Automated Quality Model Management Using Semantic Technologies

Reinhold Plösch[1][a], Florian Ernst[1] and Matthias Saft[2]
[1]*Business Informatics-Software Engineering, Johannes Kepler University Linz, Austria*
[2]*Siemens AG - Foundational Technologies, Garching, Germany*

Keywords: Quality Models, Semantic Quality Models, Quality Model Tailoring, Query-Based Quality Model Tailoring, AI Based Metric Classification.

Abstract: The starting point for this paper and service for the query-based generation of quality models was the requirement to be able to manage software quality models dynamically, as detailed domain knowledge is usually required for this task. We present new approaches regarding the query-based generation of software quality models and the creation of profiles for quality analyses using the code quality tool *SonarQube*. Furthermore, our support for the automatic assignment of software quality rules to entries of a hierarchical quality model simplifies the maintenance of the models with the help of machine learning models and large language models (HMCN and SciBERT in our case). The resulting findings were evaluated for their practical suitability using expert interviews. The results are promising and show that semantic management of quality models could help spreading the use of quality models, as it considerably reduces the maintenance effort.

1 INTRODUCTION

Software quality models (QMs) are typically used for examining, improving and forecasting the overall quality of a software product (Deissenboeck et al., 2009; Wagner et al., 2015; Wagner et al., 2012a). There is no universal guideline for defining such models. The ISO/IEC 25010 provides a fixed structure, while the Quamoco meta-model (Wagner et al., 2015; Wagner et al., 2012a) offers a more flexible approach that addresses the standard's limitations.

However, as (Wagner et al., 2012a) state, most of the concepts provided by ISO/IEC 25010 are not objective enough to be measurable and to assess the actual quality of a software product. Furthermore, the ISO/IEC 25010 does not state how the individual quality properties should be assessed and how the results of these measurements should be aggregated to get a final result (Wagner et al., 2012a). The Quamoco meta-model has been selected and, therefore, all further steps are going to be performed using this specific QM.

As large QMs are difficult to maintain due to the required knowledge of the model's specific application domain, a tool for effortless tailoring to support the QM expert is needed. The models we developed and that are applied for embedded system development typically contain 1000+ metrics and rules. With our approach, a QM expert should be able to automatically generate new quality models based on already stored QMs, considering the specific quality requirements of a project.

As a novelty of our approach, we do not only consider metrics and rules as provided by SonarQube, but also so-called *dynamic project related data*, which can be seen as usages of certain metrics within a predefined set of projects in SonarQube. This data is called dynamic as it changes relatively fast compared to more static-like data in SonarQube-like metrics and rules. Examples of these dynamic data would be the number of rule violations in relation to all rules of the same type of severeness, the number of violations per 100 lines of code, and the won't fix and false positive rate. Additionally, the relevancy of a metric or rule could be measured by looking at a *fixing factor*, to identify whether violations of this metric are increasing, decreasing or relatively steady.

Furthermore, it should also be feasible to generate so-called *Quality profiles* for SonarQube based on the aspects described above. A quality profile can be seen as a collection of metrics and rules which are grouped into a profile. These profiles are then associated with projects.

In order to be able to answer these queries for QM tailoring, the static-like, dynamic and QM-related

[a] https://orcid.org/0000-0002-4659-2385

Plösch, R., Ernst, F. and Saft, M.
Automated Quality Model Management Using Semantic Technologies.
DOI: 10.5220/0013502900003964
In *Proceedings of the 20th International Conference on Software Technologies (ICSOFT 2025)*, pages 223-232
ISBN: 978-989-758-757-3; ISSN: 2184-2833

data need to be stored to answer queries fast. It was decided that QM-related aspects should also be stored in RDF along with other software quality aspects.

With new SonarQube updates and new plugins for new languages, the number of new rules and metrics which need to be assigned manually to an already-existent QM increases. Therefore, a machine-learning (ML) model should be used for the automatic classification of these new metrics into the hierarchical structure of a QM. Furthermore, the developed software component should also be able to manage the ML model and semi-automatically re-train the model if necessary.

Based on this context, the following RQs were defined:

- RQ 1: How suitable are query-based quality model tools for tailoring and generating software quality models?
- RQ 2: Is the automatic categorization of software metrics based on machine learning algorithms suitable?
  - RQ 2.1: How suitable are machine learning algorithms according to typical evaluation metrics based on the available data?
  - RQ 2.2: How suitable is the automatic categorization from a quality model expert's view?

RQ 1 and RQ 2.2 will be assessed by using survey guided interviews of software quality model experts with a resulting analysis. For RQ 2.1, an experiment of ML models that are found alongside the hierarchical text classification is going to be performed and evaluated.

The reminder of this paper is structured as follows. First, relevant literature regarding this topic is reviewed in Section 2. Afterwards, the concepts behind the proposed tailoring approach are described in detail in Section 3. Section 4 provides an overview of the QM tailoring service's components. Similar to the previous section, Section 5, highlights the ML service's core modules. Both service approaches are then reviewed and evaluated in Section 6. Finally, Section 7 concludes this paper.

2 RELATED WORK

Pressman (Pressman, 2010) gives the following working definition for software quality: "*An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.*" (Pressman, 2010, p. 400).

(Pressman, 2010) also summarizes other software quality concepts. For example, (Garvin, 1987) developed a multidimensional approach regarding quality in general, and according to (Pressman, 2010), these eight dimensions are also capable of analyzing software quality. The dimensions are *performance quality*, *feature quality*, *reliability*, *conformance*, *durability*, *serviceability*, *aesthetics* and *perceptions*. One problem with these dimensions is that most of them can only be measured subjectively. The "hard" factors – measurable factors – can also be divided into two sets, namely into factors that can be directly (e.g., software defects) or indirectly (e.g., usability) measured (Pressman, 2010).

McCall (McCall et al., 1977) groups factors into product activities, namely *product revision* (change of a product), *product transition* (adaption to new domains), and *product operation* (the operation) (McCall et al., 1977). E.g., within the activity *product revision*, which is related to software changes, the factors with the corresponding questions are as follows: *maintainability* (Is the software fixable?), *flexibility* (Is the software changeable?) and *testability* (Is the software testable?) (McCall et al., 1977). Again, one drawback of using these factors is that, according to (Pressman, 2010), most of the proposed factors can only be assessed indirectly.

Nevertheless, these factors provide a solid base for evaluating the quality of a software product (Pressman, 2010). In software QMs, per definition, these software quality attributes are typically brought into a hierarchical order. This decomposition has no universal meta-model, so it is performed without any pattern on these quality attributes (Deissenboeck et al., 2009). Therefore, according to (Deissenboeck et al., 2009), any further decompositions on large models are difficult to understand and might lead to repetition, as quality attributes might overlap.

While there is still ongoing work in how to structure the abstract term *Software Quality*, there are no approaches known from the scientific literature that support systematic tailoring of quality models to the quality requirements of a software project.

3 QUALITY MODEL TAILORING CONCEPTS

In this section we describe how we relate the quality model concepts to semantic models and we describe in more details which static and dynamic data can be used for quality model tailoring.

3.1 QMs and Knowledge Graphs

QMs represent important knowledge on software quality and knowledge graphs are used for representing them in a semantic way (see Figure 1 and Figure 2). The first part of the knowledge graph contains information about the quality model hierarchies. The resource that represents a QM is called *QualityModel*. For each QM, an identifier and several hierarchies are stored. Each hierarchy that belongs to a QM, represented through the class *QualityModelHierarchy*, also consists of an identifier and property that connects the hierarchy to its root element through the property *hierarchyId*. An entry of a QM hierarchy in which no distinction between leaf and inner nodes is made, is defined by the class *HierarchyItem*, which can also contain zero or more hierarchy items that are the particular node's child nodes. Additionally, with each QM hierarchy item, a name, a description, and an identifier is stored. Besides these directly attached literals, connections to the classes *QualityProperty*, *Entity*, *Rule*, and *Impact* exist.

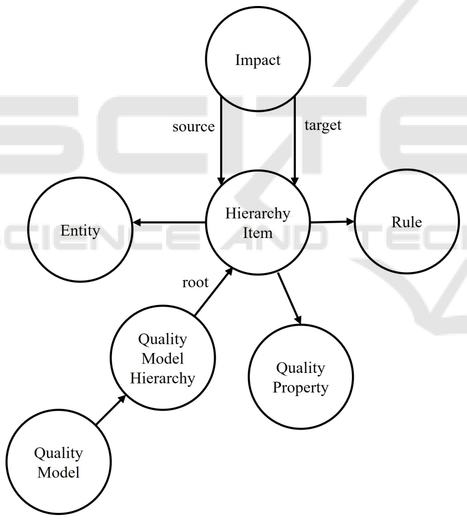

Figure 1: Fragment of the knowledge graph for QMs showing the model-related relationships.

A quality property expressed by the class *QualityProperty* contains a property name combined with a value and optional a so-called *extra property* which provides additional information for a quality property like coverage, importance, trustworthiness, and effort.

The next class represents an *entity* and is also named after it. In the original Quamoco QM, each factor has an associated entity that describes a certain fragment of a software product or a resource involved during its lifetime (Wagner et al., 2012b). Examples for entities are *class*, *interface*, or *method*. In the data model, the name of an entity is associated with the *rdfs:label* property. Optionally, it is possible to connect each hierarchy item with various rules expressed with the *Rule* class.

Impacts are represented by their eponymous class. Each impact denotes an impact of one hierarchy's entry into another hierarchy. The connection is made via a source and a target property that both reference hierarchy items. As an example, the factor *complexity@Class* typically has a negative impact on the quality factor *maintainability*. Technically, an impact's effect is stored as a number which might either be -1 (negative), 0 (no effect), or 1 (positive). Additionally, each impact has also an optional description. The overview of the mentioned resources and their relationships are shown in Figure 1.

3.2 Static Data

In this paper, static data refers to rule-related data for rules that do not regularly change. In contrast, dynamic data, described in the next section, contains information about the usage of rules in actual projects with their related issues over time. As already mentioned before, metrics and rules are central elements of static data. SonarQube (depending on the configuration) provides e.g. 1.000+ metrics and rules for the programming language Java.

It should be noted that in RDFS, it is not possible to enforce that, e.g., each rule must have an associated SonarQube repository. Nevertheless, with Shapes Constraint Language (SHACL) shapes (Knublauch and Kontokostas, 2017), it is possible to validate knowledge graphs and hence, to ensure that the constraint mentioned above is fulfilled. Otherwise, the knowledge graph is not valid. Additionally, SHACL also provides so-called SHACL rules (Knublauch et al., 2017) that are used for inferencing new triples

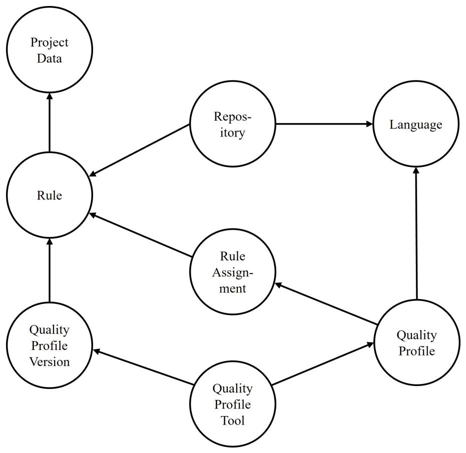

Figure 2: Fragment of the knowledge graph for QMs showing the rule related connections of static and dynamic data.

from the provided data.

Besides the repository, a rule is also associated with various quality profiles using the *RuleAssignment* class. However, not every rule must be associated with a quality profile. Furthermore, a severity and rule type is set in every rule assignment, which overwrites the default severity and rule type from a rule's default values. A class of type *QualityProfileTool* is used to represent the rules' source tool. Quality profiles are a concept of SonarQube and might, therefore, not be available in other tools, which is also the case for repositories. Each quality profile and also each repository is associated with a certain language class that represents the corresponding programming language of both types.

A version of a software quality tool is described with the class *QualityProfileVersion*, which is associated with a quality profile tool and contains references to those rules that are available in the specific version. Rules are represented through the eponymous classes. For each individual rule, an identifier, a default severity, tags, a default rule type, default and actual remediation costs, an HTML description, a status, the name, and the creation time are stored. The remediation costs are represented using one class of the remediation cost's hierarchy that is modeled using multiple inheritance. The base class *RemediationCosts* contains a time unit representing the unit of the remediation cost's time. Remediation costs can either be a single constant, represented by the *ConstantRemediationCosts* class. This linear function only consists of a coefficient (class *LinearRemediationCosts*), or finally, a combination of both types, i.e., a linear function with an offset provided by a constant (class *LinearRemediationCostsWithOffset*).

3.3 Dynamic Data

The idea behind the *dynamic aspects* is to collect usage data of rules' issues in projects within SonarQube which then can be further used for tailoring QMs. One example could be to exclude rules where the ratio of related issues that are marked as won't fix to the total number of issues of the corresponding rule is larger than $x\%$. This could make sense as it is an indicator that the related rule is not suitable for a specific application domain.

These measures are stored in a time series database, and the latest values are persisted in the RDF storage to imitate a materialized view and to speed up querying. Together with a software quality expert from academia, the following dynamic aspects were identified:

- Open issues in total and as a ratio with all issues of the same severity
- Total number of issues that are marked as won't fix and as a ratio based on the total number of issues of the specific metric
- Total number of issues that are marked as false positive and as a ratio based on the total number of issues of the specific metric
- Usages of a rule as absolute value and as ratio based on the pre-configured list of projects that should be taken into consideration
- Issues per 100 lines of code (LOC)
- *Fixing* factor, which indicates whether the number of open issues is increasing, decreasing, or steady

For each defined metric, the operators $<$, $>$, $<=$, $>=$, $==$, and $!=$ are supported. As mentioned, these measures are calculated for every rule available in SonarQube. In the configuration section of this service, it is possible to define several project sets that will be used for calculating the previously defined measures. This makes sense, as quality managers want to use the data of similar projects for tailoring of the QMs.

The class *ProjectData* represents a current dynamic aspect entry for a single project set identified by the data property *projectDataIdx*, i.e., an entity is created for every rule and project set combination. This property is simply the project index set from this service's configuration part. All other properties (the absolute value or ratio) are computed beforehand and stored in the entity of class *ProjectData*.

3.4 Quality Profile Generation and Rule Export

In this section, the process of generating quality profiles which can further be directly imported into SonarQube without any additional changes is explained. Moreover, the export of rules is also presented. Both operations are performed using the service's REST interface.

Currently, the possible query parameters for the rule export are entries of a quality model, i.e., hierarchy items including their child items on which certain rules are associated can be used, including the entity tag separated by the @ tag, so if no tag is used, all hierarchy items regardless their corresponding entity will be taken into consideration. The next query parameter filters according to a given list of severity types on which the rules should be filtered. Next, it is possible also to exclude rules by their corresponding identifier.

Additional, so-called extra values can be attached with the — sign. It is also possible to include more quality properties, separated by a comma. The following query parameter allows the rule selection by one or several quality profile tools, presented in the format `tool name:tool version`. It should be noted that the tool version is optional in this context. A comma again separates the different tools. Next, it is possible to filter rules by a provided programming language. Finally, the last query parameter allows users to filter rules by a quality profile's name. All of the mentioned query parameters are optional. Hence, if no parameter is defined, all rules will be used.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX sqm: <http://siemens.com/sqm#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?repository_key ?rule_key ?severity
WHERE {
    ?language sqm:languageId "{language}".
    ?language a sqm:Language.
    ?repo sqm:language ?language.
    ?repo a sqm:Repository.

    ?repo sqm:repositoryKey ?repository_key.
    ?rule sqm:fromRepository ?repo.
    ?rule a sqm:Rule.
    ?rule sqm:ruleId ?rule_key.
    ?rule sqm:defaultSeverity ?severity.
    ?tool a sqm:QualityProfileTool;
          rdfs:label "Sonarqube";
          sqm:toolVersion ?toolVersion.
    ?toolVersion sqm:containsRule ?rule.

    {{where_clause}}
}
```

Figure 3: Query fragment for exporting quality profiles from the knowledge graph.

The rules' query is based on the SPARQL fragment presented in figure 3. This query selects the repository keys and the rule identifiers of all rules that match the WHERE clause. Before the query is issued to the SPARQL endpoint, the placeholder `{{where_clause}}` is replaced with the dynamically built query part that is based on the query parameters presented before. This allows the query to be built dynamically and to allow the variation of parameters. After the query is issued to the server and the result has been received, the given tuples are converted into the data transfer object class `RuleExportEntry`, which is then serialized into the corresponding JSON format using the pydantic library, which is included in FastAPI.

Based on this general rule query model, typical tasks like generating a new quality profile or generating a new quality model can be executed automatically. This allows to regenerate these quality profiles or quality models after changes of a SonarQube installation on the fly. Changes to the installation could result in new or changed metrics and rules.

4 QUALITY MODEL TAILORING SERVICE

The quality model tailoring service imports software quality rules from SonarQube and quality models from Quamoco XML files. Besides the incremental import part, the service also supports the required query-based rule, quality profile, and quality model export features as described in the previous section. In the following paragraphs, selected modules and components are described.

SQMDB Manager Component: The Software Quality Model Database manager component, which is represented through a regular Python class, is used to separate the business logic from the REST interface by abstracting all operations that are callable from the REST interface.

Repository Module: The repository module currently only consists of the `RDFRepository` class. It could be argued that within this module, also all dynamic aspects-related storage classes should be located. Nevertheless, as the `InfluxDBStorage` is only used within the dynamic aspects importer, it was decided that it should be placed within the dynamic aspects module.

Import Module: The import module contains various components for importing quality rules from different sources; currently SonarQube and cppcheck are supported. Additionally, this module also contains classes for importing quality model hierarchies. Currently, the hierarchy import from Quamoco XML files is supported.

Export Module: Within the export module, Python functions containing the corresponding logic are grouped into adequate components respectively Python files, namely `QuamviewExporter`, `RuleExporter` and `SonarQubeExporter`. Each component computes the result classes of the particular export use case. For example, the `SonarQubeExporter` consists of a single Python function, called `export_quality_profile` whose output is an XML document containing the quality profile, which can then be imported into a SonarQube instance without any additional modification.

Dynamic Aspects Module: The dynamic aspects module contains a base class for storages of dynamic aspects and a Python file containing all relevant model classes that correspond to dynamic data besides the `DynamicAspectsImporter` and `InfluxDBStorage` components. Within the `DynamicAspectsImporter`, the data from the configured SonarQube instance is queried, transformed, and finally stored in the corresponding storage.

5 MACHINE LEARNING SERVICE

The machine learning service is a separate Python-based service that assigns rules to corresponding QM hierarchy entries. The source of the rules does not matter, as only a unique identifier of a rule, its name, and description is transferred to this service using the provided REST API. Currently, this service supports the Multi-label Classification Network (HMCN) (Wehrmann et al., 2018) and the SciBERT model (Beltagy et al., 2019).

Database Module: As the imported or predicted rules need to get persisted, a SQLite DB has been chosen as a simple local database for tabular data. The database module contains the base class for all database operations, namely `BaseDAL`. Currently, only a data access layer for the SQLite database, namely `SQLDAL`, is implemented. However, in the future, if a new data storage is needed, only the abstract base class `BaseDAL` needs to get extended. In this module, also an importer is implemented, which provides helper functions for importing training and test data from pandas' data frames.

Model Module: The module for the models contains two major components, namely the classifier component, which contains all classifiers that were implemented for this service, and the dataset component, which contains a PyTorch-based dataset together with helper functions that are needed for the training process as well as for the construction of the dataset that is used for prediction.

The basic functionality of a classifier is implemented in the abstract base class `BaseClassifier`, which also contains data classes for the interim results. Each classifier that should be supported in this service has to implement this abstract base class, which is later used in the dependency injection part to map the actual implementation to this abstract base class. As mentioned before, a SciBERT and HMCN implementation are supported in this service.

Configuration Component: The classes for the environment-config serialization package are implemented in the configuration component.

With the help of the `ModelType` enum, the user is able to choose between the HMCN and the SciBERT model. For each of these two model types, a separate class is created. The serialization library allows to have nested configuration classes while only having environment variables with flat values. This feature is used as follows: the root class `Config` captures the model type, which is the enum mentioned before. Additionally, the configuration classes for both models are included using a group from the environ package. Both model configurations are marked as optional, whilst the enum class is set to mandatory and selects which model should be used in this service.

Migration Component: The migration component contains the code that is needed in order to perform the database migrations using alembic. Therefore, this component mainly consists of two parts that are required by the library, namely, the versions which represent the actual migrations and the environment configuration that can be used to fine-tune the migration process. This command creates, based on the defined sqlalchemy models, the corresponding migrations to create and alter the database tables accordingly. In addition, it is also possible to downgrade a migration.

Routing Component: Within the routing component, all routes for the web API are defined. These routes are then used in FastAPI to provide the corresponding REST interface. Each route is again annotated with an API version which is currently set to 1. It was decided to use API versioning to provide better backward compatibility in the future when this service is used in production. To decouple the web interface from the actual implementations, the dependency injection library pythondi is used.

Preprocessing Component: The preprocessing component consists of two preprocessing services, one for SciBERT named `BERTPreprocessingService` and the class `PreprocessingService` for the HMCN model.

Both services extend the abstract base class `BasePreprocessingService`, which acts like an interface and is used in the dependency injection library for binding the correct preprocessing service based on the model used according to the service's configuration.

The `BERTPreprocessingService` uses the class `BertTokenizerFast` from the transformers library to prepare the input text into the format that is required from BERT, whilst the preprocessing service for the HMCN model uses a "classical" preprocessing pipeline starting with the tokenization and the removal of stop words, i.e., words that often occur and do not deliver any useful information, then followed by a simple preprocessing pipeline provided by the natural language library gensim. Then, the words are lemmatized using the wordnet lemmatizer from the NLTK library. Finally, the individual tokens are mapped to the corresponding indices of the word embedding matrix.

Main component: The main component is the main entry point of this service and first loads the configuration and, based on the provided configuration, the final ML model. If no model is currently saved, the main component will initiate the model learning process by loading the provided test and training data from the given CSV files shipped within this service. Afterward, the FastAPI library is initialized, and the corresponding routes are registered.

6 EVALUATION

As mentioned before, in this section, the general procedure of the evaluation that was performed to evaluate the usefulness of both services from a software quality expert's viewpoint is elaborated. With regard to the experience level of each individual software quality expert, it was ensured that experts with experience spanning several years from both academia and industry related software quality products were selected.

6.1 Questionnaire and Participants

The overall evaluation process consists of two parts, namely an interactive presentation with the interviewee, which demonstrates the usage of both services as well as pre-defined use cases, and a questionnaire that is built upon this presentation which is further used for the evaluation.

The idea behind this whole process is to provide the interviewee with an interactive presentation rather than just offering an explanatory video with the questionnaire due to the highly abstract topic.

Thus, the interviewee is enabled to ask specific questions in real time and possible questions or misunderstandings can be clarified right away. In general, the interviewee is asked to rate the usefulness of certain parameters and use cases according to a Likert scale. Additionally, for almost each question group, the interviewee is also provided with a free text field where additional information can be submitted.

The questionnaire starts with general questions, such as the interviewee's age, years of experience in software development, the industry of their current job and the size of their company. The next part is the general RDF data model that was developed, which consists of the software quality rules, the dynamic data entries and the software QMs.

Here, the interviewees are asked to rank the suitability of this data model with respect to completeness and adequacy of the distinct data items with the possibility to optionally enter additional remarks to the data model. After this part, the possible query parameters for the rule and quality profile export are shown, followed by the use cases for the rule and quality profile export. The next section covers the QM tailoring part, starting with the possible query parameters followed by the corresponding use cases.

Since the use cases of both prototypical services are rather abstract and the future users of these services belong to a small group of software engineers that are specialised in software quality engineering, this evaluation's participants were selected accordingly. Eight software quality experts participated in the final evaluation. Some more detail on the study population will be given in the next section.

6.2 Interview Results

In this section, the rule and quality profile-related items of the questionnaire are explained.

The parameters of the rule and quality profile export are split into three categories that are displayed in Figure 4, Figure 5 and Figure 6.

The first category shows the importance of the parameters that are supported in the rule exporting process. Then, in the second category, results regarding the quality profile export parameters are evaluated. Afterwards, the importance of each dynamic aspect is presented in the last category.

In the following sections, the individual parameters are presented together with their explanation and the final results, starting with the rule export parameters.

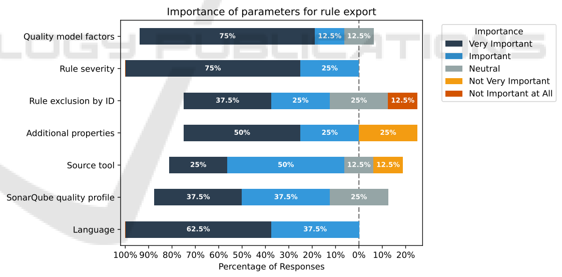

Figure 4: Importance of parameters for rule export.

As depicted in Figure 4, the first parameter is used to filter rules according to their associated factors of the QM, which are separated by a comma. In general, almost all responses rate this parameter as very important or important. The next parameter represents the severity of a rule. Here, it is also possible to filter with multiple severity levels, which are separated by a comma. As before, 75% of the responses rate this parameter as very important and the rest as important.

In contrast to these two parameters, only 37.5% of responses rate the rule exclusion by ID, where rules can explicitly be excluded by their given ID as very important and 25% as important. Nevertheless, 12.5% rate this feature as not important at all.

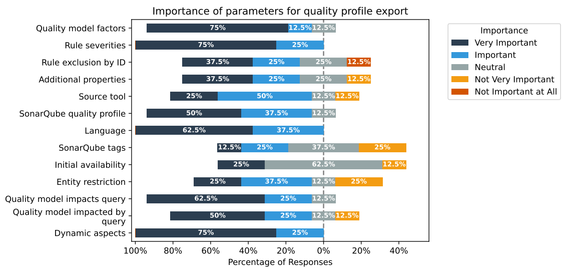

Figure 5: Importance of parameters for quality profile export.

Regarding the additional properties that are the remainder of the Quamoco QMs, 75% of the interviewees rated the importance as very important or important. The rest, 2 participants, rated this feature as not very important. For the source tool, i.e. the tool from which the rules are imported, it is possible to indicate several tools together with an explicit version.

The tools are again separated with a comma, and the version number is separated with a ":" after the corresponding tool's name. E.g., the tool SonarQube with version 10 would be as follows: "SonarQube:10". However, 50% rated this parameter as important and 25% as very important. One participant rated this parameter as neutral or as not very important in each case.

The penultimate rule export parameter is the source quality profile from SonarQube which allows to user to specify a quality profile from which the corresponding rules should be exported. Here, 37.5% each rated this export parameter as very important or important. The rest rated the importance as neutral.

The last parameter, the programming language of the rule, is used to restrict the rules to certain programming languages. All participants rated this parameter as very important or important. To summarize, all provided parameters were rated mostly positively by the interviewees.

In Figure 5, the parameters for exporting quality profiles for SonarQube are shown. As the first seven parameters are the same and their results are roughly the same, they are skipped in this part of the questionnaire's evaluation.

The first new parameter is the possibility to filter rules for quality profiles using tags that are provided by SonarQube. Again, a comma is used to separate these values. 37% of the participants rated this parameter positive. The exact number rated this parameter as neutral and 25% as not very important. For the initial availability, which is used to filter rules by the date they were added in SonarQube, only 25% rated this feature as very important. More than 50%, however, rated this neutral, and one person rated it as not very important.

The entity restriction which allows the user of this service to restrict rules by their associated QM entries' entity type like @Source Code, @Statement, etc. This feature was rated positively by 62.5% of the responses. One interviewee rated it as neutral, and the rest, 25%, as not very important. The next parameter is the QM impacts query which is used to select rules where an associated QM entry impacts another QM entry.

It is also possible to filter according to the type of impact, which is either positive (+) or negative (-). The plus sign can also be omitted and is used internally as default when no sign is provided. 62.5% of the responses rated this parameter as very important and 25% as important. The rest, 25%, rated it as neutral.

For the next parameter, which is the inverse of the QM impacts query, namely the QM impacted by query, the responses are almost the same with the exception that 50% rated it as very important and one interviewee as not very important. The parameter list for the export of quality profiles is concluded by the dynamic aspects whose detailed query options are explained and evaluated in the next paragraph. Three out of four recipients rated the usage of dynamic aspects for quality profile export as very important, and the remainder as important.

To summarise the overall importance of the parameters, almost all provided parameters were categorised as very important or important except for SonarQube tags and the initial availability of rules.

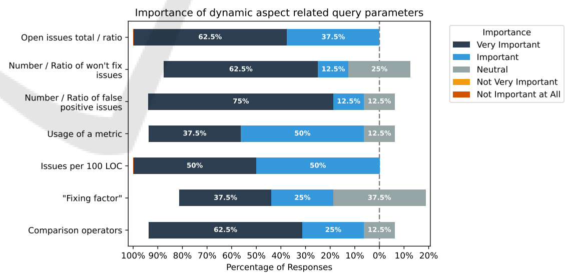

Figure 6: Importance of dynamic aspect related query parameters.

In total, six dynamic aspects were implemented and evaluated together with the provided query operators using the questionnaire, with the results displayed in Figure 6. The first dynamic aspect is the number of open issues as an absolute value, respectively as a ratio to the number of open issues with the same severity. Here, all participants rated the query option positively, with 62.5% as very important and the rest as important.

Next, the total number of issues marked as won't fix and the ratio variant with the absolute number of issues marked as won't fix compared to the total number of the corresponding rule's open issues. Most of the interviewees ranked this query parameter as very important or important. However, 25% ranked it as neutral.

The next parameter is almost the same as before; nevertheless, focusing on the total number of issues marked as false positives and again as a ratio. 75% rated this parameter as very important; in each case, one interviewee ranked it as important and neutral.

Whilst previous parameters focus on the number and types of issues of a certain rule, the next parameter focuses on the usage of a rule in a list of predefined projects. A usage of 60 % would mean that a certain rule is used in 60% of pre-defined projects. Additionally, it is also possible to define multiple project lists. Almost all interviewees rated this parameter positively with 37.5% as very important, 50% as important and one result rated as neutral. In the next dynamic aspect, the number of issues per 100 lines of code (LOC) is used to identify rules where the ratio of open issues per 100 LOC matches the provided query criteria. All interviewees rated this aspect as very important respectively important with an equal share. The last aspect is called "Fixing factor" and denotes whether the number of open issues is currently rising, steady or declining. With 62.5% of the interviewees finding this aspect as important and 25 % as important, it is mostly rated positively. Nevertheless, 37.5 % rated it as neutral. The final question was about the provided comparison operators, namely $<, >, <=, >=, != $ and $==$. Here, 62.5% rated these as very important and 25 % as important. For the rest, one interviewee rated it as neutral.

To summarize the evaluation of the dynamic aspects: overall, the aspects are rated mostly without any exception as very important and important without any responses rating any aspect respectively the comparison operators as unimportant.

In this last section of the interview results, the results regarding the suitability of the automatic rule association using the proposed ML service are evaluated. For the evaluation, the best-performing model, the SciBERT model, was chosen.

All three evaluation measures, precision, recall and the f1-score were presented to provide the interviewees with the same background information. Additionally, the classification process's best and worst performing target classes were shown to give the interviewees an intuition of a typical task the ML model will perform. Overall, 37.5% each ranked the suitability as highly suitable and neutral. 25% rated it as suitable, as shown in Figure 7.

However, during the interviews, a great interest

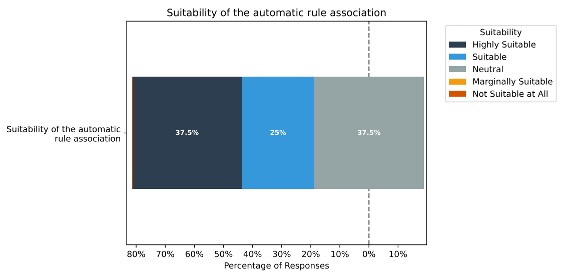

Figure 7: Suitability of the automatic rule association.

in the possible usage of this approach was shown. Nevertheless, real-world application is still missing and needs further evaluation to see how such a model would perform on real data in the future.

6.3 Threats to Validity

The statistical conclusion validity can not be analysed since no statistical hypothesis testing was performed to answer the research questions. No threats regarding the proposed groups by (Shadish et al., 2002) could be found for the internal validity. However, regarding the construct validity, it could be argued that a threat regarding the so-called *inadequate explication of constructs* by (Shadish et al., 2002) might occur since there could be a differentiation between the use cases and parameters presented and the actual usefulness of this service.

Nevertheless, the participants expressed the usefulness of both services both textually using the free text options and verbally during the interviews. Regarding the external validity of these experiments, a possible threat would be concerning the interaction of the observed relationship with the setting, i.e., does the effect change when the setting changes, which would be, in the case of this study, the participants. As the questionnaire was specially designed for software quality experts, this threat is minimised since this is also the target group. The threat is also minimised by choosing software quality experts from different backgrounds, i.e. various industries and academia and various experience levels.

7 CONCLUSION AND FUTURE WORK

The research questions RQ 1 and RQ 2.2 were answered using the questionnaire. Regarding the suitability of query-based QM, which was asked in RQ 1, the experts ranked the proposed use cases of the QM tailoring tool as mostly important, with no use case being entirely ranked as unimportant. Therefore, the developed approach, which uses a service for tailoring QMs and exporting quality profiles for SonarQube, can be considered suitable.

Regarding RQ 2.2 where the suitability of the automatic rule association of rules to QMs from a software quality expert's perspective should be examined, the questionnaire's results and the verbal feedback from the interviewees indicate that a fully automatic rule association would not be acceptable. This is due to the fact that some interviewees work in highly regulated environments where misclassifications must be avoided at all costs. However, the interviewees agreed that this ML service would be appropriate for generating an initial draft, which would then require explicit review by a software quality engineer.

To summarise the overall results of the evaluation, with some exceptions the participants of the survey rated almost all use-cases of the QM tailoring and quality profile export service as useful, as expected. In addition, the participants showed great interest in the concept of the dynamic aspects where dynamic project-related data from SonarQube is used to tailor QMs and generate issue-specific quality profiles for SonarQube which are used for further analysis.

Special interest was shown regarding the "fixing factor". It was suggested to use this metric to develop a traffic light system in a QM viewer respectively editor where those traffic lights indicate on sub-hierarchies of the corresponding QM whether the number of open issues is currently rising, declining or steady.

For the automatic rule classification using the adapted and fine-tuned ML model from the literature research and the experiments, the participants of this evaluation where sceptical regarding the usage in a fully-automated real world scenario due to the restricted environment as a participant remarked. However, the participants overall agreed that it would be beneficial to the software quality management process, if the ML service is not used totally automatic but rather in a semi-automatic way, i.e., the classification results of this service are not used as a final classification. Instead, these results are used as a suggestion for the software quality expert who then marks the classification manually.

Future work based on both developed services could include more practical evaluation and usage on real world projects in the context of larger companies. Especially for the ML service, the classification in a real world setting needs to be evaluated further.

REFERENCES

Beltagy, I., Lo, K., and Cohan, A. (2019). SciBERT: A Pretrained Language Model for Scientific Text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3615–3620, Hong Kong, China. Association for Computational Linguistics.

Deissenboeck, F., Juergens, E., Lochmann, K., and Wagner, S. (2009). Software quality models: Purposes, usage scenarios and requirements. In *2009 ICSE Workshop on Software Quality*, pages 9–14.

Garvin, D. A. (1987). Competing on the Eight Dimensions of Quality. *Harvard Business Review*. Section: Consumer behavior.

Knublauch, H., Allemang, D., and Steyskal, S. (2017). SHACL Advanced Features.

Knublauch, H. and Kontokostas, D. (2017). Shapes Constraint Language (SHACL).

McCall, J. A., Richards, P. K., and Walters, G. F. (1977). Factors in software quality. volume i. concepts and definitions of software quality. Technical report, GENERAL ELECTRIC CO SUNNYVALE CA.

Pressman, R. S. (2010). *Software engineering: a practitioner's approach*. McGraw-Hill Higher Education, New York, 7th ed edition.

Shadish, W. R., Cook, T. D., and Campbell, D. T. (2002). Experimental and quasi-experimental designs for generalized causal inference. *Experimental and quasi-experimental designs for generalized causal inference.*, pages xxi, 623–xxi, 623. Place: Boston, MA, US Publisher: Houghton, Mifflin and Company.

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

Wagner, S., Lochmann, K., Heinemann, L., Kläs, M., Trendowicz, A., Plösch, R., Seidi, A., Goeb, A., and Streit, J. (2012a). The quamoco product quality modelling and assessment approach. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1133–1142.

Wagner, S., Lochmann, K., Winter, S., Deissenböck, E. J., Herrmansdörfer, M., Heinemann, L., Kläs, M., Trendowicz, A., Heidrich, J., Plösch, R., Göb, A., Körner, C., Schoder, K., Streit, J., and Schubert, C. (2012b). The quamoco quality meta-model. Report.

Wehrmann, J., Cerri, R., and Barros, R. (2018). Hierarchical Multi-Label Classification Networks. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5075–5084. PMLR.