Abstract—The quality of graphical software or business process models is influenced by several aspects such as correctness of the formal syntax, understandability or compliance to existing rules. Motivated by a standardised software quality model, we discuss characteristics and subcharacteristics of model quality and suggest measures for those quality (sub)characteristics. Also, we extended SonarQube, a well-known tool for aggregating and visualising different measures for software quality such that it can now be used with repositories of business process models as well. This allows assessing the quality of a collection of models in the same way that is already well-established for assessing the quality of software code. Given the fact that models are early software development artifacts (and can even be executable and thus become a part of a software product), such a quality control can lead to the detection of possible problems in the early phases of the software development process.

I. INTRODUCTION

Software and business process models are nowadays an important artifact in the software development process. Very frequently, they are used for discussing and documenting requirements and for designing the architecture of a software system. It is also possible to generate parts of the software code or test cases directly from the models. The models can even be directly executable. For example, a BPMN model could be executed in a BPMN engine. This way, the model itself becomes a part of the software.

It follows, that the same care should be taken for assessing the quality of such models as it is the case for assessing software code quality. In this article, we will show how an established framework for aggregating and visualising reports for software quality can be extended to show information about the quality of graphical models as well. This way, it is possible to monitor the state and progress of the quality in a software project - starting in the early phases before a single line of code has been written.

The focus of this article is not to develop new methods for measuring model quality but to make the results of existing tools available in an aggregated view. For this purpose, we make use of existing quality checks for different aspects of model quality. Such quality checks are already implemented in several modelling tools. For example, Argo/UML [1] defines so-called critics as a mechanism to inform the modeler about typical problems in a UML model. Similar approaches are applied for example in the modelling tool bflow* Toolbox [2], by the EMF Refactor tool [3] or by a graph-based solution described by Mens et al. [4].

It is our aim to summarise and categorise the results delivered by existing tools in several views. These views should help a reviewer to get a first impression on the quality of a collection of models. Furthermore, the reviewer will be enabled to “drill down” into the problems by locating frequently occurring defects or finding the parts of the model repository that contain the largest number of problems.

The rest of this paper is structured as follows: In Sect. II-A, we discuss existing software quality models. The structure of those quality models is used to suggest a similar quality model for software and business process models. This quality model suggests a categorisation of quality characteristics and subcharacteristics in Sect. II-B and II-C. In Sect. II-D we discuss measures for those quality (sub)characteristics. Sect. III is dedicated to our tool. It gives an overview about its architecture (Sect. III-A) and lists the quality measures considered by our tool (Sect. III-B). In Sect. III-C we show different views for visualising the quality of a collection of models.

II. QUALITY MODELS

A. Software Quality Models

ISO 9126, the standard for the definition and evaluation of source code quality, decomposes the term “software quality” into six quality characteristics (Functionality, Reliability, Usability, Efficiency, Maintainability and Portability). Each quality characteristic is further decomposed into quality subcharacteristics. For example, the quality characteristic Usability is decomposed into Understandability, Learnability, Operability, Attractiveness and User Compliance. Finally, the standard assumes that the subcharacteristics can be measured by quality metrics that calculate (numerical) values for a subcharacteristic. These decomposition results in three hierarchical levels as illustrated in Figure 1.

Although ISO/IEC 9126 has been replaced by ISO/IEC 25000 in 2005, the basic approach did not change. The current version ISO/IEC 25010 (issued in 2011) defines eight quality characteristics instead of the former six. Also, some of the quality characteristics have been renamed, e.g., ‘Efficiency’ to ‘Performance efficiency’. Furthermore, some subcharacteristics have been added.

The quality (sub)characteristics can either be measured by automated tools or manually by qualified persons. One of the purposes of the measures is to support the software development process by finding and fixing errors, bad coding style, indicators of performance problems or security issues. This will finally increase the software quality. Approved manual methods to analyse and rate aspects of software quality are...
code reviews and explorative tests. Though these are more expensive than automated tools, they can better evaluate quality characteristics like “Maintainability”. E.g., for a machine, there is no difference whether a variable has been named “x” or “firstOfStack”. For a human it is. Even if a tool is clever enough to know that “speaking” variable names are more comprehensible than short ones, it cannot offer advice because of its limited knowledge about the context and the business domain. “Usability” is another example for a quality attribute which should rather be measured by a human reviewer instead of a tool.

However, the measurement of several other quality characteristics defined by ISO/IEC 9126 can be supported by automated tools very well. Well-known tools for this purpose include FindBugs, PMD or CheckStyle which are based on static code analysis. Other well-established ones are unit testing frameworks or profilers. Ayewah et al. [5] notice that tools like FindBugs “[...] can find important defects in software.” They can mark possible defects as warnings or errors. Afterwards, code reviewers can decide how critical they are and balance the cost of fixing against severity. Software quality will be consequently monitored and improved which helps to save costs [6].

All of these concepts can be categorised into two groups, procedures operating on compiled (POCC) or non-compiled code (PONCC). While tools supporting POCC are often distributed with an integrated development environment (IDE), tools for PONCC are not. For aggregating and visualizing information generated by the latter, frameworks such as SonarQube ¹ (formerly Sonar) have been developed. SonarQube is a plug-in based web application that manages the results of various code analysis tools.

The working principle of SonarQube can be described as follows: After setting up a configuration for a project, the source code files in this project will be processed by various analysis tools that have been defined in the configuration. Each tool makes its results available to SonarQube. When the reports from all tools are available, SonarQube creates several views that integrate the results reported by the tools.

For example, a dashboard view shows files or packages with the most problems, a timeline or global metrics of the analysed project. Because SonarQube is fully extensible, there are many plug-ins that add support for common programming languages or non-programming languages such as XML. Different customizable quality profiles can be used to set up the scope of the reports. By protocolling the results of the analysis process, project timelines can be created. So the progress of the quality in a project can be measured (see Hashiura et al. [7]).

B. Quality Characteristics for Business Process Models

For business-process models, there do not exist standards comparable to ISO/IEC 9126 for defining model quality. However, a remarkable number of scientific papers discuss the quality of conceptual models; for a comprehensive overview see [8] and [9]. While there are different suggestions for quality characteristics for conceptual models, many of them are based on the three quality characteristics developed by Lindland et al. [10]. For our purposes, will also use the classification by Lindland et al.: (for alternative classifications of quality goals see [11]). Based on linguistic theory, Lindland et al. identified three quality characteristics:

1) **Syntactic Quality**: The model should adhere to the rules of the language.
2) **Semantic Quality**: The model should correctly describe all the relevant facts from the domain it intends to explain.
3) **Pragmatic Quality**: The model should be easy to comprehend by its users, and their interpretation of the model should correspond to the intended meaning of the model.

C. Quality Subcharacteristics for Business Process Models

We categorise quality requirements mentioned in various sources [9] into the three quality characteristics syntactic, semantic and pragmatic quality. Of course, there would be other options for building such a categorisation. However, discussing the advantages or disadvantages of different categorisations in detail is not the main focus of this paper. Instead, the interested reader is referred to special papers about this question such as [12], [9], [11], [13], [14], [15], [16].

1) **Subcharacteristics of Syntactic Quality:**

**Formal Syntax**: The symbols used in the model should conform to the rules of the language. Usually these rules are expressed by means of a grammar or a metamodel. However, we define the correctness of the formal syntax in a broader sense: In addition to the rules formally given by a grammar or metamodel, we also consider additional constraints that can be imposed by the syntax definition. For example, in BPMN such a constraint could be that a sequence flow from a gateway to itself is forbidden. [17] includes a deeper discussion of BPMN constraints that are not formally included in the BPMN metamodel.

In addition to those general syntactic constraints that are imposed by the rules of the modelling language, there may be further naming conventions or style rules defined as organisation-wide standards. Those will not be regarded by us as syntactic rules. Instead, we classify them as sub-characteristics of “pragmatic quality” (see Subsect. II-C3)

¹http://www.sonarsource.org

![Image](http://www.sonarsource.org)

**Fig. 1.** Illustration of the ISO 9126 software quality model
2) Subcharacteristics of Semantic Quality:
Correctness: All statements about the domain given in the model should be true.
Relevance: All statements given in the model should be relevant to the purpose of modelling.
Completeness: All relevant facts in the domain should be contained in the model.
Consistency: There should be no contradictions within a model. As a graphical model can contain more than one diagram, this includes the requirement that there should be no contradictions between the different diagrams.
Executability: It should be possible to define a formal semantics for the model and to execute the model without errors.

3) Subcharacteristics of Pragmatic Quality:
Comprehension: It should be possible to understand the model without too much effort.
Maintainability: It should be possible to change the model without difficulties.
Adaptability: It should be easy to transform the model to other model languages or technologies.

D. Quality Measures for Business Process Models

In order to come to a conclusion about the overall quality of the model, the quality subcharacteristics described in the previous subsection can be measured by several measures. In this subsection, we describe a selection of such measures - without making any claim that this selection is exhaustive. While the quality (sub)characteristics given in the previous section apply for all kinds of software and business process models, many measures discussed in this subsection will be specific for business process models, in particular for those that aim to model the flow of control of a process. In Sect. III-B, we will show how we used these measures for business process models in the language Event-Driven Process Chains. However, our approach is in no way limited to this modelling language.

1) Syntactic Quality:
Formal Syntax: The simplest outcome of a measure for syntactic quality is a yes/no decision whether the model conforms to the syntactic rules. Alternatively, it is possible to count the number of syntax rule violations or to measure the quotient between the number of such violations and a Size measure such as “number of nodes in the model”.

2) Semantic Quality:
Relevance / Completeness: Recker [18] states that “it is impossible to assess the model against an objectively existent business process”. In particular, this statement is true if we are interested in correctness checks that can be done automatically by a computer system. Cares and Franch [19] also point out that “it is not possible to check semantic validity without the audience”, and they suggest a method to measure semantic quality by asking readers of the model whether their interpretation of the model corresponds to their knowledge about the domain. It would be possible to obtain the measures defined in [19], [20] or [21] by a system that forwards the models to human readers for a review and collects their answers. However, when our interest is in analysing models without human involvment, we can not consider such methods.

However, often the business process depicted in the model is restricted by compliance rules. Such rules can originate from legal constraints or from the business rules in an organisation. If those rules are given in a formalised way, we can define Compliance measures that quantify how well the model adheres to the rules. Many formal approaches for compliance checking can be found in the literature [22], [23], [24], [25]. Possible measures include a yes/no decision whether a model adheres to all compliance rules, the number of rule violations or the number of rule violations, weighted by the importance of the rules.

Another possible approach to deal with the subcharacteristics Relevance and Completeness would be to compare the element labels in the model with concepts included in a domain, expressed by an ontology. There are first suggestions in this direction [26], [27], but they have the disadvantage that a domain ontology must be built in addition to the model.

Consistency: For business process models containing more than one diagram, there should be no contradictions between the diagrams. For example, it could be the case that a business process model contains an organisational chart which depicts the organisational structure of a company. If another model shows the order of activities and the responsibilities, all persons who are responsible for performing an activity should exist in the organisational chart. A measure for consistency could be the number of contradictions found in a model.

Executability: The quality characteristic Executability requires that it should be possible to give one and only one “meaning” (i.e. formal semantic) to a model. Kindler has shown that for certain kinds of business process models this is not the case in a satisfactory way [28]. A measure can calculate whether such a formal semantics exists (i.e according to Kindler’s terminology, the Cleanliness property has to be checked [28]). Another property that can be tested automatically is Soundness. It was first defined in context of workflow nets in [29]. Roughly spoken, this property means that

- a business process instance that has been started by invoking a start event finishes by reaching an end event,
- the end event is reached only once and
- every element of the model contributes to a proper run of the business process.

Several techniques such as state-space analysis or Petri-net invariants can be used for checking the soundness of a business process model [30], [31], [32].

3) Pragmatic Quality:
Comprehension / Maintainability: Lindland et al. [10] say that for pragmatic quality, “there is one pragmatic goal: comprehension”. While it is not possible to measure the comprehension of a model automatically, we can measure factors that are known to influence the quality subcharacteristic Comprehension.

Due to the limited resources of the human mind, large models are more difficult to understand than smaller ones. Therefore, measures for the Size can be indicators for the difficulty to understand a model. We can measure the size of models for example by simply counting the activities or all nodes that a model contains.
In addition to those Size measures, a large number of Complexity metrics that are not simple size measures have been suggested for business process models [33], [34], [35]. An example of such a measure is the nesting depth within a model. It has been shown that some of these metrics are related to the understandability of a model [36].

Some of those complexity metrics deserve to be discussed as categories for measures on their own. On such category is Modularity: Decomposing a large process into sub-processes is a common way to enhance the understandability of a process model [37]. Ideas for developing Modularity measures are discussed in [38].

Another kind of measures deal with Cycles: Experiments suggest that models with cycles (loops) are more likely to contain an error than models without a cycle [39]. Measures that intend to count the number of cycles in a business process model can be defined based on graph theory.

Graph-theoretical approaches can also be applied for developing Structuredness measures. Structuredness requires that a business process model follows a strict block structure: Each node that splits the control flow in many alternative or parallel paths is followed by exactly one corresponding node that joins the alternative or parallel flows [29]. We agree with Lindland et al who state that “a structured model is not always the most concise because the price for the structure may have been additional overhead”. While there are occasions where a non-structured model is the better choice [40], [41], structured modelling should be preferred in many cases [42]. Structuredness not only helps to make a model easier to understand, it also contributes positively to the quality subcharacteristic Adaptable (because only well-structured models can be transformed into languages that require a block-structure such as BPEL).

All the measures for pragmatic quality discussed so far are rather “technical” measures that evaluate the graph structure of a model. However, as the ultimate aim is comprehension by a human reader, the quality of the graphical layout and the textual information should be assessed as well.

There exists a large body of literature on the relationship between the Graphical Layout of a visual model and its understandability. Language-independent results lead to guidelines such as “maximise symmetry” or “minimise edge crossings” [43]. Starting from those general rules, more specific guidelines can be developed for IT diagrams in general [44] and business process models in particular [45]. Measures can quantify the degree of derivation from these guidelines.

Element labels, i.e. the texts inside the model elements that verbally describe the concept that the box represents, are further important contributors to model quality. For example, misunderstandings can be avoided if always the same phrase is used to describe the same concept throughout all models in a modelling project. There is a wide range of measures for the quality of the textual information (Label Quality) in a model.

A study of Mendling and Reijers [46] suggests that the labelling style has an influence on understandability. Therefore, conventions requiring a consistent labelling style (such as “verb-object” for activities) can be useful. Algorithms that can check whether a label follows a given naming convention have been described by Delfmann et al. [47] as well as by Leopold at al. [48]. Measures can use such algorithms in order to quantify the derivations from existing naming conventions.

Furthermore, Leopold et al. used different natural language processing techniques for calculating metrics based on the length of activity labels and on the occurrence of different labelling styles [49]. Friedrich [50] exploits the lexical database WordNet\(^2\) to calculate a measure that aims to find labels with a high chance of ambiguity.

In addition to the general measures for pragmatic quality, there may also be organisation-specific measures. These measure the adherence to Organisation-Wide Style Rules. If a modelling notation allows more than one way to express a certain fact, it is often helpful to introduce such rules requiring that always one of those alternatives has to be used in all models. For example, the BPMN standard offers three ways for modelling a decision: Two differently-looking symbols for an exclusive gateway (which have the same meaning anyway) and conditional flows (where a task has two outgoing arcs). For such cases, modelling in an organisation can become easier if all models use one and only one of those variants [51]. Guidelines such as Ambler’s “Elements of UML Style” [52] or collections of best practices for BPMN by Suarez et al. [51], Silingas and Mileviciene [53] or La Rosa et al. [54] list common best practices for graphical models. Once again, measures can count the violations of such rules.

III. THE SONARQUBE PLUG-IN FOR EPCS

A. Architecture of our Tool

In this section, we describe SonarQube Plug-in for EPCS. It collects quality measures calculated by various tools and presents the results in various views.

The process of running the SonarQube Plug-in for EPCS can be sketched as follows: Somewhere within the model repository there is a configuration file which contains settings for the used modelling language and locations where the models can be found. This configuration file is read by the SonarQube Runner, a tool provided by SonarQube to start the analysis (Step 1). Then, depending on the settings, model files are read and committed to the SonarQube server (Step 2). Based on the configuration given for the modelling language, the server knows which plug-ins have to be called for processing. For business process models in the language Event-Driven Process Chains, this is our SonarQube Plug-in for EPCS.

This plug-in calls three tools: Label Style Checker, EPC-Metrics Calculator and Prolog Analyzer. Each such tool calculates a number of measures of quality subcharacteristics (Step 3). When all calculations are done, the SonarQube Plug-in for EPCS transmits all results to the SonarQube database (Step 4).

The analysis process can be started interactively (for example from within a business process modelling tool) or automatically by a batch job. When it is finished, a reviewer can analyse the generated reports (Step 5). These reports include multiple views which are accessible and configurable by the SonarQube web service. This way, a reviewer can

\(^2\)http://wordnet.princeton.edu
Quality Subcharacteristic | Measure(s) | Reference(s)
--- | --- | ---
Syntactic Quality → Formal Syntax | Number of Violations of Syntax Rules for EPCs (using the rules described in [55]) | [55]
Semantic Quality → Executability | Pattern-based search for soundness violations (using the rules described in [56]) | [29], [56]
Semantic Quality → Relevance | Pattern-based search for needless elements (using the rules described in [57]) | [57]
Pragmatic Quality → Size | Various count measures (such as number of activities) (using the tool described in [58]) | [33], [35]
Pragmatic Quality → Loops | Number of loops in the model (using the tool described in [58]) | [35]
Pragmatic Quality → Label Quality | Violations of naming conventions (using WordNet) | [46], [57]
Pragmatic Quality → Structuredness | Test of well-structuredness (yes/no-decision) | [42]
Pragmatic Quality → Graphical Layout | Various metrics for the graphical layout (such as number of edge crossings) (using the tool described in [58]) | [58]
Pragmatic Quality → Complexity | Various complexity metrics (using the tool described in [58]) | [33], [34], [35], [58]

**TABLE I. MEASURES IMPLEMENTED IN OUR TOOL (SELECTION)**

---

Fig. 2. Illustration of the architecture and measuring process

Fig. 3. Most common problems

**B. Implemented Measures**

The tools which are invoked by the SonarQube Plug-in for EPCs can be described as follows: The Label Style Checker parses all function labels of a model. Then it uses Wordnet to evaluate whether the label does match the “verb-object” style. The EPCMetrics tool calculates several metrics, in particular for the quality subcharacteristics Size and Complexity. The Prolog Analyzer creates a Prolog fact base that contains all the information which is included in the business process model. Then it commits these facts to a Prolog processor (SWI-Prolog\(^3\)) and runs a Prolog program which searches for known problem patterns [56], [57]. Finally, the results of the program are parsed and committed back to the SonarQube Plug-in for EPCs.

So, after all three tools have been finished, the SonarQube Plug-in for EPCs receives a number of results (i.e. calculated measures) which will be committed to the SonarQube database. Figure 2 illustrates the whole process.

**C. Visualisation**

After calculating various measures for a collection of models, SonarQube provides many possibilities to present the results. The views can be configured according to the needs of the reviewer. Some possibilities are shown in the figures: The view in Fig. 3 shows the number of problems found in the repository and the number of occurrences of the most common problems. Fig. 4 shows a ranking of the models with most problems, and Fig. 5 shows how quality measures change in the course of time. In addition to such problem messages, detailed measurement results can be shown for each model (see Fig. 6). Other views can be used to compare model repositories (see in Fig. 7, Fig. 8 and Fig. 9).

For generating these views, we created three collections, each with 49 respectively 50 models from the following sources: models published in scientific papers in one repository, models created by students for bachelor/masters theses and term papers an another one, and models that have been used in real life business or government projects in a third repository.

Based on the wide (and extensible) range of visualisations provided by SonarQube there are various capabilities of analysing and comparing model repositories. For example,

---

\(^3\)http://www.swi-prolog.org
Fig. 4. Models with most problems

Fig. 5. Progress of model quality

Fig. 6. Metrics for a model

Fig. 7. Comparing model repositories

Fig. 8 shows that the models from real life projects are larger on average than those from scientific model repository and those constructed by students. While such views are helpful for getting an impression on model quality, the person who reviews a model repository has still the responsibility for interpreting the results. Deliberately, we did not include recommendations such as “model size should not exceed $x$ nodes” into our tool. The reason for this is that there are simply no convincing research results that would allow such a statement. For example, a look into the literature shows that [59] suggests that a business process model should be decomposed into smaller sub-models if it has more than 65 nodes. Other articles suggest such a decomposition if the model size exceeds 50 nodes [60] or even only about ten nodes [53]. An even lower number is given by the IDEF0 standard which requires that a diagram must not have more than six boxes, otherwise the model has to be decomposed.
All those recommendations are based on the assumption that models are too difficult to comprehend if they exceed a given size, but there is no general answer what should be regarded as a suitable upper limit. Even for those papers that make recommendations based on experiments (such as [60] or [59]), a closer look reveals that the experimental setup does not allow to make conclusions which can serve as general guidelines.

IV. Conclusion

In our paper, we presented an approach and a tool for assessing the quality of a collection of business process models. This is done by calculating, aggregating and visualizing various measures for collections of business process models. The key idea is that model quality is reviewed in the same way as software code quality. Also, the environment used for evaluating the quality of models is the same one that is used for evaluating code quality. Given that conceptual models in recent years became more and more important artifacts in the software development cycle, it is just a logical step to assess their quality with the same rigor and using the same frameworks which are already used for assessing software quality. This becomes even more important if models (such as BPMN models) become executable and are not only a prerequisite of software development, but even a part of the final software product.

However, even if the purpose of modelling is not developing software, but to understand and to optimise business processes, we are convinced that an integrated tool for assessing the quality of the models is still helpful.

We believe that the idea presented in our paper can even be used in a wider sense: An interesting direction for future research could be to extend the work by using a quality model that aims to measure not only the quality of the business process model but also of the depicted process itself. Suggestions for quality sub-characteristics and quality metrics that can be used for this purpose have been made in [61], [62], [63] and [64]. In a similar way, measures on business process models could be used to estimate the functional size of the software that aims to support the process depicted in the model [65].

REFERENCES
