**Available Online at www.ijcsmc.com**

**International Journal of Computer Science and Mobile Computing**

**A Monthly Journal of Computer Science and Information Technology**

**ISSN 2320–088X**

***IJCSMC, Vol. 4, Issue. 5, May 2015, pg.574 – 590***

**RESEARCH ARTICLE**

# Software Quality Assessment Tool Based on Meta-Models

**Rositsa Doneva[1], Silvia Gaftandzhieva[2], Zhelyana Doneva[3], Nevena Staevsky[4]**

[1,2] University of Plovdiv "Paissii Hilendarski", Bulgaria

[3] CSC, Bulgaria

[4] Bebit GmbH, Germany

[1] rosi@uni-plovdiv.bg, [2] sissiy88@uni-plovdiv.bg, [3] zdoneva@csc.com, [4] nevena.staevsky@bebit.de

***Abstract — In the software industry it is indisputably essential to control the quality of produced software systems in terms of capabilities for easy maintenance, reuse, portability and others in order to ensure reliability in the software development. But it is also clear that it is very difficult to achieve such a control through a 'manual' management of quality. There are a number of approaches for software quality assurance based typically on software quality models (e.g. ISO 9126, McCall's, Boehm's and Dormey's models) and software quality metrics (e.g. LOC, McCabe's cyclomatic complexity, Halstead's metric, Object-oriented metrics) for assessment of various quality characteristics. Since the appearance of the software quality assurance as a field in the software engineering, researchers have been looking for ways to automatically assess and manage the quality of the software systems. This paper presents a conceptual design of a comprehensive solution, referring to the automation of the software quality assessment process. The designed software tool allows the definition of software quality models, based on standards, and enable the setting of matching between criteria of a software quality model and appropriate software quality metrics. The automatic definition and application of software quality models and software quality metrics is based on relevant supported by the software tool meta-models proposed in the paper.***

***Keywords — Software quality, Software quality model, Software quality metric, Meta-model, Automated software quality assessment***

## I. INTRODUCTION

Different studies [9], [13], [17], [21] show that for the most software systems the costs for maintenance are between 60% and 80% of the total costs. An unwritten rule is also that the costs for maintenance can only increase, but never decrease. Therefore, it is essential during the development of software systems, that their quality be controlled in terms of capabilities for easy maintenance, reuse, portability and others.

There are a number of approaches for software quality assurance (SQA) known. The study, presented in this paper, focuses on the so called static techniques that, unlike the dynamic techniques, which are primarily a subject of the "Testing"-field, don't require the execution of the assessed software systems.

 

Different models, related to SQA, exist. For example: the standard model ISO/IEC 9126 [11], updated in the recent years through the SQuaRE series of standards; the models of McCall [10] and Boehm [3], [4], on which ISO/IEC 9126 is based; the model of Dormey [7], etc.

In general, the ***software quality model*** (SQMo) consists of three major components: quality characteristics (factors), their sub-characteristics (criteria), and corresponding metrics for their assessment – they that can measure the process of design and development, as well as the software itself.

One of the most popular SQMo – ISO 9126, for example is a hierarchy of six characteristics (`functionality, reliability, usability, efficiency, maintainability and portability`), their corresponding sub-characteristics (27 in total) and examples of quality metrics. Some metrics are purposed to assess internal qualities, such as `Traceability, Cyclomatic Complexity of McCabe` [22], etc., and others – external qualities, such as `Functional Adequacy, Accuracy to Expectation`, etc.

Commonly, in order to apply a certain ***software quality metric*** (SQMe), the required measurements are performed 'manually', because they require a human assessment in terms of source code, documentation, etc. Such 'manual' approach in the implementation of procedure, related to software quality (SQ) assessment, has a number of disadvantages. They may:

- lead to errors, because they significantly depend on the person, who makes the measurement. But yet humans may disregard and even consciously ignore some problems;
- take a long time. For example, with measurements of source code people have to read and comprehend a significant amount of code that they may not have even written;
- be partial and incorrect. For example, in a situation, where the quality manager of a software product requires assessment reports from a certain developer, and later he uses those reports to assess the work of the same developer.

Therefore, the 'manual' measurements do not meet the requirements of measurement procedures that include objectivity and exactness (i.e. achieving equal results with each measurement "run"). Their disadvantages acquire even more burdensome with the modern trends of the software development, which widely use outsourcing and integration of open source components. It means that, in order to ensure a reliability in the software development, it is not only sufficient to check the correctness of the general functionality, but also to assess the internal qualities of the integrated external components. In this case, the 'manual' measurement of quality is not applicable.

Regarding the SQ, it is also important to mention, that many users of software products (such as government organizations or people, who work in fields, related to the security and safety) require a certification of quality of the product development process, for example according to standard ISO 9000. The meeting of these requirements implies conducting of a well-founded quantitative statistic control over the product quality, to achieve continuous improvement of the SQ and the process of its development. Such a control would be very difficult to achieve through a 'manual' management of quality, because of the above-mentioned reasons.

Since the appearance of the SQA as a field in the software engineering, researchers have been looking for ways to automatically assess and manage the quality of software systems.

Unfortunately, the definitions of many known SQMo and their corresponding metrics are not precise enough and allow different interpretations, which makes them unsuitable for automatic assessment of software products.

The main approaches, applied in this field, are the formalization of definitions – through developing of common definition frameworks – and the specification of meta-models (mostly of software metrics). These methods are implemented as software tools, which automate the process of SQ assessment.

Plenty of elaborations and research projects, related to the (self-) assessment and management of SQ through automation of the corresponding procedures, are available. Only a few examples, that illustrate different approaches and solutions, are discussed in this section.

Cavano and McCall [6] developed a framework that allows a clear identification and definition of factors and criteria for quality, their relationships with metrics, as well as the assessment/combination of factors, criteria and metrics. The proposed framework simplifies the automation of SQA and it serves as a basis for the development of the SQMo of ISO 9126.

The Moose Re-engineering Environment [1] is a tool, developed on the basis of the meta-model for object-oriented metrics FAMIX (a result from the realization of FAMOOS project) and a framework of 50 object-oriented metrics (30 of which are independent of the programming language), which assess the major part of the mandatory elements of object-oriented software systems.

Lee and Chang [14] presented a tool, called RAMOOS, suitable for the assessment of the factors `reusability` and `maintainability` of software systems (depending on the criteria `complexity` and `modularity`). The tool is developed on the basis of their own SQMo and a semi-formal definition of a metrics set, proposed by them.

An area of interest is the QM tool, which supports the definition of metrics and development of SQMos, based on the ISO/IEC 9126 standard [5].

A different approach (called ADEQUATE) for the quality modelling and SQ assessment through an adaptive quality model is proposed in [12]. In addition, a software tool has been created, that supports the implementation of the ADEQUATE approach and allows users to create and visualize relationship charts, polarity profiles and overall quality scores of a product, a

 

process or a resource. The tool includes a database for storing and retrieving of quality criteria definitions and taken measurements.

Furthermore, there is a wide range of software tools to facilitate the validation and assessment of software products, developed in different programming languages [22]: multi-language (Moose, Copy/Paste Detector, Axivion Bauhaus Suite, BugScout and etc.), .Net (FxCop, StyleCop and etc.) Java (PMD, Soot, Squale, Fortify and etc.), C/C++ (Lint, Splint, Clang, cppcheck and etc.). Similar tools are designed for the static assessment of SQ and they are based on different high-level programming representations, such as UML, Java Modeling Language, oriented graphs, etc. One of them – ConQAT [19] – is worth noting. It is an integrated set of tools that allows SQ characteristics to be monitored continuously. It provides a full set of tools to analyze quality of programs written in many programming languages (e.g. Java, C #). Furthermore, ConQAT is integrated with various tools for quality analysis (e.g. PMD, FindBugs, FxCop).

This paper proposes an approach to overcome the above-mentioned disadvantages of 'manual' SQA. It presents a conceptual design of a comprehensive solution, referring to the automation of the SQ assessment process through software tools. These allow the definition of SQMo, based on standards, and enable the setting of matching between criteria of SQMo and appropriate SQMe. The automatic definition and application of SQMo and SQMe is based on relevant supported by the software tools meta-models.

The paper begins with a short introduction in the field of SQ, its terminology and existing solutions for automatic assessment and control of the quality of software systems. Secondly, the requirements of an appropriate software tool with an application in SQ assessment are analyzed (Section 2). Furthermore, some meta-models are designed to serve as a basis for a programming system with such functionality (Section 3). And finally, the paper presents the conceptual design of the system: potential users, the main sub-systems, relevant use cases and a typical scenario for the process flow (Section 4).

## II. REQUIREMENTS ANALYSIS

The specific problem area of this work is the SQ assessment, considered in the context of automation of the related activities.

The aim of the analysis is to give a theoretic reasoning to the proposed solution, i.e. to define requirements, related to the design and development of a platform (supposedly a web-based one), which automates the SQ assessment process and is based on standards and metrics.

The defining of the requirements for the designed system is made from the perspective of the following *SQ definition*, which is based on general principles for quality in the context of software, but is furthermore adapted (in order to meet the set goals) to reflect the aspects of the internal quality of software products: ***The quality of software is a set of characteristics of the software systems, that are essential for satisfying certain specified requirements***. ***These requirements are defined in the form of characteristics and sub-characteristics, and are possibly assessed using software metrics. The metric values express the level of/ability for fulfilling the needs.***

The requirements analysis is introduced through a series of observations/conclusions made, and resulting requirements to/solutions for the designed software tool (see Table 4.1). Since the purpose of the analysis is purely the development of a common conceptual design of the system, only ***basic functional requirements*** are formulated here.

TABLE 1. FUNCTIONAL REQUIREMENTS ANALYSIS OF THE DESIGNED SOFTWARE SYSTEM

| ***№*** | ***Analysis Observation/ Conclusion*** | ***Requirement to/ Solution for the Software System*** |
|---|---|---|
| 1. | ***O1.*** The known practices for assessment of the quality of the software often suffer from inaccuracy, no reusability, or form a subjective assessment, mostly because they are executed 'manually' and depend on the impartiality, experience and competence of the assessors | ***R1.*** Enables *automated quality assessment*, based on formalized patterns of the participating items:<br>• ***R1.1.*** with a high *degree of objectivity, reliability and portability*<br>• ***R1.2.*** where the formalized patterns are based on *standards, metrics and best practices* |
| 2. | ***O2.*** A significant problem in assessing the SQ is the multi-aspect nature of the concept "quality of software systems". Multiple heterogeneous components, such as source code characteristics, supportability, efficiency, portability, etc., are assessed in different contexts (for the needs of SQA, for forecasting, etc.) and according to various requirements (complying to the practices or standards of different organizations) | ***R2***. Provides the opportunity for modelling of different *assessment tools/elements* (of SQ, metrics, procedures), i.e. the maintenance of relevant meta-models, which includes possibility for:<br>• ***R2.1.*** *Definition* of the models<br>• ***R2.2.*** *Interpretation* (configuration, application, tracking, etc.) of the built models. For example, it provides real-time information about the status of a procedure. |
| 3. | ***O3***. A common feature of all SQMos is the trend to precisely organize the elements of quality (factors, attributes, criteria) in a *related structure of not intersecting and uncorrelated categories* | ***R3.*** Offers a maintenance of *meta-models* for definition and application of hierarchical SQMo |
| 4. | ***O4***. *Appropriate assessment metrics* correspond to every quality criteria | ***R4.*** Offers a maintenance of *meta-models* for definition and application of SQMe |

 

| № | ***Analysis Observation/ Conclusion*** | ***Requirement to/ Solution for the Software System*** |
|---|---|---|
| 5. | ***O5.*** For the completion of a stage of the SQ (for example the application of a certain SQMe) a series of steps – a procedure – has to be performed (such as data collection, metric evaluation, etc.). These *procedures* vary in purpose and regulations, and can involve *heterogeneous participants* | ***R5***. Offers a maintenance of *meta-models* for definition and application of procedures, related to SQ assessment |
| 6. | ***O6.*** The SQ assessment is performed by *groups of participants* of different importance, and with different roles and responsibilities (e.g. project manager, quality assurance manager, developer, tester, user of software system) | ***R6***. Allows a flexible *user management* – authorization, individual access rights/management of different functions, etc. |

***Additional functional requirements*** for the software tool in question are:

- to support information, assisting the construction of models (e.g. which of the already built metric models are applicable for assessment of a certain characteristic/sub-characteristic of SQMo);
- to provide storage (an integrated database) of the developed SQMo (metrics, assessment procedures in prepared, initiated or completed stage, , etc.);
- to be able to store concrete results from the application of developed models (e.g. past, upcoming, open and completed assessment procedures);
- to support information about already built models, which have a certain meta-model as a basis;
- to support information about concrete applications of the developed models;
- as a result of the above three to be useful by:
- reuse;
- search;
- analyses, assessments, etc.;
- to automate the preparation, processing, exchange (between the different participants in the assessment) and even the generation of documents, needed in SQ assessment (quality plans, analyses, reports, statements).

## III. PRESENTATION OF META-MODELS

***Meta-models*** that are supported by the conceptualized software system allow modelling of assessment instruments for *definition and interpretation* of models, with the help of specific standards (such as SQMo, metrics, procedures). The task to define them is quite a complex one, and its solution is furthermore the success key to the overall solution for automated assessment of SQ.

With the help of the proposed meta-models (represented below in UML class and sequence diagrams), it should be possible to set SQ models that satisfy the following ***basic requirements***:

- *Unambiguous*. The definition of SQMo, metrics, measurement procedures, etc. is unambiguous, and allows one single interpretation of the model: with unchanged input data, the same results are always obtained. This would allow comparison of results, obtained by different teams and in different software systems; this would increase the transparency of the obtained data, facilitating its interpretation, and would also contribute to external assessments, and so on;
- *Based on standards*. Respecting the objectives, set above, the assessment of SQ has to be done in accordance with known standards, appropriate software metrics and practices. Where necessary, relationships between built models (e.g. between standards and metrics) have to be defined;
- *Reusable*. The concept of the definition of models one the basis of underlying meta-models allows their application for different software systems and programming languages (i.e. reusability), or even for high level representations of software systems (such as UML). Thus, the models are defined only once, and can be applied to all existing or future programming systems and languages, provided they make sense in the corresponding context;
- *Repeatable* (this requirement could be considered as a part of the latter, but here it is separately included to provide a better clarification). It is possible to repeat each step of the interpretation of the model, which leads to a highly efficient assessment of the quality of software systems. For example, using the same SQMo for assessing software systems, designed in different programming languages, would save time for adapting to the new software system, for a redefinition or new implementations of metrics, quality models or tools. Another resulting advantage is the possibility to automatically conduct the same assessments repeatedly for different versions of the same software system, thereby obtaining comparable results;
- *Automated*. This requirement is not absolute, but it suggests the automated application of tools for SQ assessment, whenever it is possible (e.g. retrieving data from metrics, and calculation). In this way, efficiency and reliability in the assessment are achieved;
- *Maintainable*. The models are flexible and can be modified and extended (e.g. when it is necessary to reflect new, unknown elements and relations of new programming languages). This requirement is provided by the concept of the definition of models one the basis of underlying meta-models.

 

Since the designed programming system is based on a common approach to support assessment tools meta-models, the deriving models contain some common elements – all of them support information about:

- *assessed objects* (association of objects with corresponding models that can be used in their assessment);
- *container model*, i.e. which other model (of procedure, metric, or quality), built in the system, it is part of (if applicable). The container model is the receiver of the output data (e.g. a model of a particular metric is related to the model of the procedure for its evaluation).

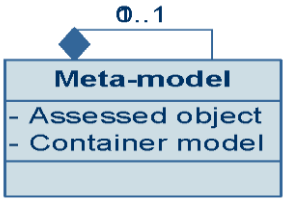


Fig 1. UML class diagram of abstract meta-model

This is presented in the project through a specially provided common (abstract) meta-model (see Fig. 1).

*A. SQMo meta-model*

***The purpose of the SQMo*** meta-model is to allow modeling of SQMo based on standards. An additional requirement is to provide development and interpretation of quality models, which can be both general and specific to a given area, according to the desire of the model developer.

Among the many published SQMo there are some, which explicitly compare certain metrics against some quality factors, such as maintainability or susceptibility to faults.

Other models are more general and provide a specific methodology for definition of the relation between the metrics, quality characteristics and quality factors or objectives. The most famous methodologies are McCall's approach Factor-Criteria-Metric (FCM), [10] and Basili and Rombach's approach Goal-Question-Metric (GQM), in which the definition of appropriate metrics is made by asking questions to achieve the stated quality objective [2]. A number of other improved quality models have been developed, but the only internationally-standardized SQMo is the ISO/IEC 9126 [11]. Therefore, in the implementation of the meta-model requirements, set out in this work, ***the meta-model presented here is most conformable to this latter standard.***

As already explained, according to ISO/IEC 9126 standard, as well as according to other SQMo, any model of SQ principally includes three structured components: quality characteristics, quality sub-characteristics and metrics (in terms of ISO standard).

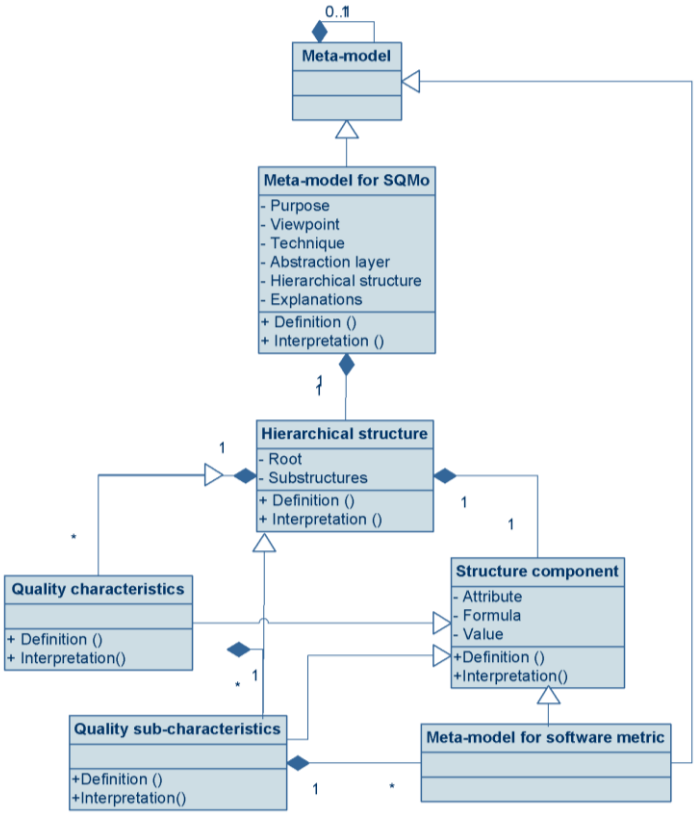


Fig. 2. UML class diagram of a meta-model for SQMo (for a meta-model see Fig.1, for a meta-model for software metrics see Fig.4)

 

Therefore, ***a key feature of the proposed in this work meta-model for SQMo*** is the ability to define a `hierarchical` structure of these three components – quality characteristics, sub-characteristics and software metrics for their measurement (see Fig. 2). The presentation of each of these three components includes the ordered three <`Attribute`, `Formula`, `Value`> (a `definition`, a `type`, etc. can also be included.), where `Attribute` is the name of corresponding component, `Formula` is either a mathematical formula or a relation to a software evaluation tool (e.g. of metrics) and `Value` sets the quantitative value of its measurement (assessment). The formula for calculating the average of the lower level components of the structure, is considered as the default value for `Formula`. It is furthermore possible to set a corresponding `Weight`, according the component's importance for the assessment of the quality in this model. In this case, the default value for `Formula` will be the formula for calculation of the weighted average, according to the lower level nodes and their specified weights. In order to facilitate the interpretation, further characteristics of the hierarchical components could be also set: `Scale` – to determine the scale, used at measuring the value (e.g. `Nominal`, `Ordinal`. `Interval`, `the Ratio`, see http://en.wikipedia.org/wiki/Level_of_measurement), and `Area` – to determine the value range of `Value`. These possibilities are not shown in the meta-model, presented in Fig. 2, but the meta-model can be easily expanded.

In conformity with the suggestions, made by the authors of [20], ***several additional elements*** are included in the meta-model (see Fig. 2). For simplicity, these are presented as attributes of the class 'Meta-model for SQMo', although they are actually classes of their own. These elements are ***important*** in order to avoid any overlapping and/or contradictions in the diverse SQMo:

- `The purpose of the model` – whether it is for construction, forecasting or assessment;
- `Viewpoint` – whether it is from the viewpoint of the product, the user, the production or the price;
- `Techniques` – if SQMo is focused on a specific technique, e.g. inspection;
- `Abstraction layer` – the detail level of the model, e.g. whether it is general or for a specific product.

***In addition*** (see Fig. 2), it is provided for including of `Explanations` – information about good and bad practices, and so on.

An example of a SQMo fragment (based on the ISO/IEC 9126 standard), which gives an idea of its hierarchic nature, is presented in Table 2.

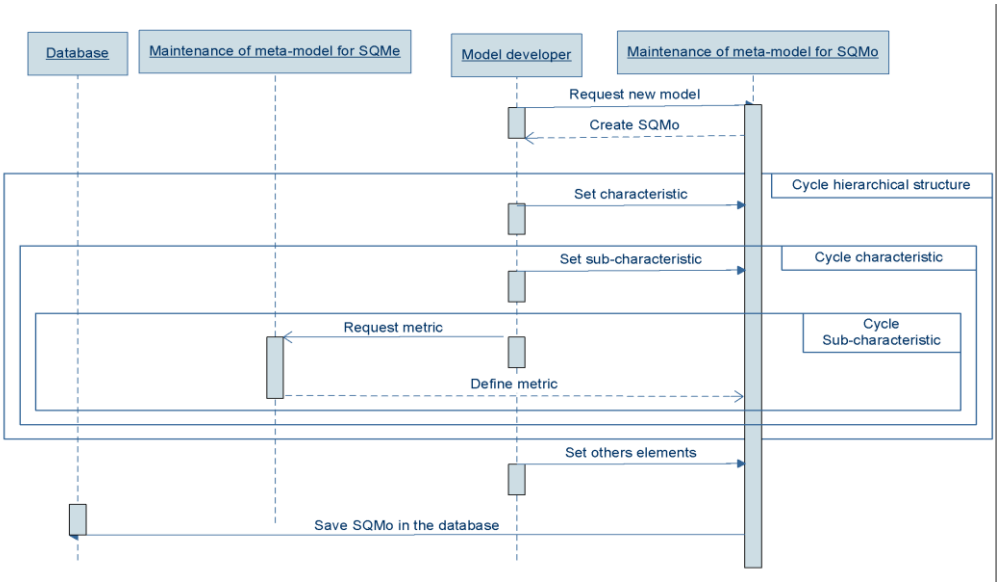


Fig. 3. UML sequence diagram of definition of SQMo (for Define a metric see Fig. 5)

All of these components are described ***in the definition of SQMo, based on this meta-model*** (see Fig. 3), including the components of the hierarchic structure of characteristics and sub-characteristics. Each sub-characteristic of the quality is associated with a metric model that is either built in advance, or for the particular case, and provides its measurement (see Fig. 5). This is achieved with the help of the module `Maintenance of a meta-model for SQMe`. Nevertheless, the major role in the choice of metrics is played by `the Model developer`, who measures their adequacy to the purpose and the assessed characteristics, the clarity of the interpretation, the efficiency of their obtaining, etc. The so built SQMo is stored in the database. Before proceeding to the definition of a completely new SQMo, the `Model developer` can perform an automated search in the database (e.g. according to its `Purpose` or `Abstraction layer`), to find among the previously developed and stored SQMo an appropriate one, that suites the currently assessed software project. This possibility is not shown in Fig. 3, but is presented as a part of the typical process flow scenario, at the time of using the designed system (see Fig. 10).

 

A principal role of the ***interpretation of SQMo based on a meta-model*** [1] is to generate the results from the quality assessment. These can be presented in the form of an overall assessment of the product quality, obtained through a bottom-up evaluation of the hierarchic structure components, according to the specified formulas and possibly weights, or in another specified form (a diagram, a report). In addition, it displays in a user-friendly way the values of the different SQMo elements.

*B. SQMe meta-model*

***Software metrics*** are necessary in the field of software engineering for taking technical and project decisions, for managing the creation process of software systems, as well as for the presented in this work subfield – SQ assessment – as a tool for valuation of internal attributes of SQMo.

Many metrics have been reviewed in the literature so far, which are popular enough (some of them are addressed in the Section 1), but there still doesn't exist a uniform metric to provide a universal result for different SQMo.

Many of the used SQMe are in practice applicable to a large class of (and even to all) software systems. But at the same, a significant part of the most important metrics is being developed individually, according to requirements of the specific project or the subject area.

In order to allow for the creation of appropriate metrics for a selected SQMo and a specific software system, the definition of metrics in the here designed system is based on a common meta-model.

***The purpose*** of the meta-model for SQMe is to allow modelling of SQMe, appropriate for the assessment of the criteria and factors of the SQ. It is assumed, that the meta model, supported by the system, allows the definition and interpretation of the so called direct metrics (by the evaluation of which a single value is obtained), as well as of indirect metrics (which provide more than one value). An example of a direct metric is `Lines of Code` (`LOC`) – the number of lines of source code in the output text of the program, whereas an example of an indirect metric is `Programmer Productivity` – productivity of the programmer in terms of written lines of source code, instructions, pages of documentation, test cases, etc.

The most popular SQMe, which have been reviewed in the literature, are considered in the development of the meta-model presented here – those of the well-known McCabe's classical metrics [22], by Halstead [23] and packages from Chidamber and Kemerer's metrics [18], by Li and Henry [18] and by Bieman and Kang [16]. But it is the examples of internal metrics, given in ISO 9126-3, that are primarily considered as a base for the meta-model for SQMe. The reason for that is the specific aim of the work to assess SQ through measurement of the internal characteristics of the product. Examples of external metrics are given in ISO 9126-2.

The meta-model for SQMe, supported by the system (see Fig. 4), facilitates the formal definition of metrics in an unambiguous and a programming language independent way. It contains the following ***basic elements***:

- `Attribute` (Name) – a short symbolic metric name, by which it is unambiguously identified and is possibly known in the literature (this is the element `Attribute` from `Structure component`, see Fig. 2). For example `LOC`;
- `Formula` – defines a method for an automated obtaining of the metric value and can be of two types. At the first one `Formula` (this is the element `Formula` from `Structure component`, see Fig. 2) contains a link to an external software tool for the evaluation of the metric. At the second type it is formally defined in a declarative language, using for example the theory of sets and the mathematical apparatus. For example, the formal definition of `LOC`;
- `Value` – specifies the quantitative value of the measurement (assessment) – this is the element `Value` from `Structure component`, see Fig. 2;
- `Procedure` – defines relation to the procedure model, built using the meta-model for procedure (see the following section), for obtaining of the metric value. In addition, the element can contain a verbal description of a procedure, giving the user instructions about the evaluation of the metric. For example, instructions regarding the procedure of collecting necessary information for the evaluation. If the metric `LOC` is used for an illustration, this instruction can require the logical lines of source code to be counted and the value to be formed as a result from the procedure. The adoption of the element `Procedure` is necessitated by the fact, that regardless of the valuation method of the metric ('manually', e.g. through inspection of source code or using of empirical dependencies, or by automated tools, or by expert way – polls, etc.), it is always based on the implementation of some procedure;
- `Type` – sets the type of metrics, according to an accepted by the model developer qualification, depending on the software aspect the metric mostly affects (e.g. `complexity`, `architecture and structure`, `design and coding`, etc.). For example, the type of `LOC` is `complexity`. This element is provided in the meta-model in order to facilitate the demand and supply of an appropriate metric among the previously developed and stored in the database models of SQMe;
- `Sub-type` – sets the specific sub-category (if there is any) of the metric type (described by the above element of the meta-model), according to an adopted by the model developer classification. For example, the metric of type

[1] As an example of an interpretation see Fig. 8 – the interpretation of a procedure

 

Complexity can have the sub-type: size, complexity of interaction and structural complexity. In such case, the sub-type of LOC is size;

- Description – the metric's text description in a natural language; what is measured with its help. This description is a summary of how the metric is described in the scientific literature. It doesn't set its formal definition;
- Scope – defines a programming component, to which the metric refers. This could be the software system as a whole itself, a package, a folder or another component, containing separated parts of the system, a module, a file or a class of the software systems, a procedure, a function or a method, etc. For example, the scope of LOC can be a module or the software system;
- Highly-related characteristics of SQ – lists the factors and criteria of the quality (e.g. through their symbolic name – attribute), for the assessment of which this metric is appropriate (e.g. according to ISO 9126-3). The relations among factors/ criteria and metrics, thus defined, would facilitate the demand and the supply of an appropriate metric among previously developed and stored in the database models of SQMe in the building of SQMo;
- Related characteristic of SQ – lists the factors and criteria of the quality, for which the assessment of this metric is somewhat appropriate;
- References – provides references to literature, projects, repositories and programs, where the metric is used, realized or described.

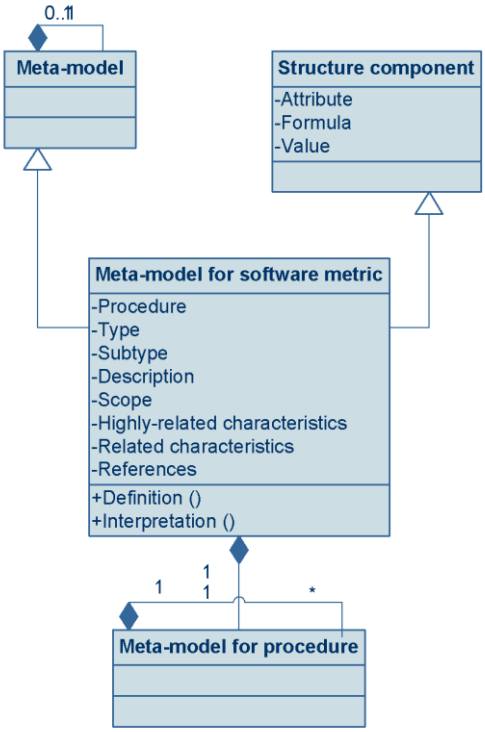


Fig. 4. UML class diagram of the meta model for SQMe (for Meta-model see Fig. 1, for Meta-model for procedures see Fig. 6)

Table 2. contains an example of the SQMe model of McCabe [22] for Cyclomatic Complexity (CC). Column 1 Element contains the elements of the meta-model. Each of them will be assigned a value (Column2 Definition), according to ISO/IEC 9126-3 at the time the model is built.

TABLE 2.THE MODEL OF SQME OF MCCABE FOR CYCLOMATIC COMPLEXITY

| *Element* | *Definition* | *Comment* |
|---|---|---|
| Attribute | CC | |
| Formula | Testworks METRIC | A link to an external software tool , see http://www.soft.com/TestWorks |
| Value | | Obtained after the interpretation of the model |
| Procedure | 'The value is calculated automatically by Testworks METRIC, but only for source code, written in C, C++, Ada and F77' | |
| Type | complexity | |

 

| *Element* | *Definition* | *Comment* |
|---|---|---|
| Subtype | structural complexity | |
| Scope | Module | |
| Highly-correlated | Usability/Understandability, Usability/ Learnability | Usability/Understandability, Usability/ Learnability, etc. |
| Related characteristics | Functionality/Interoperability, Reliability/ Maturity | Functionality/Interoperability, Reliability/Maturity, etc. |
| Refferences | http://en.wikipedia.org/wiki/Cyclomatic_complexity | The list of references and literature is not comprehensive |

***The definition of the model of SQMe, based on this meta-model*** (see Fig. 5), can be conducted in to ways, according to the Model developer's view – he can either extract a definition of an already created model of SQMe, or define a new metric. In the first case (Request for search of a metric), the module Maintenance of meta-model for SQMe automatically searches for an appropriate metric among previously developed and stored in the database models of SQMe by the criteria of the query. The query criteria define a search on some set (and/or) of values of the model elements, such as Attribute, Type, Subtype, Scope, Highly-correlated characteristics, related characteristic of SQ. If the search is successful, the module offers a list (ordered in accordance with the search criteria) with found models of SQMe, from which the Model developer eventually selects a metric. In the second case (Query for a new metric) or if the first case was not successful (e.g. no appropriate metric has been found), the module Maintenance of meta-model for SQMe defines a new metric. This requires that the values for all of the above metric elements be set. In order to associate the metric with a corresponding procedure for obtaining the metric value, a search for an appropriate procedure among the previously developed and stored in the database procedure models is conducted. If such a procedure is not found, a new procedure for the specific case is defined (see the following section). After a successful completion of all stages of the definition, the so found/built model of SQMe is provided for further use (e.g. to the module for the Maintenance of the meta model of SQMo). All new and/or updated model data is stored in the database.

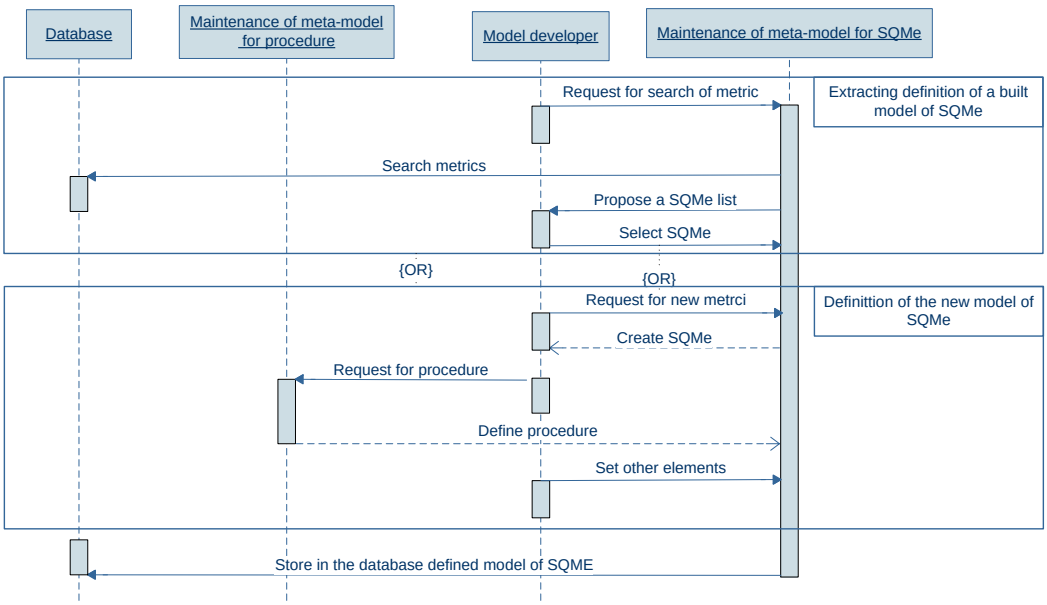


Fig. 5. UML sequence diagram of SQMe definition (for Procedure Definition see Fig. 7)

***The interpretation of SQMe based on this meta-model*** (as an example of interpretation see Fig. 8 – procedure interpretation) aims in a user-friendly way: to display the values of the different SQMe elements (mostly of the static ones), and to generate as a result the SQMe value, which has to be stored in the element Value. Depending on the definition of the model, this value can be obtained in four different ways: by using the module Maintenance of the meta-model for procedure for activating and implementing of the procedure to valuate the metric, through an automatic calculation of a given Formula, through a direct setting of the value by the user, or in an automated way by an external software tool for valuation of metrics.

 

*C. The meta-model of a procedure, related to SQ assessment*

***The meta-model of a procedure, related to SQ assessment,*** is primarily intended to maintain the modeling and interpretation of procedures for valuation of SQMe, but it could also be used to maintain other procedures in this context (see Conclusion).

The modelling of two main procedure types, with respectfully a 'hierarchical' and a 'flat' management, is to be enabled by the meta-model. The 'hierarchical' model implies a strict control at the interpretation of procedures, regarding users and roles, startup, input of information, etc. The 'flat' model, on the contrary, allows for the implementation of procedures by all interested users with a role, freely chosen by themselves.

The meta-model of a procedure, presented here, ***is based on*** the main ideas, experience, practices and principles in the field of Business Process Modelling (BPM) and its supporting technology – Workflow Management Systems (WMS), the interest in which is continuously increasing. Since the introduction of BPM [24] and for more than 40 years now, possibilities have been researched for applying BPM in different contexts (business management, software modeling, etc.).

The approach for modelling of procedures related to SQ assessment (by analogy with the approach used by WMS) is an ***adequate solution***, because the respective procedures have clearly defined stages (steps) with regulated input and output data and explicitly defined conditions for transition to the next steps of the procedure. In turn, storing the built models would allow their reuse. The generality of this approach is demonstrated as it is applied in modelling of procedures for evaluation of the quality in higher education [8].

Due to the complexity of the meta-model of procedures for SQ and the necessity to delegate separate activities to different executers, the meta-model has to be decomposed into ***simpler activities (steps)*** of two main types – elementary and composite. As a result of conducted studies on BMP and WMS (http://www.workflowpatterns.com) different sets of standard workflow activities in terms of sequence, branching, integration, synchronization, selection, etc., have been suggested and formally defined. This concept is outlined in the procedure meta-model presented bellow only in the most general form.

***In the meta-model each procedure is presented as a composite activity*** (indicated by a correspondent value of the `Type` element) and involving (as a value of the `Activities` element) all activities that need to be implemented within the modelled workflow. The presentation of each sub-procedure/activity of the respective procedure in the meta-model includes the following ***main elements*** (see Fgure 6):

- `Attribute` (Name) – a short symbolic name of the procedure/activity;
- `Description` – a text in natural language, describing the procedure and its purpose;
- `Type` – defines the step type according to a type system adopted by the model developer (e.g. in accordance with the above-mentioned samples – `elementary` or `composite sequence`);
- `Management` – defines `hierarchical` or `flat` model of procedure management;
- `Executer` – specifies who is responsible for the implementation of the activity, for example by specifying their role in the software project – `quality manager`, `project manager`, etc.). This can be also a group of executers with shared responsibilities. In the procedure/activity with `hierarchical` management model it is compulsory to determine this value;
- `Time parameters` – start dates and deadlines for implementation – relative or absolute, as well as other time requirements;
- `Preconditions` – description2 of conditions, which must be satisfied prior to the activity start;
- `Input` – description of data, resources, etc., necessary for the implementation of the step;
- `Activities` – indication of all activities included in the current step, for example through their symbolic name attribute);
- `Control` – describes the order of execution of activities included in the current step (for example sequence, repetition, etc.), as well as the conditions for completing or a choice of activities, way of integration (and/or) and so on;
- `Output` – description of the results of the execution of the current step, such as data, documents, resources, etc. (usually used as an `Input` to next steps in the workflow);
- `Status` – value that indicates the current state of the activity (for example `inactive`, `active`, `completed`, etc.);
- `Related methods` – lists the models used for building the procedure (for example models of SQMe through their symbolic name – `attribute`). The so defined relations (as between a metric and a model of procedure) facilitate the demand and supply of a suitable procedure among the models previously developed and stored in the database.

[2] It can be a textual description in a natural language or a formalized presentation appropriate for automated interpretation of the procedure.

 

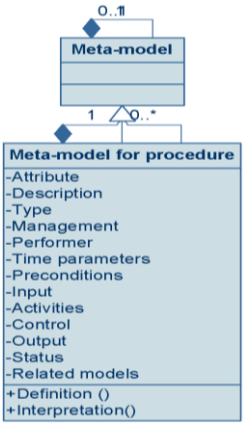


Fig. 6. UML class diagram of the meta-model for procedure/activity (for Meta-model see Fig. 1)

***The definition of the model of a specified procedure (workflow) based on this meta-model*** (see Fig. 7), similar to the definition of the model of SQMe, can occur in two variants:

- extracting a definition of an existing model of a procedure/activity through an automated search;
- defining a new model of a procedure/activity, which requires the input of concrete values for model elements. This includes modelling of all constituent activities (if any), and furthermore for each of them – defining of their own procedure model.

The created model is provided for further use (e.g. to the module `Maintenance of the meta-model for SQMe`) and is stored/updated in the database.

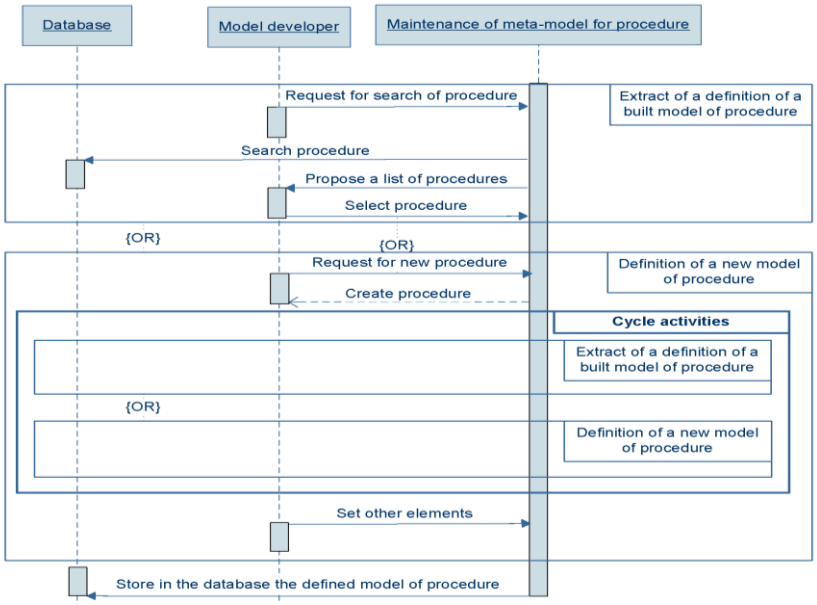


Fig. *7.* UML sequence diagram of ***definition of procedure***

***The interpretation of the so built workflow model*** of a procedure/activity is activated when the value of the respective element `Status` becomes `active` (see Fig. 8).

The values of the different elements can be visualized in a suitable way and the procedure is implemented by the indicated `Executer` in accordance with the values of elements `Time parameters`, `Preconditions`, `Input`, `Activities`, `Control`. The implementation can be automated, if `Pre-conditions`, `Input` and `Output` are set with formalized presentation, or if done by a user, who abides by the respective textual descriptions. The implementation of a certain procedure/activity is considered completed when all results of its implementation (described in `Output`) are available, and the values of all included activities (described in `Activities`) become `Completed`. In this case the value of the element `Status` of the procedure also becomes `Completed`. The stipulated mechanism of interpretation allows the procedure implementation to be interrupted at any time and respectively restored with the carrying out of those activities whose `Status is active`.

 

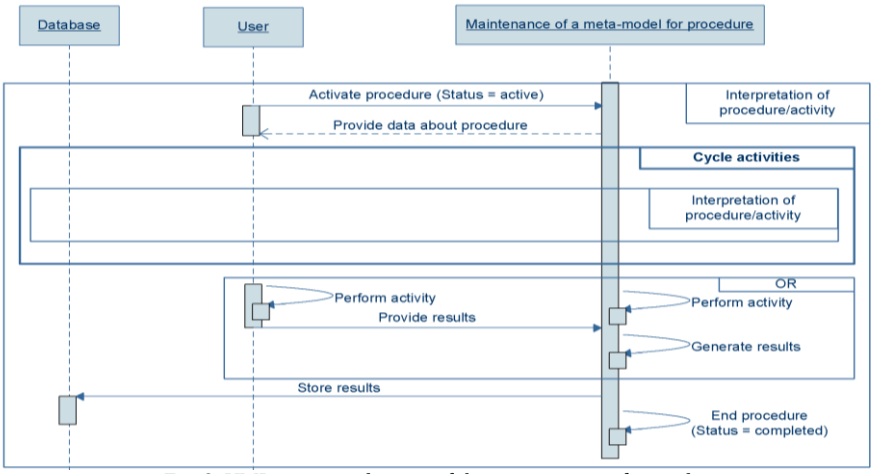


Fig. 8. UML sequence diagram of the interpretation of procedure

## IV. CONCEPTUAL DESIGN

Based on the analyses of the functional requirements in terms of the designed software system (see Table 1), this section presents a conceptual design of the concrete software system – a complete solution for automating the SQ assessment process. The project is represented by UML diagrams and reflects all main aspects and concepts, based on standard approaches to SQ assessment.

***The designed system*** (supposedly a web-based one) for SQ assessment provides functionalities for building a hierarchical SQMo (usually based on the ISO/IEC 9126 standard, similar to the SQMo fragment example, presented in Table 3) and the definition of respective metrics, as well as means for their automated valuation and analysis.

TABLE 3 FRAGMENT OF SQMO [15]

| ***Characteristic*** | ***Sub-characteristic*** | ***SQMe*** |
|---|---|---|
| Functionality | Interoperability | LOC(Lines of Code), SIZE2(Number of Attributes and Methods), CC(McCabe Cyclomatic Complexity), WMC (Weighted Method Count), DIT (Depth of Inheritance Tree), CBO(Coupling Between Objects), CF(Coupling Factor), DAC (Data Abstraction Coupling), MPC (Message Passing Coupling) |
| | Security | LOC (Lines of Code), SIZE2 (Number of Attributes and Methods), NOM (Number of local Methods), CC (McCabe Cyclomatic Complexity), WMC (Weighted Method Count), DIT (Depth of Inheritance Tree), CBO (Coupling Between Objects), CF (Coupling Factor), DAC (Data Abstraction Coupling), MPC (Message Passing Coupling) |
| Reliability | Maturity | LCOM (Lack of Cohesion in Methods), TCC (Tight Class Cohesion), LOC (Lines of Code), SIZE2 (Number of Attributes and Methods), NOM (Number of local Methods), CC (McCabe Cyclomatic Complexity), WMC (Weighted Method Count), RFC (Response For a Class), DIT (Depth of Inheritance Tree), NOC (Number Of Children) |
| | Fault-tolerance | LOC (Lines of Code), SIZE2 (Number of Attributes and Methods), NOM (Number of local Methods), CC (McCabe Cyclomatic Complexity), WMC (Weighted Method Count), DIT (Depth of Inheritance Tree), CBO (Coupling Between Objects), CF (Coupling Factor), DAC (Data Abstraction Coupling), MPC (Message Passing Coupling) |
| | Recoverability | LOC (Lines of Code), SIZE2 (Number of Attributes and Methods), NOM (Number of local Methods), CC (McCabe Cyclomatic Complexity), WMC (Weighted Method Count), DIT (Depth of Inheritance Tree), CBO (Coupling Between Objects), CF (Coupling Factor), DAC (Data Abstraction Coupling), MPC (Message Passing Coupling) |

In accordance with the selected meta-models and the specified functionality, the software application logic consists of ***three sub-systems***:

- Maintenance of meta-model for hierarchical SQMo;
- Maintenance of meta-model for SQMe;
- Maintenance of meta-model for the model of procedure (workflow).

As means of definition and implementation ***each subsystem includes:*** tools for definition of models (editor), an internal presentation and storage of defined models (in a database), and an interpretation engine (for configuration, implementation, tracking, etc..) of the so built models of SQ/metric/procedure.

 

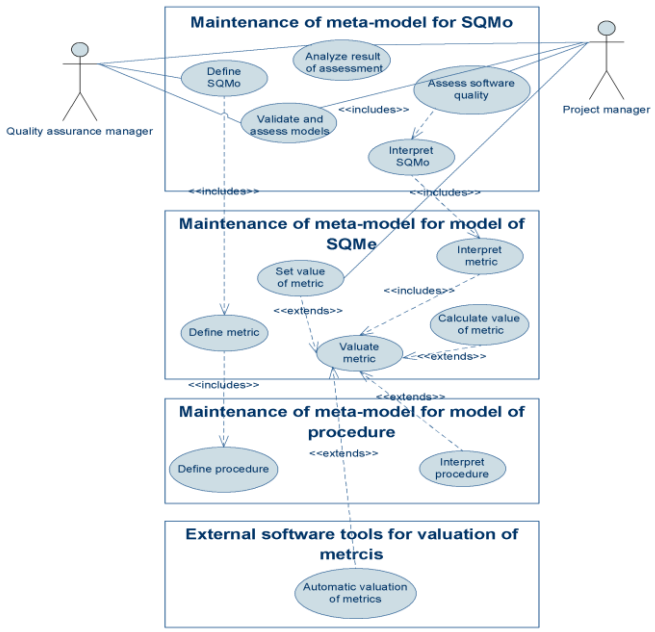


Fig. 9. Users, systems and main use cases

The proposed conceptual design adopts a simplified model of possible users/executers of the system (not taking into account system administrators, external users, etc.). There are ***two types of users***:

- `Quality assurance manager` – an expert, who is responsible for the definition of SQMo. This includes identification of its target quality characteristics and their relations, as well as development of corresponding SQMe, which can serve as quality indicators. Another task of his is, if necessary, to adapt a certain SQMo to the needs of different software projects;
- `Project manager` – is responsible for the quality assessment of the software project on the basis of the specified quality model. He analyses the assessment results, and if necessary, takes corresponding decisions (e.g. countermeasures or correction of the project plan).

***In general, the system*** is presented in Fig. 9 with its users, subsystems and the related main use cases.

In the practice of software engineering the process of quality assessment runs in similar schemes. For this reason, the designed software system automates such a process, by maintaining a typical scheme of SQ assessment. ***The typical workflow scenario*** is developed in accordance with the accepted approaches and methodologies for development of SQMo and is presented by UML sequence diagrams in Fig. 10. It includes the following ***main steps*** [3] (the software system supports the automated processing of the steps from Step 2 to Step 5, including their sub-steps).

***Step 1.*** Designing of the quality model (see fragment of SQMo presented in Table 3) for a specific software product (or a whole class of similar products). This step and its substeps are not supported by the software system because its implementation requires creative solutions by `the Quality assurance manager` and `the Project manager`;

- ***Step1.1.*** Definition of assessment purposes;
- ***Step 1.2.*** Identification of product components;
- ***Step 1.3.*** For each product component – definition of the quality characteristics, which are important factors in terms of the assessment purposes;
- ***Step 1.4.*** For each of the characteristics – selection of sub-characteristics to serve as specific criteria for the quality assessment;
- ***Step 1.5.*** Selection of appropriate metrics for the assessment of each sub-characteristic;
- ***Step 1.6.*** Setting matching limit values for each (sub)-characteristic, in order to determinate, whether the corresponding factor or criteria of the quality is achieved.

***Step 2.*** Definition of the quality model (see Fig. 3), based on the design from Step 1. Step 2 is performed automatically by the `Quality assurance manager` and is supported by the system module `Maintenance of the meta-model for SQMo`;

[3] Steps 2 and 4 are described only in general and are schematically presented in Fig. 10, because their sub steps are discussed in detail in the definition of the presented above different models based on meta-models.

 

- ***Step 2.1.*** Database search for an existing appropriate SQMo. If successful, the ready definition is extracted from the database and the workflow moves on to Step 3 or 4, if not – to Step 2.2;
- ***Step 2.2.*** Setting of values for `Model purpose`, `Viewpoint`, `Technique`, `Abstraction layer` and `Explanations`;
- ***Step 2.3.*** Modelling the hierarchical structure of characteristics and sub-characteristics of the designed SQMo (see Fig. 3). This process is supported by an appropriate structural visualization of hierarchical components. Every characteristic can be broken down into sub-characteristics, and for each of them the values of its `Attribute`, `Formula` and `Value` can be set;
- ***Step 2.4.*** Definition of a metric model, providing measurement of each specified sub-characteristic. The metric model of SQMe can be selected from the existing ones in the database or be created anew. This step uses the module `Maintenance of the meta-model for model of SQMe` (see Fig. 5);
- ***Step 2.5.*** Definition of a procedure model for each metric that serves as an instruction or as an exactly formalized definition of its valuation (optional). It includes eventually the definition of model activities (sub-steps, if any) for each procedure. The step uses the module `Maintenance of meta-model for model of procedure` (see Fig. 7);
- ***Step 2.6.*** Storing the defined quality model into the database;

***Step 3.*** Validation and assessment of the defined models (of procedures, SQMe, SQ). This step is automatically performed by `the Quality assurance manager` and the `Project manager` with the help of the modules for Maintenance `of meta-models` and their tools for appropriate visualization, interpretation (see Step 4.) and editing of models;

***Step 4.*** Interpretation of the SQMo. The `Project manager` automatically applies the defined (according to Step 2) quality model to assess the quality of a software product. This step is executed by the three main modules of the system that support visualization of the model elements in an appropriate form, and provide a convenient interface and interpretation tools. It includes a bottom-up interpretation of the defined models, which are part of the hierarchical model structure.

- ***Step 4.1.*** Interpretation / assessment of metrics (the element `Value` obtains a numeric value) in one of following four ways:
  - direct setting of the value by `the Project Manager`;
  - automated calculation of the value, on the basis of the formally specified (in declarative language) formula in the metric definition;
  - automated calculation of the value by external software tools for metrics valuation;
  - interpretation of a valuation procedure (see Fig. 8);
- ***Step 4.2.*** Automatic calculation of values of the upper level hierarchical components by given formulas and weights, on the basis of those of the lower levels;
- ***Step 4.3.*** Storing the generated quality assessment of the product into the database;

***Step 5.*** Analysis of the quality assessment results, obtained in Step 4. The current step is automatically performed by `the Quality assurance manager` and `the Project manager` with the help of the modules for `Maintenance of meta-models` and their tools for appropriate visualization (a tree-like presentation of the obtained values for the corresponding structure components down to the level of metrics). The module `Maintenance of the meta model for SQMo` also includes tools for a summary presentation of results, such as an overall quality assessment of the product, diagrams, reports, etc.

In the final design of the system the ***following additional options*** *would be recommended*:

- integration of the models visualization tools into *a full-featured editor*, which presents the built model in the form of a tree and allows editing of its components (characteristics, metrics, etc.) through an appropriate interface;
- *dynamic interface*, i.e. only those elements of the interface, which are active/can be used at the moment, should be available and visible to the users;
- definition, storing and using of *patterns* (e.g. of requirements for SQ, or of methods for development of SQMo);
- storing and tracking of *the history of assessment* for a specific software system at different stages of its life cycle;
- designing the database as a *structured repository* of developed models and assessments, grouped by well-selected categories and sub-categories (e.g. SQMo can be grouped by purpose), and thereby improve the models reusability;
- implementation of various tools for *communication, publishing and sharing* of different types of information;
- enabling *import/export* of models and data in different formats;
- including *help* and useful information for the user (e.g. information about methodologies for building of quality models)
- including a *glossary of terms*, related to SQ, which can be complemented by the user, etc.

 

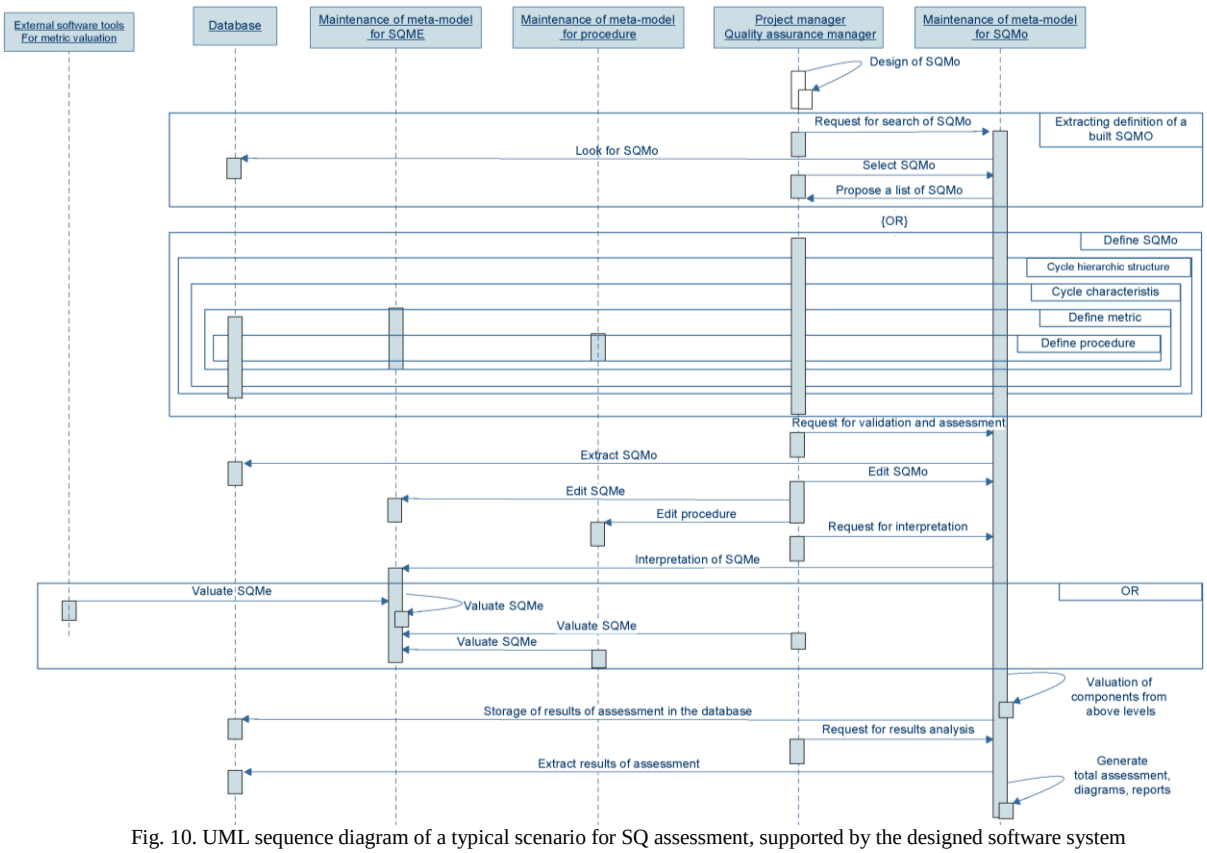


Fig. 10. UML sequence diagram of a typical scenario for SQ assessment, supported by the designed software system

As a result with the help of the designed system, for every software project an appropriate quality model can be built (consistent with the quality objectives established on the basis of the context of the project). Moreover, all previously defined models, related to the quality, can be used.

## V. CONCLUSION

This work examines the answer to the question 'Is it possible to automatically assess the SQ, in accordance with the model defined by the standard ISO/IEC 9126 and using of appropriate metrics?'

The answer, which prevails as a result of the presented meta-models and the project of a software system, is positive. It is though clear, that the proposed approach for automated SQ assessment is exposed to many ***risks***, which are hard to be controlled. For example, using inaccurate definitions of standards and models as a basis for modeling within the system, or assessing in it a wide variety of software systems with major differences, regarding their size, programming language, architecture, etc. Other significant issues arising in the implementation of SQMo by ISO/IEC 9126 and other similar models are: the abstraction of the defined SQ attributes; the characteristics of the model present mostly the developer's point of view and not that of the user of the software product; no specific instructions on how to present the overall assessment of the SQ; no indication of how to perform the measurements of the software characteristics.

Because of the above issues, in order to successfully apply the described approach and to fully benefit from the automated SQ assessment, the system users, involved in the building of the different models, have to strictly comply with the main requirements to the models that are outlined in Section 2, as well as to take account of the following ***recommendations***:

- only models, which are formalized in advance without interdependent factors should be used as a basis for modelling with the help of the system (e.g. similar to those described in [12], [15] or known frameworks (such as those of [6]);
- only validated metrics are to be used (which are sufficiently well presented in the literature);
- in case that an own SQMo is built, it is necessary:
  - the selected quality characteristics and sub-characteristics have to be significant, relevant and measurable (qualitative and quantitative);
  - an appropriate assessing scheme has to be provided;
  - the selected metrics have to generate objective assessments;
  - the metrics have to be not too complex, and should not to require a lot of resources to collect relevant information, etc.

 

The outlined objective risks don't reduce the advantages that would bring the designed in this paper tool for the software engineering professionals. ***The main advantage*** of the proposed approach for building of software systems is that it facilitates to a very large extent the quality assessment of a software product, selected by the user. The assessment is performed on a basis of a given quality model, which is designed in accordance with the quality objectives. Additionally, the work of experts is assisted, in terms of the final analysis of assessment results. But the applicability of the chosen solution is evidenced by a number of other significant ***advantages***:

- it allows the building of SQMo (through support of underlying meta-models), that:
  - provide SQ assessment based on software metrics;
  - are based on standards and best practices;
  - may be unique, i.e. built entirely according to the needs of the project manager and the specifics of the concrete product;
  - can be completely maintained by the system – from their definition, up to the analysis of the results from their application;
  - can use any set of metrics, because they are defined on the basis of a corresponding underlying meta-model and can be selected on the basis of the SQMo quality attributes, defined by the project manager;
  - it is not necessary to use metrics that have to be valued automatically, because an automated definition and implementation of procedures for assessment is supported, on the basis of an appropriate underlying meta-model;
  - can be applied repeatedly for one and the same software system, as well as for different ones. Hence, the built models are dynamic, and not static. They allow assessment of the status of the developed software product, in different phases of its life cycle, with the possibility to adapt the assessment purposes;
- it allows the data, obtained in the assessment of the software system at the different phases of its life cycle, to be used for making predictions and analysis of specific risks, related to those phases;
- the definition of models is facilitated by the ability to reuse parts of or whole previously defined models, related to the quality;
- it assists the comprehension of developed SQMo and the assessment results through their appropriate graphical presentation. In this way they can serve for improving the communication between the different participants in the assessment process (e.g. between experts in the field of the application of software systems and these in the field of the software technologies);
- it supports both the flow and control of many processes within the SQ assessment for different objectives, different software systems and organizations;
- the idea of automated building of models by the user and their interpretation (via supported meta-models) provides:
  - an easy integration of new SQMo, metrics and procedures, related to quality assessment, or restructuring of already created ones (without writing new code)
  - management of processes and minimization of errors during their execution;
  - monitoring of participating members, resources and other, that provide objectivity and transparency at the execution of processes;
- it allows a comparison between different models of quality or of SQMe, etc., which are of a similar nature.

***Additional advantages***, given by such a decision, are:

- the quality of the SQ assessment process increases;
- optimization of necessary resources – assessment processes are accelerated and costs are reduced, due to the reduction of 'manual' work;
- the typical human ability to 'forget' tasks, the lack of precision, or any omission are avoided;
- established deadlines are tracked automatically;
- the risk of errors is reduced;
- the duplication of data or efforts is avoided;

It is important to note, that the presented ideas, approaches and solutions can find a lot of other ***applications that are not related to specified tasks set in the work***, such as:

- to assess not only software products, but also their development process;
- to automate other activities related to SQ:
  - assessment and validation of metrics;
  - verification and validation of software systems and others;
- the functionality of the module, supporting procedure modelling, can be also used for the definition and management of other procedures, related to the quality, e.g. with the GQM approach for designing of SQMo and metric definition, and with even other processes in the field of the software engineering;
- with the help of the same module again it is even possible to model procedures for assessment of objects, which are not related to SQ in any way, e.g.: projects, business plans, company employees and many others.

 

The results of the study, presented in the paper, form well both a theoretical and a practical basis for the further development of a user-friendly software instrument that offers a rich set of tools for automation of the SQ assessment process. Amongst them there are tools for modelling of SQMo and corresponding metrics, which are applicable in the assessment, analysis and improvement of various software products. The development of such a product would objectify similar activities and would increase the efficiency (in the terms of time and resources).

## REFERENCES

[1] Bar H., M. Bauer et al., The FAMOOS Object-Oriented Reengineering Handbook, 1999, http://scg.unibe.ch/archive/famoos/handbook/4handbook.pdf, last accessed 15.01.2015.
[2] Basili Victor, Gianluigi Caldiera, H. Dieter Rombach, The Goal Question Metric Approach, 1994, http://www.cs.toronto.edu/~sme/CSC444F/handouts/GQM-paper.pdf, last accessed 15.01.2015.
[3] Boehm B.W., Software and its Impact: A Quantitative Assessment, Datamation, Vol.19, No.5, May 1973, p.49-59.
[4] Boehm B.W. et al., Characteristics of Software Quality, North Holland. 1987.
[5] Botella P., X. Burgués, J.P. Carvallo, X. Franch, G. Grau, J. Marco and C. Quer, ISO/IEC 9126 in practice: what do we need to know?, in Software Measurement European Forum 2004, pages 297-306 Rome, January 2004, http://www.essi.upc.edu/~webgessi/publicacions/SMEF'04-ISO-QualityModels.pdf, last accessed 15.01.2015.
[6] Cavano J., McCall J., A framework for the measurement of software quality, SIGSOFT Softw. Eng. Notes 3, 5 (January 1978), 133-139, http://portal.acm.org/citation.cfm?id=811113, last accessed 15.01.2015.
[7] Cote Marc-Alexis, Suryn W., Georgiadou E., Software quality model requirements for software quality engineering, In: Software Quality Management & INSPIRE Conference (BCS), 2006, http://profs.etsmtl.ca/wsuryn/research/SQE-Publ/Quality%20model_requirements.%20SQM2006.pdf, last accessed 15.01.2015.
[8] Doneva R., Z. Doneva, E. Hadzhikolev, S. Hadzhikoleva, G. Totkov, Automation of Procedures for Evaluation of the Quality in Higher Education (Based on a Meta-Model), Proc. Of 4th International conference on e-learning in higher education, 11 to 13 May 2012, Svishtov, ISBN: 978-954-23-0747-1, 221-228.
[9] Erlikh L., Leveraging Legacy System Dollars for E-Business, IT Pro, pages 17-23, May 2000.
[10] Hyatt L., L. Rosenberg, A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality, 8th Annual Software Technology Conference, Utah, April, 1996.
[11] ISO, ISO/IEC 9126-1 "Software engineering – Product Quality – Part 1: Quality model", 2001.
[12] Khaddaj S., Gerard Horgan, A Proposed Adaptable Quality Model for Software Quality Assurance, Journal of Computer Sciences, 1: 482-487, 2005, http://www.thescipub.com/pdf/10.3844/jcssp.2005.482.487, last accessed 15.01.2015.
[13] Kumar A., B. S. Gill, Maintenance vs. Reengineering Software Systems, Global Journal of Computer Science and Technology, Volume 11, Issue 23, Version 1.0, December 2011, Global Journals Inc. (USA), Online ISSN: 0975-4172 & Print ISSN: 0975-4350.
[14] Lee Y., Chang K., Reusability and maintainability metrics for object-oriented software, ACM Southeast Regional Conference'2000. pp.88-94, http://portal.acm.org/citation.cfm?id=1127737, last accessed 15.01.2015.
[15] Lincke R., W. Löwe, Validation of a Standard- and Metric-Based Software Quality Model, 10th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, 2006.
[16] L. Ott, J. Bieman , B-K Kang, and B. Mehra. Developing measures of class cohesion for object-oriented software, Proc. Annual Oregon Workshop on Software Metrics (AOWSM'95), June 1995, http://www.cs.colostate.edu/~bieman/Pubs/aowsm95.pdf, last accessed 15.01.2015.\
[17] Ogheneovo E.E., On the Relationship between Software Complexity and Maintenance Costs. Journal of Computer and Communications, 2, 1-16. http://dx.doi.org/10.4236/jcc.2014.214001, 2014.
[18] Sarker M., An overview of Object Oriented Design Metrics, Umea University, Sweden, 2005, www8.cs.umu.se/education/examina/Rapporter/MuktamyeSarker.pdf, last accessed 15.01.2015.
[19] Technische Universität München, Continuous Quality Assessment ToolKit, https://www.cqse.eu/en/products/conqat/overview/, last accessed 15.01.2015.
[20] Wagner S., F. Deissenboeck, An Integrated Approach to Quality Modelling, WoSQ '07: Proc. of the 5th Workshop on Software Quality, 2007, http://www4.in.tum.de/~deissenb/publications/2007_wagners_integrated_quality_model.pdf, last accessed 15.01.2015.
[21] Weller Thomas, It's the maintenance, stupid!, http://geekswithblogs.net/ thomasweller/archive/2010/01/27/its-the-maintenance-stupid-or-something-is-rotten-in-developerland.aspx, last accessed 15.01.2015.
[22] Wikipedia, Cyclomatic complexity, http://en.wikipedia.org/wiki/ Cyclomatic_complexity, last accessed 15.01.2015.
[23] Wikipedia, Halstead complexity measures, http://en.wikipedia.org/ wiki/Halstead_complexity_measures, last accessed 15.01.2015.
[24] S. Williams, Business Process Modelling Improves Administrative Control, In: Automation, December, 1967, pp. 44-50.

 