The SQALE Models for Assessing the Quality of Real Time Source Code

Jean-Louis LETOUZEY[1], Thierry COQ[2]

1: DNV IT Global Services, Arcueil, France, jean-louis.letouzey@dnv.com
2: DNV IT Global Services, Arcueil, France, thierry.coq@dnv.com

Abstract: We need standardized methods to objectively evaluate the quality of an application's source code. The hope is that over time, standards for doing such assessments will emerge, generalizing their use. Aiming to contribute to this standardization process, we present in this paper two elements of the SQALE (Software Quantitative Assessment based on Lifecycle Expectations) method used by DNV IT Global Services: the Quality Model and the Analysis Model.

By construction, the SQALE Quality Model has two main advantages: it is organized in layers and can be understood using two different viewpoints: the point of view of the developer and the point of view of the owner or user. The SQALE Analysis Model is based on the concept of a remediation index. The purpose is to objectively rate and quantify the characteristics of a piece of software. A concrete example of the joint use of the two models is presented here.

Keywords: Source Code, Quality Model, Analysis Model, evaluation, assessment

1. Introduction

The need to assess and know the quality of the software one has required or paid for is not new. For example, it should be possible to know how the code will accept future evolutions, or whether it will be easily possible to outsource its maintenance to a third party team. Although some large customers have developed their own methods, or even their own tools to satisfy such assessment needs, regularly assessing the delivered software has not been generalized. This situation is partly the result of a lack of an accepted standardized method allowing systematic assessments to be performed within reduced schedules and costs. This absence of methods to assess software is also felt by software development managers.

A software development organization's maturity level and the capabilities of its processes may be assessed and measured on a scale as provided by a standard model such as CMMI [1, 2]. However, as far as the software product resulting from these processes is concerned, there is no complete and calibrated system to assess and rate its quality.

The two main components of an assessment method are on the one hand: a quality model, and on the other hand: an analysis model.

- The quality model defines the quality characteristics (or attributes), expected in the context of the assessment. Characteristics are broken down into detailed sub-characteristics. Such a breakdown allows the definition of precise controls to be set up for the product assessment.
- The analysis model contains the rules to characterise or rate the products (or components thereof) based on measurements obtained on the control points of the quality model. This is the "analysis model" defined in standard ISO/IEC 15939 [3].

Therefore DNV has developed the SQALE method and models to satisfy the needs in assessing strategic software source code. We describe hereafter the two SQALE models to contribute to the collective work necessary to build a common assessment standard. These models have been developed with the goal to be automated with static analysis tools.

2. The SQALE Quality Model

Compared to other Quality Models [4, 5, 6], this model has four important properties:

- It has been built in a systematic manner that takes into account the lifecycle of source code.
- It is structured in hierarchical layers.
- It is a requirement model and not a set of best practices to implement.
- It takes into account both the developer's point of view and the software user's point of view.

To define the basic quality characteristics of our model, an approach based on the typical lifecycle of a source code file was used.

This approach organizes sequentially the requirements and needs for source code quality as represented in figure 1. In this sequence, testability is the first characteristic and reusability is the last one required.

The layered model presented in figure 2 is the end result of this approach. This basic model may be adapted by taking into account the expected lifecycle for the source code to be assessed.

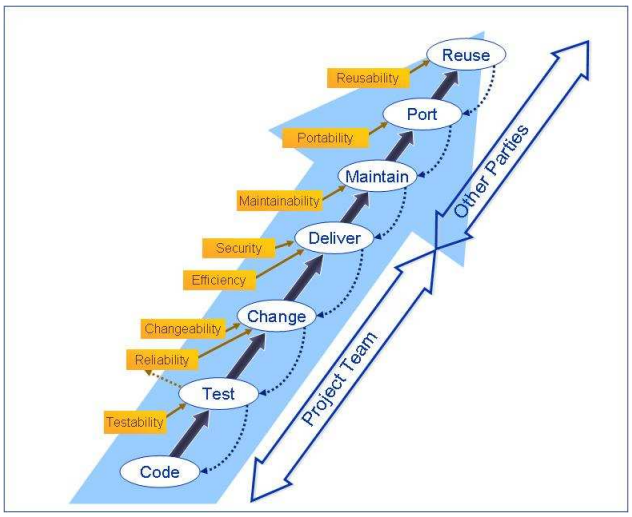

Figure 1: The "File" Lifecycle used for the SQALE Quality Model. Dependencies between Activities and Quality Characteristics.

The standard Goal/Question/Metrics (GQM) from Basili [7] was applied to build all additional sublevels. In this structure, a sub-characteristic or a requirement used in the model is never duplicated in any higher level. It is used only once in the model. Using a control point and a sub-characteristic only once is an essential rule of the SQALE model, which must be followed at all times.

Figure 2: The first level of the generic SQALE Quality Model

The sub-characteristics Testability or Changeability traditionally associated to Maintainability in existing models are now first level characteristics of the SQALE model. It is a consequence of the fact that these characteristics are needed earlier in the lifecycle of a source code file.

All the levels of the SQALE Quality Model are organized as described in figure 3.

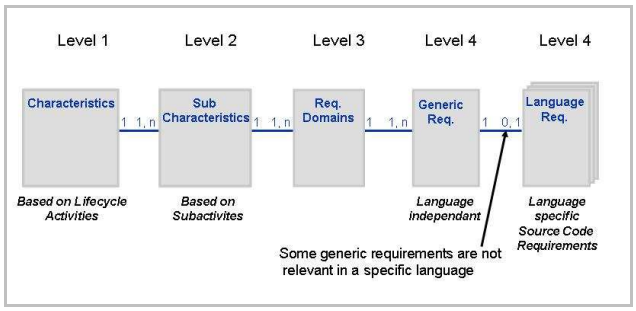

Figure 3: Structure of the SQALE Quality Model

A particular property of the SQALE quality model is that it is a requirements model. Each lower level leaf of the model is a quality requirement to be satisfied. This is in accordance with Ph Crosby's first "absolute concept" [7]: "Quality must be defined as conformance to requirements. That means that all requirements are atomic, consistent, non redundant and verifiable.

As we mentioned before, the generic SQALE Quality Model must be instantiated for a given context and language as some requirement could be not relevant for a given language or some high level characteristic not selected as quality goal for a given application. We provide in figure 4 an extract of our quality model regarding the reliability characteristics for the C++ language.

Req. Domain	Requirement
Data	There is no malloc
Data	There is no implicit sign extension
Data	There is no potential table overflow
Data	There is no potential division by zero
Data	There is no illegal shift
Data	There is no unitialized return value
Data	There is no function with a variable numbers of arguments
Data	There is no use of freed memory
Data	There is no unitialized variable
Fault Tolerance	All final clause of a switch statement is a default clause
Computation	There is no comparison between floating point
Computation	There is no instruction with loss of precision
Computation	The value of an expression shall be the same under any order of evaluation that the standard permits
Logic flow	All functions have a single point of exit at the end of the function
Logic flow	There is no infinite loop
Logic flow	There is no iteration variables modified in the body of a loop
Resource	There is no memory Leak
Synchronization	There is no variable declared in a protected section used outside the section

Figure 4: Some SQALE Quality Model requirements associated to the reliability layer (extracted from our complete C++ quality model)

The last property of the SQALE Quality Model is its capacity to support two different points of view, the developer point of view and the owner/user point of view, as described in figure 5.

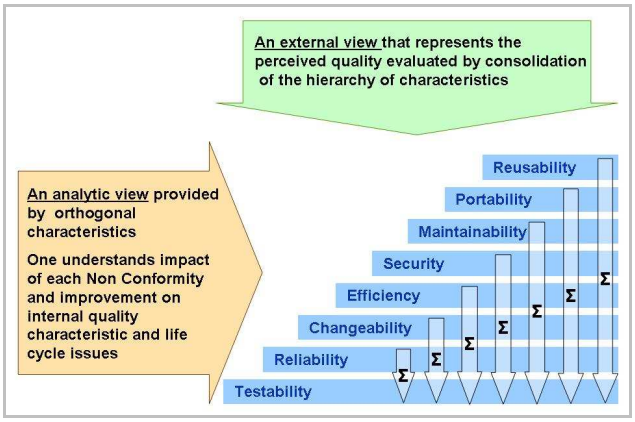

Figure 5: The two viewpoints of the SQALE Quality Model

3. The SQALE Analysis Model

As stated previously, the SQALE quality model is a requirements model. The way it has been built ensures the quality targets a total absence of non compliances. As written by Ph. Crosby, assessing a software source code is therefore similar to measuring the distance which separates it from its quality target. To measure this distance, the concept of remediation index has been defined and implemented in the SQALE analysis model.

An index is associated to each artefact of the software source code (for example, a file, a module or a class). The index represents the remediation effort which would be necessary to correct the non compliances detected in the component, versus the model requirements.

Since the remediation index represents a work effort, the consolidation of the indices is a simple addition of homogenous information, which is a critical advantage of our model.

A component index is computed by the simple addition of the indices of its elements. A characteristic index is computed by the addition of the base indices of its sub characteristics. A sub-characteristic index is computed by the addition of the indices of its associated control points.

Remediation effort depends on the type of violation.

For example correcting a presentation defect (bad indentation, dead code) does not have the same unit effort cost as correcting an active code defect (which implies the creation and execution of new unit tests, possible new integration and regression tests). In the end, the remediation indices provide a means to measure and compare non compliances of very different origins and very different types. Coding rule violation non-compliances, threshold violations for a metric or the presence of an antipattern non-compliance can be compared using their relative impact on the index.

When implementing the SQALE Analysis Model, a remediation function is associated to each basic requirement of the model. The parameters of this remediation function depend on the typology of the remediation of the non conformance. Following static analysis results, a “Non conformance table” is then transformed into a “Remediation cost table”. Then it is easy to aggregate row or column to build indices as described in figure 6.

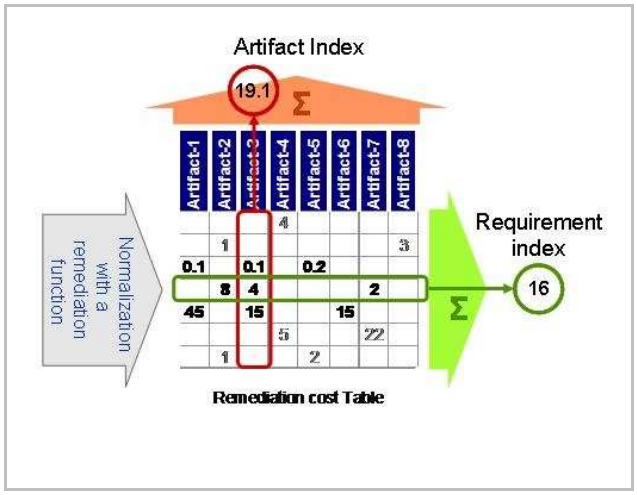

Figure 6: Index consolidation into the two hierarchies; the quality model hierarchy and the artefact hierarchy.

4. Practical Use

The SQALE quality and analysis models have been used to perform many assessments of software source code, of various sizes and in different application domains. The same layered generic Quality Model has been used to assess Cobol, Java, embedded Ada, C or C++ software source code.

Let’s take an example of a piece of a piece of real-time software, (anonymized), possible used in the telecom industry. This piece of software is coded in C and C++, over several years, is broken down in several parts, some of which are considered generic and reusable by the software team (part A and B), some of which are specific to the application (part C and D). This application’s objective quality is assessed using the SQALE models, in order to derive a set of actions to plan the future of this application.

The process of measuring the SQALE quality of a piece of software is as follows:

1. define the scope of the assessment, in this case the application, with the reusable parts included,
2. tailor the SQALE quality model, with a selection of the control points and a review of the control point's threshold,
3. set up the SQALE automated measurement tool chain and qualify the measurement chain using test cases,
4. gather the source code of a given configuration of the application, a fully compilable and linkable set of software source code should be made available
5. perform a first analysis of the available source code, check the whole software is analyzed, verify preliminary results by human inspection on a few selected cases (worst cases and best cases to check for false positives and false negatives)
6. Compute the aggregate indices and analyse the results to identify the main issues and recommended actions
7. Present the results to the team for improvement.

Some steps are described further below, where needed.

In step 1, the scope of the assessment is defined. In this case, the team describes the software as being composed of two reusable and reused parts and two application-specific parts.

The two reused parts come from different origins. One of the two application-specific parts has been subcontracted.

In step 2, The SQALE model used here is limited to 5 layers. Its quality model for this application is adapted to the C/C++ implementation. In particular, object-oriented metrics are taken into account to assess Testability, Changeability, Maintainability and Reusability. The quality model is proposed by the SQALE assessment team and reviewed by the application software team's representatives and managers. A consensus is obtained to be sure the model is consistent with the team's quality objectives.

In step 3, a set of tools is selected to perform the base measures. Various tools are used to precisely measure line count (including comments and mixed lines), compute metrics and measure copy/paste. Such tools need to be qualified to ensure that they do indeed measure the control points as expected by the SQALE quality model. In addition, tools must be checked against a selection of code to ensure they are capable of performing the measurement. For example, in this case, the application's size is several million lines of code, and some tools' performance degrades when the size increases.

In step 5, a human inspection confirms the preliminary results obtained automatically, and allows the assessment team to obtain visual evidence of the worst and best cases. For example, a piece of code might exhibit an operation with a V(G) exceeding a hundred, where the V(G) is the result of a set of statements, and not a simple "switch" statement. Additional examples may present a (long) piece of code without any comments, duplicated code (sometimes over several pages long) or other such large non compliances to the quality model.

In step 6, the indices are aggregated by characteristics and sub-characteristics, by file and application part, and finally aggregated to the whole application. Two sets of indicators are provided:

One set of indicators provide detailed quality status reports, including the top 10 for each characteristic.

The second set of indicators enables detailed analyses based on the part type, the team, etc to be conducted, allowing justified decisions to be taken.

For example, the testability of the various parts (see figure XX) shows how the reusable parts (A and B) are more testable than the specific parts of the application.

Index densities are computed based on the size (SLOC and complexity) of the application.

Ratings are computed based on the following table (see figure 7). The remediation effort resulting from the aggregate indices for each file is compared to the effort estimated by the team to develop this file from scratch. A five-level rating system is used: above 30% of remediation effort, the quality of the file is rated at red; below 0.9, the quality is rated at green. Intermediate colours (light green, yellow, orange) are used for intermediate values based on a logarithmic scale [0 – 0.9 – 3 – 9 – 30 – +∞ [.

From	To	Rating	Color
<	0.9	A	
0.9	3	B	
3	9	C	
9	30	D	
30	>	E	

Figure 7: Rating table

The ratings for the application are presented below (see figure XX). The application's reusable code presents overall better quality with issues remaining in testability and maintainability. However specific code presents larger issues with testability, especially the newer part.

Step 7 presents the results of the SQALE assessment.

The size of the application is first presented in lines of code and in complexity (decision points). The indices and index densities of each characteristic is then presented, as well as some illustrations of the top ten worst and best cases. The overall indices and rating are finally presented. The buy-in at this stage is achieved; the team recognizes its results. The recommendations for improving the software can then be presented, usually in terms of refactoring. Quick wins and base characteristic (Testability) improvements are promoted, leading to visible improvements to the software quality. Generally, improvements should only be performed on a deployed code base when another change (bug correction, functional improvement) warrants it.

In this particular example, the improvements should be focussed on the application-specific parts, particularly on the testability, reliability and changeability characteristics. The maintainability non compliance corrections should be deferred. The subcontractor should be requested to improve its processes and the quality of its product to match the results of the internal team. Training to code quality should be provided to its development team.

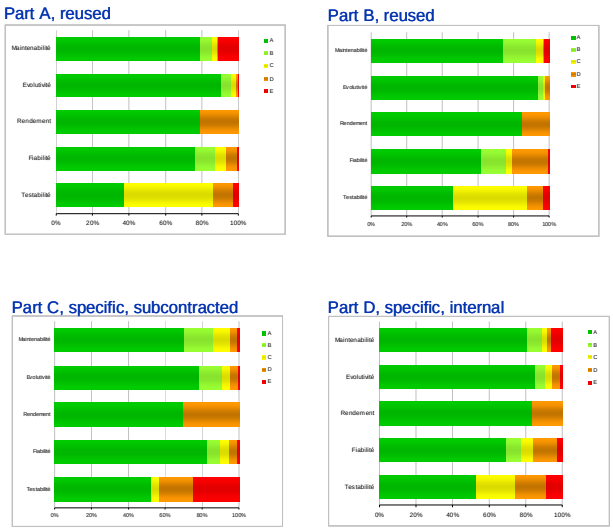

Figure 8: SQALE assessment results, by part and characteristic

These recommendations are not new, of course, but the SQALE measurement method provides justification for the recommended course of action. For example, it also provides justification for the well-known fact that some teams may spend large amounts of time on so-called quality and not seeing the objective results, when they are improving maintainability issues by improving the comment ratio and not addressing the base testability issues.

During the improvements, the SQALE models can be used to measure the progress, in particular by integrating the measurement process within the build process.

5. Conclusion

More than just a quality model, we propose a standard approach for developing quality models derived from the software lifecycle. This approach enables us to precisely specify the expectations on software source code. The resulting models remove several weaknesses in current reference quality models. The quality models implemented using the SQALE method present two similarities with the CMMI process model:

- On one hand, like the CMMI, they organize the requirements in levels organized by priority.

- On the other hand, like the CMMI model which may be used to assess capability or maturity, the SQALE quality models may be used in an analytical manner or from the point of view of an owner or user of the source code. Such similarities of the models and the assessment methods allow us to perform joint product/process assessments in a fast and efficient

manner with high added value results to the development or owner teams.

5. References

[1] SEI, *CMMI for Acquisition Version 1.2*, CMU/SEI 2007-TR-017, Carnegie Mellon University, Pittsburg,2007

[2] SEI, *CMMI for Development Version 1.2*, CMU/SEI-2006-TR-008, Carnegie Mellon University, Pittsburgh, 2006

[3] ISO/IEC, International Standard, «ISO/IEC 15939: 2007 (E), *Systems and software engineering –Measurement process*, 2007

[4] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merrit, M., *Characteristics of Software Quality*, North Holland, 1978

[5] McCall, J. A., Richards, P. K., and Walters, G.F., *Factors in Software Quality*, The National Technical Information Service, No. Vol 1, 2 and 3, 1977

[6] ISO, International Organization for Standardization, «9126-1:2001, *Software engineering – Product quality, Part 1: Quality model*», 2001

[7] Victor Basili, *Software Modeling and Measurement: The Goal/Question/ Metric Paradigm*, Technical report. University of Maryland at College Park, 1992

[8]. Crosby, P. B., *Quality is free*: the art of making quality certain, New York: McGraw-Hill, 1979