (e.g., no. of parameters, depth)
		Resource utilization: resources used by the model when it is already trained	Required storage space

Table 1 (continued)

View	Entity	Property	Example quality measures and checklists
Data	Development data	Representativeness: degree to which the data is representative of the statistical population	Statistical tests (e.g., two-sample t-test, etc.)
		Correctness: degree to which the data is free from errors	Outlier detection measures (e.g., Z-score)
		Completeness: degree to which the data is free from missing values	No. of missing values
		Currentness: degree to which the data is up to date w.r.t. the current task	Age of data
		Intra-Consistency: consistency of the data within a dataset, e.g., the data does not contradict itself or the formatting is consistent	Value ranges, word counts
		Train/Test Independence: degree to which the *training* and *test subsets* are independent of one another	Statistical tests (e.g., two-sample t-test, etc.)
		Balancedness: degree to which all classes (labels) are equally represented in the dataset	Ratio of classes
		Absence of bias: degree to which the data is free from bias against a given group	Ratios of groups
	Development and runtime data	Inter-consistency: consistency between different datasets, e.g., formatting, sampling methods used	Value ranges, crosswise outlier detection measures
Environment	Training process	Environmental impact: degree to which the training process impacts the environment	Energy consumption
	Society	Social impact: degree to which the ML component impacts society	Impact on employees
	Scope	Scope compliance: degree to which the application of the ML component respects its intended scope of use	Value ranges, novelty detection measures

Table 1 (continued)

View	Entity	Property	Example quality measures and checklists
System	Output supervision	Effectiveness: degree to which the output supervision algorithm detects false outcomes of the ML component	False positive/negative detection rate
		Supervision overhead/efficiency: resources used for monitoring a given ML component	Time, memory used, etc.
	Scope supervision	Effectiveness: degree to which the scope supervision algorithm detects context changes	No. of out-of-scope cases
		Supervision overhead/efficiency: resources used for monitoring the application scope	Time memory used, etc.
	Other non-ML components	Here we refer to the relevant subset of the quality properties of the standard ISO/IEC 25010, which are not listed here for space reasons	
Infrastructure	Infrastructure	Infrastructure suitability: degree to which the infrastructure matches the ML component needs (e.g., in terms of hardware type, computation capability, bandwidth, memory, etc.)	Computational and storage capabilities
	Training algorithm	Training efficiency: resources used for training a given model	Time, memory used, etc.
	Execution algorithm	Execution efficiency: resources used for executing a given trained model	Time, memory used, etc.

9 Step 6: Instantiate ML quality model for use case

Based on the reference elements worked out for the use case of Fujitsu's Accounting Center and the quality meta-model presented in Sect. 4, it is now possible to derive a concrete quality model.

When building a quality model for an AI component, it is important to note that you typically do not make use of a single algorithm, for which the quality should be determined, but you rather have a whole solution idea composed of different steps, a so-called processing pipeline, which makes use of different data-based models for which quality has to be determined individually.

In the use case described above, we could make use of a solution idea based on so-called decision trees for building a classification model. Before running the decision tree classification algorithm for building a classification model, the data has to be prepared accordingly, e.g., missing or incomplete values must be dealt with. Therefore, one sub-step of preparation could be the imputation of missing values using an imputation model. For each part of the processing pipeline, different entities and their different properties as defined in Table 1 can be of relevance. Example entities include the trained model or the data used for training the model, while example properties include completeness or correctness. Both entities and properties can create hierarchies. Figure 5 shows an example of what a simplified entity hierarchy could look like.

In correspondence to the entity hierarchy, one could now think of relevant properties of certain entities for addressing the quality of the whole pipeline. An example breakdown structure is shown in Figs. 6 and 7. On the higher levels of the hierarchy, we are interested in the general quality (*property*) of each step of the pipeline (*entity*). For instance, when using an imputation model, one could be interested in the development correctness and stability (*properties*) of the trained model (*entity*). The corresponding excerpts of the quality model are shown in Fig. 6. The processing pipeline is modeled as a hierarchy of entities (steps and sub-steps). As can be seen in the figure, the quality model generically talks about the "quality" of the corresponding step of the pipeline. On the lower levels of the quality model, all relevant entities and properties can be found for each step of the pipeline, such as "trained model × stability". Each property of an entity has a set of measures assigned to it and an evaluation rule describing how to evaluate the measures. These rules objectively specify our quality requirements. For instance, the imputation model is considered correct if the mean magnitude of relative error (MMRE) is lower than or equal to 5%. Or, the imputation model is considered stable if the standard deviation (SD) of different runs of the model is close to zero.

Based on these evaluation rules, it is now possible to compare different models objectively. At the bottom of Fig. 6, two example imputation models are characterized using the measures specified. As can be seen, I2 performs better regarding MMRE, but fails regarding the SD criterion.

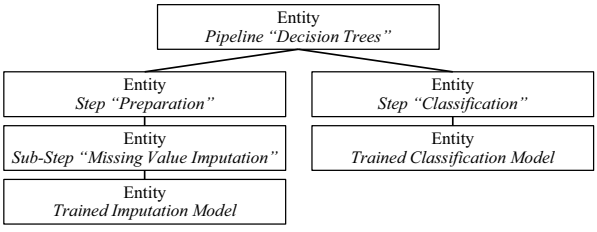
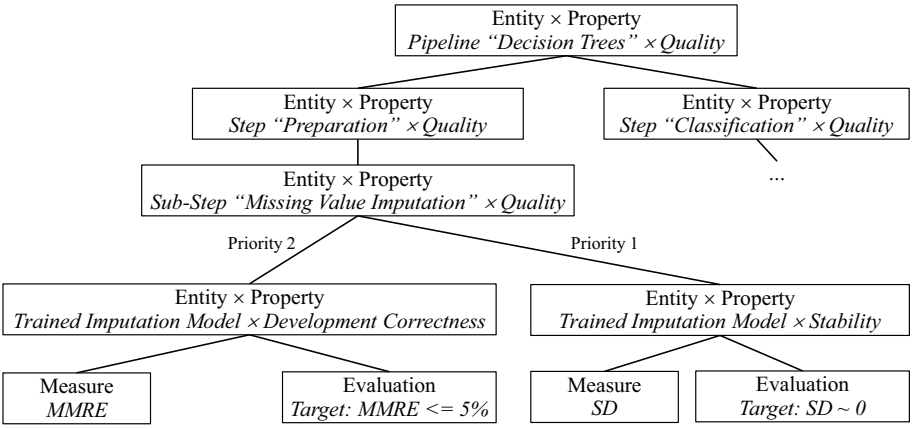


Fig. 5 Example hierarchy of entities to address for processing the pipeline "Decision Trees"



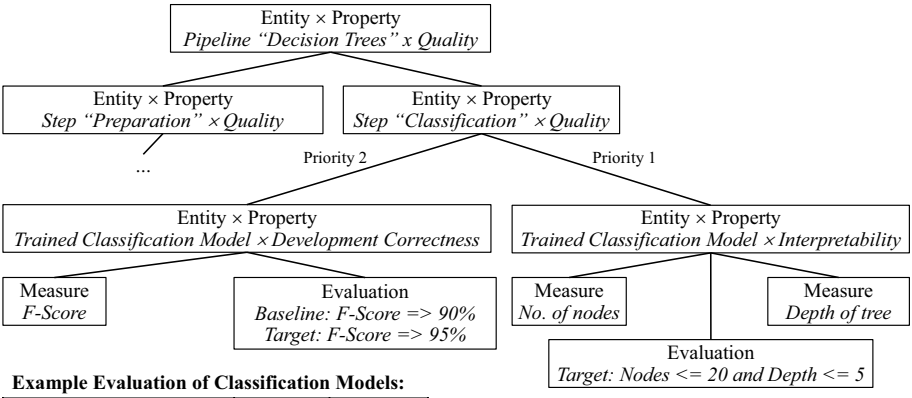
Example Evaluation of Imputation Models:

Measures	I1	I2
MMRE	5%	3%
SD	0	0.2

Fig. 6 Example quality model for the processing pipeline of the "Preparation" step

In the same way, the quality model can be completed for the other steps of the processing pipeline. Figure 7 shows further properties, measures, and evaluation rules for the classification step. In practice, many additional properties have to be considered, not only regarding the trained model, but also other entities addressing the data, the environment, the system, or the infrastructure.

The example presented here only makes use of evaluation rules on the leaf nodes of the quality model. Depending on the required degree of formality, an aggregation of measures can be performed along the hierarchy of properties and entities with corresponding



Example Evaluation of Classification Models:

Measures	C1	C2
F-Score	95%	97%
Nodes	20	300
Depth	3	10

Fig. 7 Example quality model for the processing pipeline of "Classification" step

evaluation rules on each level. In that case, one would end up with a single quality index on the top level. Using this kind of aggregation mechanism is quite common for software quality models when presenting the results of a quality evaluation to different software managers.

10 Discussion of validity

In this article, we first proposed a categorization of quality properties as well as entities in the form of different views/entities. This classification is the result of a literature-based review, discussions with industrial partners, and our own experience in the development of ML components. To scientifically assess and consolidate a useful and systematic grouping of quality properties for ML systems, several iterations will be necessary (e.g., case study, systematic literature review, mapping study).

We also derived a quality model specifically tailored for a given use case. The definition and the relevance of the quality properties were first discussed internally in a workshop with experts. Later, three case studies with a focus on requirements engineering for ML systems were conducted by Fujitsu Laboratories (see the full details in (Nakamichi et al., 2020)). The goal of the first case study was to verify that the requirement analysts actually consider the different properties (22 in total) in their requirement definition. For each property, 3 ML developers, 2 software developers, and 1 project manager were asked (1) whether they know and use it and (2) whether it requires customer agreement and what is the status of the agreement. The results showed that, in this specific case, most of the properties (77%) were already known and used (Nakamichi et al., 2020). The objective of the second case study was to verify whether the developers were aware of the proposed quality measures (34 in total). For each quality measure, 3 ML developers, 2 software developers, and 1 project manager were asked whether they know it and what is its measurement status in their respective projects. The results showed that 45% of the submitted quality measures were already used, 28% were not used yet (answer "want to measure"), 15% were not measured because of technical or cost difficulties, and 10% were not needed (Nakamichi et al., 2020). The goal of the third case study was to probe the effectiveness of the quality measures (6 in total). For each quality property and a corresponding measure, 3 ML developers and 3 software developers were asked about the measurement status and the effectiveness of the measure. The results showed that 83% of the submitted measures were deemed effective (Nakamichi et al., 2020). The performed case studies provide a first confirmation that the quality properties identified are valid and meaningful for developers. In this previous paper, the authors did not go into the details of the how the quality model presented was built.

In terms of limitations, we see two main aspects:

1. We used a specific process model for listing potential entities and their quality properties. As stated above, new process descriptions have been proposed in the literature. Although the CRISP-DM model is generic enough (see, for example, the different reviews done in this field, like (Martinez-Plumed et al., 2020; Mariscal et al., 2010; Kurgan & Muslek, 2006)), some phases may still be missing. Furthermore, we did not investigate other process-related aspects yet, e.g., what qualities have to be assured in which activity and handled by which role. We believe that the proposed views/entities can help to establish a mapping between roles (e.g., Data Scientist, Data Engineer, etc.) and quality properties or measures. For example, Data Scientists are usually in charge

of building models and are in direct line when it comes to measuring the impact of data quality on the models' outcomes. However, data engineers are the ones that can usually implement new data quality improvement actions. Architects with a good understanding of data science methods (such as ML) will be needed to solve problems on the system level.

2. Second, our viewpoint for defining the quality model was more the data science perspective. Integration with classical software/systems engineering qualities (such as those defined by ISO/IEC 25010) is missing. There is as yet no consensus on the naming of quality properties related to ML components. Furthermore, whereas some of the proposed properties can be easily classified under existing ISO/IEC 25010 ones (e.g., the model's Goodness of Fit could potentially belong to Functional Correctness), others may be more difficult to classify (such as Scope Compliance). Whether the ISO/IEC 25010 is the right framework for ML components is still an open issue.

11 Lessons learned

We are completely aware that the model we developed is quite specific to the use case in which it was applied and that other use cases may require different quality properties and, in consequence, different measures. However, we would like to share an excerpt from the lessons we learned from following the described methodological approach. Even though some of these are known from other fields, we nonetheless think it is worth mentioning them in the context of developing ML systems:

1. Context and use case must be clear. As pointed out before, there are many application fields and potential ML-based solutions available. It is very important to be as clear as possible about the general application context. ML components should never be used just for the sake of being fancy, but always because there is the profound assumption that they will add concrete value for the application context. The quality properties that are important mainly depend on this.
2. Iterative approach: The ML model, its application context, and its use case have to be adjusted over time and some initial assumptions will turn out to be false. Therefore, it is important to follow an iterative approach when developing the ML system and to be able to quickly identify dead ends and take different paths. Having a clear picture of what quality properties are important and how to quantify them is crucial for this, as it allows us to immediately see whether we can fulfill them with our solution path.
3. Multidisciplinary work: As we stated at the beginning of this article, different kinds of knowledge must come together to develop quality-aware ML systems. For instance, a data scientist knows how to measure the fairness or stability of the trained model, a software/system engineer knows how to assure the quality of the overall system, and a domain expert knows whether the ML system really solves the problem better than a traditional software system.
4. The devil is in the details: We learned that it is easy to talk about abstract generic quality properties, such as those defined by ISO/IEC 25010, on a high level. To define meaningful quality properties, we had to break them down into concrete qualities of entities and define how to operationalize these properties with measures.
5. Quality-aware process/guidelines: Even though there are defined processes for ML model building (such as CRISP-DM) and for software engineering (such as rich and

agile processes) with elaborate practices for improvement (such as DevOps approaches), an integrated process is missing, nor do guidelines exist on how to bring everything together with a clear focus on the quality of ML systems.

12 Conclusions

This article presented how to construct a concrete quality model for an ML system based on an industrial use case. Compared to the existing body of knowledge in the field of quality modeling, and specifically with regard to the quality of ML systems, we added the following aspects:

- Systematic construction process for quality models of ML systems.
- Adaptation of the concept of software quality meta-models to ensure uniform documentation of relevant entities, quality properties, measures, and evaluation methods.
- Table of reference quality elements for ML systems as a systematic overview and classification of relevant views, entities, quality properties, and measures for ML systems based on existing research.
- Example instantiation of a quality model for a concrete industrial use case (purchase order requests) to illustrate the applicability of the different steps of the construction process.
- Lessons learned from applying the construction process in order to support other researchers and practitioners in avoiding typical hurdles and issues.

Even though the proposed construction process lacks a comprehensive evaluation, it was designed and performed together with researchers and practitioners in the field and may guide others in the systematic construction of quality models for ML systems.

The resulting list of reference quality elements was specifically derived for the industrial use case, but may be generalizable for similar kinds of ML systems.

Regarding future work, we plan to perform more case studies to empirically validate the construction process on other use cases regarding its applicability and usefulness. In particular, we want to apply the process to other ML tasks (like regression or unsupervised learning) and learn about the impact on the quality model. Moreover, we plan to develop an approach for evaluating the overall quality of ML systems based on a predefined quality model. The overall goal is to have reference quality models for evaluating certain types of ML systems.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

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. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291–300.

Arpteg, A., Brinne, B., Crnkovic-Friis, L., & Bosch, J. (2018). Software Engineering Challenges of Deep Learning. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 50–59. IEEE, [S.l.].

Barocas, S., & Boyd, D. (2017). Engaging the ethics of data science in practice. *Communications of the ACM, 60*, 23–25.

Belani, H., Vukovic, M., & Car, Z. (2019). Requirements Engineering Challenges in Building AI-Based Complex Systems. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), pp. 252–255. IEEE.

Bosch, J., Olsson, H. H., Crnkovic, I., Wang X., Munch J., Suominen A., Bosch J., Jud C., & Hyrynsalmi S. (2018). It takes three to tango: Requirement, outcome/data, and AI driven development. *CEUR Workshop Proceedings, 2305.*

Calder, M., Craig, C., Culley, D., de Cani, R., Donnelly, C. A., Douglas, R., Edmonds, B., Gascoigne, J., Gilbert, N., Hargrove, C., et al. (2018). Computational modelling for decision-making: where, why, what, who and how. *Royal Society Open Science, 5*, 172096.

de Souza Nascimento, E., Ahmed, I., Oliveira, E., Palheta, M. P., Steinmacher, I., & Conte, T. (2019). Understanding Development Process of Machine Learning Systems: Challenges and Solutions. *In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM),* pp. 1–6.

Edmonds, B. (2002). The Use of Models - Making Mabs Actually Work.

Edmonds, B., Le Page, C., Bithell, M., Chattoe-Brown, E., Grimm, V., Meyer, R., Montañola-Sales, C., Ormerod, P., Root, H., & Squazzoni, F. (2019). Different Modelling Purposes. JASSS 22.

Emmons, S., Kobourov, S., Gallant, M., & Börner, K. (2016). Analysis of network clustering algorithms and cluster quality metrics at scale. *PLoS One, 11*, e0159161.

Epstein, J. M. (2008). Why model? *JASSS, 11,* 12.

Goeb, A., Heinemann, L., Kläs, M., Lampasona, C., Lochmann, K., Mayr, A., Plösch, R., Seidl, A., Streit, J., & Trendowicz, A. (2015). Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology, 62*, 101–123.

Hamada, K., Ishikawa, F., Masuda, S., Matsuya, M., & Ujita, Y. (2020). Guidelines for Quality Assurance of Machine Learning-based Artificial Intelligence. In: SEKE2020: the 32nd International Conference on Software Engineering & Knowledge Engineering, 335–341.

HLEG, A. (2019). High-Level Expert group on artificial intelligence: Ethics guidelines for trustworthy AI. *European Commission.*

Horkoff, J. (2019). Non-Functional Requirements for Machine Learning: Challenges and New Directions. In: 27th International Requirements Engineering Conference (RE2019), pp. 386–391. IEEE Computer Society, Conference Publishing Services, Los Alamitos, California.

Hossin, M., & Sulaiman, M. N. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. IJDKP 5, 1–11.

Hutter, F., Kotthoff, L., & Vanschoren, J. (2018). (eds.): Automated Machine Learning: Methods, Systems, Challenges. Springer.

IBM. (nd). Analytic Solutions Unified Method. Implementation with Agile Principles, checked on 5/8/2019.

Ishikawa, F. (2018). Concepts in Quality Assessment for Machine Learning - From Test Data to Arguments. In: Trujillo, J.C.e., Davis, K., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (eds.) Conceptual modeling. 37th International Conference, ER 2018, Xi'an, China, Proceedings / Juan C. Trujillo, Karen C. Davis, Xiaoyong Du, Zhanhuai Li, Tok Wang Ling, Guoliang Li, Mong Li Lee (eds.), pp. 536–544. Springer, Cham, Switzerland.

Ismail, A., Truong, H.-L., & Kastner, W. (2019). Manufacturing process data analysis pipelines: a requirements analysis and survey. *Journal of Big Data, 6.*

ISO/TS 8000. (2011). Data Quality.

ISO/IEC 25010. (2011). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

Kästner, C., & Kang, E. (2020). Teaching Software Engineering for AI-Enabled Systems. In: The 42nd International Conference on Software Engineering (ICSE 2020). *Software Engineering Education and Training.*

Kaufman, S., Rosset, S., & Perlich, C. (2011). Leakage in data mining. In: Apte, C., Ghosh, J., Smyth, P. (eds.) Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, Ca, USA, p. 556. ACM, New York.

Kläs, M., & Vollmer, A. M. (2018). Uncertainty in machine learning applications: a practice-driven classification of uncertainty. In M. Hoshi & S. Seki (Eds.), *Developments in Language Theory, 11088.* (pp. 431–438). Springer International Publishing.

Kleinberg, J., Mullainathan, S., & Raghavan, M. (2016). Inherent Trade-Offs in the Fair Determination of Risk Scores. arXiv.org.

Kumeno, F. (2020). Software engineering challenges for machine learning applications: a literature review. *IDT, 13*, 463–476.

Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world.

Kurgan, L. A., & Muslek, P. (2006). A survey of knowledge discovery and data mining process models. *The Knowledge Engineering Review, 21*, 1–24.

Lorenzoni, G., Alencar, P., Nascimento, N., & Cowan, D. (2021). Machine Learning Model Development from a Software Engineering Perspective: A Systematic Literature Review.

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, 106368.*

Mariscal, G., Marbán, Ó., & Fernández, C. (2010). A survey of data mining and knowledge discovery process models and methodologies. *The Knowledge Engineering Review, 25*, 137–166.

Marselis, R., & Shaukat, H. (2018). *Machine Intelligence quality characteristics*. How to measure the quality of Artificial Intelligence and robotics.

Marselis, R., Shaukat, H., & Gansel, T. (2017). Testing of Artificial Intelligence. Sogeti.

Martinez-Plumed, F., Contreras-Ochando, L., Ferri, C., Hernandez Orallo, J., Kull, M., Lachiche, N., Ramirez Quintana, M. J., & Flach, P. A. (2020). CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories. *IEEE Transaction on Knowledge and Data Engineering, 1.*

Marz, N., & Warren, J. (2015). Big data. Principles and best practices of scalable real-time data systems / Nathan Marz, James Warren. Manning, Shelter Island.

Microsoft. (2019). Team Data Science Process Documentation. Available online at https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/, updated on 11/15/2018, checked on 11/16/2018.

Nakajima, S. (2018). [Invited] Quality Assurance of Machine Learning Software. In: 2018 IEEE 7th Global Conference on Consumer Electronics (GCCE). 9–12 pp. 601–604. IEEE, Piscataway, NJ.

Nakamichi, K., Ohashi, K., Namba, I., Yamamoto, R., Aoyama, M., Joeckel, L., Siebert, J., & Heidrich, J. (2020). Requirements-Driven Method to Determine Quality Characteristics and Measurements for Machine Learning Software and Its Evaluation. In: 28th IEEE International Requirements Engineering Conference (RE).

Nistala, P., Nori, K. V., & Reddy, R. (2019). Software Quality Models: A Systematic Mapping Study. ICSSP 2019: 25 May 2019, Montréal, Canada : proceedings*2019 IEEE/ACM International Conference on Software and System Processes.* (pp. 125–134). IEEE.

Poth, A., Meyer, B., Schlicht, P., & Riel, A. (2020). Quality Assurance for Machine Learning – an approach to function and system safeguarding. In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), pp. 22–29. IEEE (uuuu-uuuu).

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). Hidden Technical Debt in Machine Learning Systems. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, pp. 2503–2511.

Shearer, C. (2000). The CRISP-DM Model: The New Blueprint for Data Mining. *Journal of Data Warehousing, 5*, 14–22.

Siebert, J., Joeckel, L., Heidrich, J., Nakamichi, K., Ohashi, K., Namba, I., Yamamoto, R., & Aoyama, M. (2020). Towards Guidelines for Assessing Qualities of Machine Learning Systems. In: Shepperd, M., Brito e Abreu, F., Rodrigues da Silva, A., Pérez-Castillo, R. (eds.) *Quality of Information and Communications Technology, 1266,* pp. 17–31. Springer International Publishing, Cham.

SPEC, D. 92001–01: Künstliche Intelligenz - Life Cycle Prozesse und Qualitätsanforderungen. Teil 1: Qualitäts-Meta-Modell. Beuth Verlag GmbH, Berlin.

Vogelsang, A., & Borg, M. (2019). Requirements Engineering for Machine Learning: Perspectives from Data Scientists. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), pp. 245–251.

Wagner, S., Goeb, A., Heinemann, L., Kläs, M., Lampasona, C., Lochmann, K., Mayr, A., Plösch, R., Seidl, A., Streit, J., et al. (2015). Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology, 62*, 101–123.

Wan, Z., Xia, X., Lo, D., & Murphy, G. C. (2019). How does Machine Learning Change Software Development Practices? *IEEE Transactions on Software Engineering, 1*.
Zhang, D., & Tsai, J. (2002). Machine learning and software engineering. In: Staff, I.C.S. (ed.) 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2002), pp. 22–29. IEEE Computer Society Press, Los Alamitos.
Zhang, J. M., Harman, M., Ma, L., & Liu, Y. (2020) Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering, 1*.
Zhang, X., Yang, Y., Feng, Y., & Chen, Z. (2019). Software Engineering Practice in the Development of Deep Learning Applications.

Julien Siebert Julien Siebert is expert "Artificial Intelligence" in the Data Science department of the Fraunhofer Institute for Experimental Software Engineering IESE in Kaiserslautern. He received a M. Sc. in Artificial Intelligence (Université Henri Poincarré, Nancy, France) and a M. Sc. in Engineering Science (École Supérieure des Sciences de l'Ingénieur de Nancy, ESSTIN, Nancy, France) in 2007, and got his Ph.D. in Computer Science in the field of modeling and simulation of complex systems in 2011 from the Université de Lorraine and INRIA Grand Est (Nancy, France).

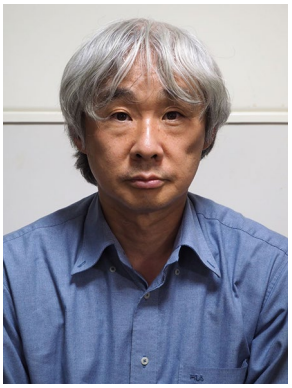
Lisa Jöckel Lisa Jöckel is a researcher in the Data Science department at the Fraunhofer Institute for Experimental Software Engineering IESE. Her work focuses on uncertainty estimation in data-driven software components. Further interests concern quality assurance for software systems containing data-driven components. She received her M. Sc. in Computer Science from the Technical University of Kaiserslautern with a major in data visualization and computer graphics.

Jens Heidrich Jens Heidrich heads the Smart Digital Solutions department at Fraunhofer IESE and is a lecturer at the Technical University of Kaiserslautern. In the area of process management, he is concerned with the continuous improvement of processes with the help of best practices and measurement data. He is a member of various program committees of international conferences and since 2011 he is on the board of the "Software Measurement and Evaluation" group of the German Informatics Society.

Adam Trendowicz Dr. Adam Trendowicz is expert "Data Analytics" in the Data Science department of the Fraunhofer Institute for Experimental Software Engineering IESE in Kaiserslautern. After completing his doctorate on software project effort and risk assessment models at the Technical University of Kaiserslautern, he worked in the field of data science and data-driven business innovation. Dr. Trendowicz currently focuses on data quality and preparation in the context of machine learning as well as on the lean deployment of data-driven innovations based on solutions from the areas of machine learning and artificial intelligence. Dr. Trendowicz is co-founder of the "Data Scientist" training and certification program offered by the Fraunhofer Alliance Big Data and Artificial Intelligence. He has also held several tutorials on business IT alignment, data preparation and analysis, software quality measurement and cost estimation. Finally, he is co-author of several books and numerous publications in international journals and at conferences.

Koji Nakamichi Koji Nakamichi is a company member of Fujitsu Ltd. After he received BS and ME from Yokohama National University, Yokohama, Japan in 1992 and 1994, respectively, he joined FUJITSU LABORATORIES LIMITED. His research area includes software development methodologies and software quality evaluation. He is a member of IPSJ.

Kyoko Ohashi Kyoko Ohashi is a company member of Fujitsu Ltd. After received BS from Tsuda College, Japan, in 1986, she joined FUJITSU LABORATORIES LIMITED. She researched software development methodologies, requirement engineering, User experience, software quality evaluation, and so on. She is a member of IPSJ.



Isao Namba Isao Namba is a company member of Fujitsu Ltd. After received MA from Kyoto University, Japan, in 1989, he joined FUJITSU LABORATORIES LIMITED. He researched natural language processing, information retrieval, software quality evaluation, and so on. He is a member of IPSJ.

Rieko Yamamoto Rieko Yamamoto is a visiting professor in the Department of Software Engineering at Nanzan University and a fellow of Japan Science and Technology Agency. After received BS from Waseda University, Japan, in 1983, she joined FUJITSU LABORATORIES LIMITED. She researched software development methodologies, software development environments, requirement engineering and so on, then moved to JST in 2021. Yamamoto received her PhD in software engineering from Nanzan University. She is a council member of Science Council of Japan and a member of IPSJ and IEEE.

Mikio Aoyama Mikio Aoyama is a professor in the Department of Software Engineering at Nanzan University, Japan. After received MS from Okayama University, Japan, in 1980, he joined Fujitsu Limited, where he was involved in the development of large-scale communications software, and the development and practice of advanced software engineering. From 1986 to 1988, he was visiting scholar at the University of Illinois, USA. In 1995, he joined Niigata Institute of Technology as a professor, then moved to Nanzan University in 2001. His research interests include requirements engineering and software engineering for cloud computing and embedded computing. Aoyama received his PhD in engineering from Tokyo Institute of Technology. He is a member of IPSJ, IEEE, ACM, and ASE.

Authors and Affiliations

Julien Siebert[1] · Lisa Joeckel[1] · Jens Heidrich[1] · Adam Trendowicz[1] · Koji Nakamichi[2] · Kyoko Ohashi[2] · Isao Namba[2] · Rieko Yamamoto[2] · Mikio Aoyama[3]

Lisa Joeckel
lisa.joeckel@iese.fraunhofer.de

Jens Heidrich
jens.heidrich@iese.fraunhofer.de

Adam Trendowicz
adam.trendowicz@iese.fraunhofer.de

Koji Nakamichi
nakamichi@fujitsu.com

Kyoko Ohashi
ohashi.kyoko@fujitsu.com

Isao Namba
namba@fujitsu.com

Rieko Yamamoto
rhd02113@nifty.com

Mikio Aoyama
mikio.aoyama@nifty.com

[1] Fraunhofer IESE, Kaiserslautern, Germany

[2] Fujitsu Laboratories Ltd., Kawasaki, Japan

[3] Nanzan University, Nagoya, Japan