The SQALE Method

Definition Document

Author: Jean-Louis Letouzey

Version: 1.0

January 27, 2012

Table of Contents

List of Figures

1. Introduction

1. Version

This document describes Version 1.0 of the **SQALE** Method (**Software Quality Assessment based on Lifecycle Expectations**). The latest version of it can be accessed on the website at http://www.sqale.org

2. Ownership and User Licence

This method is the intellectual property of **inspearit** (previously named DNV IT GS France).

inspearit declines to consider any software implementation of the method as a derived work. Software editors can freely implement the SQALE method in their tools, whether they are licensed commercially or as open source.

inspearit declines to consider any usage of the method as a derived work.

3. Scope and Objective

As the method's name specifies, the SQALE method relates to the evaluation of software quality. This SQALE Method can be applied to all deliverables and all representations of the software, such as UML models. The scope of the version described in this document is limited to the evaluation of so-called source files.

The objective of this version of the SQALE Method is to support the evaluation of a software application's source code in the most objective, accurate, reproducible and automated way possible.

This document presents the quality model, the analysis model, the indices and the indicators.

4. Limitations

This document focuses on the description of the method and its concepts. Other related documents cover or will cover subjects such as the precise definition of certain measures used, the recommended code evaluation method and the manner of interpreting the indices and the SQALE indicators.

Similarly, it does not describe the reasons that have led to its development nor the explanations concerning the justification of certain technical choices and the advantages that can be drawn from them. This information, at least partially, has already been presented in the documents referenced below [Ref. 1 to 3].

It is not seeking to describe how the SQALE Method should be implemented in tool form. Users and tool editors alike are free to choose or to construct a solution implementing the method described in this document.

5. Reference Documents

Ref. 1	J-L Letouzey, Th. Coq, The SQALE Models for Assessing the Quality of Software Source Code, DNV IT GS France, Paris, white paper, September 2009
Ref. 2	J.-L. Letouzey, Th. Coq, The SQALE Models for Assessing the Quality of Real Time Source Code, ERTSS 2010, Toulouse, May 2010
Ref. 3	J.-L. Letouzey, Th. Coq, The SQALE Analysis Model - An analysis model compliant with the representation condition for assessing the Quality of Software Source Code, VALID 2010, Nice, August 2010
Ref. 4	ISO, International Organization for Standardisation, 9126-1:2001, Software Engineering – Product Quality, Part 1: Quality Model, 2001
Ref. 5	D. Spinellis, Code Quality: The Open Source Perspective, ISBN: 0-321-16607-8, Addison-Wesley, 2006
Ref. 6	N.E. Fenton, S. L. Pfleeger, Software Metrics: A Rigorous & Practical Approach, Second Edition, ISBN 053495425-, PWS Publishing Company, Boston, 1997
Ref. 7	Th. McCabe, A. H. Watson, Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric, National Institute of Standards and Technology, Special Publication 500-235, 1996
Ref. 8	M.H. Halstead, Elements of Software Science. New York, Elsevier North-Holland, 1977

2. Fundamental Principles

The SQALE Method is based on the following nine fundamental principles:

1. The quality of the source code is a non-functional requirement.

   A software development or maintenance project has objectives that have to be achieved. These relate to deadlines, costs, functionality and quality. In order to be achieved, these objectives have to be formalised. The formalisation of the objectives is translated into requirements. Those that relate to the quality of the source code belong to the so-called non-functional requirements.

2. The requirements in relation to the quality of the source code have to be formalised according to the same quality criteria such as any other functional requirement.

   Consequently, a quality requirement concerning a software source code must at least be:

   - Atomic
   - Unambiguous
   - Non-redundant
   - Justifiable
   - Acceptable
   - Implementable
   - Not in contradiction with any other requirement
   - Verifiable

3. Assessing the quality of a source code is in essence assessing the distance between its state and its expected quality objective.

   Being that the objective is to meet the source code quality requirements; the measure of the quality of a source code can be reduced to the measure of the distance that separates it from the targeted conformity.

   Although other conceptions and other definitions of quality exist, it is this definition that the SQALE Method takes into account for an assessment of the source code.

   Comment:

   As other conceptions and definitions are more subjective, the adopted definition provides better support for automated assessment of the code's quality.

4. The SQALE Method assesses the distance to the conformity with the requirements by considering the necessary remediation cost of bringing the source code to conformity.

5. The SQALE Method assesses the importance of a non-conformity by considering the resulting costs of delivering the source code with this non-conformity.

6. The SQALE Method respects the representation condition.

   The SQALE Method has been designed by respecting this condition (see a definition of this condition in the appendix). This impacts the choice of the requirements, their organisation in the Quality Model and the aggregation rules.

7. The SQALE Method uses addition for aggregating the remediation costs, the non-remediation costs and for calculating its indicators.

8. The SQALE Method’s Quality Model is orthogonal.

   The SQALE Method uses an orthogonal Quality Model. This means that a requirement relating to one of the code’s internal attributes appears only once in the Quality Model. A requirement is linked to only one quality subcharacteristic. A subcharacteristic is linked to only one quality characteristic.

9. The SQALE Method’s Quality Model takes the software’s lifecycle into account.

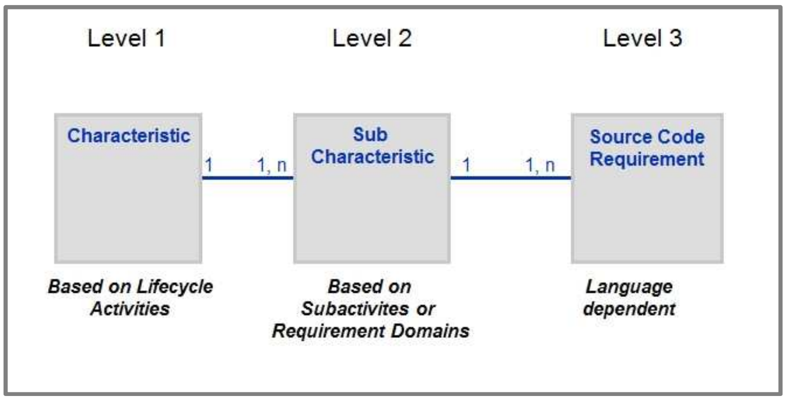
   The SQALE Method’s Quality Model’s characteristics, subcharacteristics and requirements are organised in such a way as to reflect the chronology of the needs such as they appear in a software’s lifecycle.

3. The SQALE Quality Model

1. General Structure

The SQALE Quality Model is used for formulating and organising the non-functional requirements that relate to the code's quality. It is organised in three hierarchical levels which are represented in Figure 3.1. The first level is composed of characteristics, the second of subcharacteristics. The third level is composed of requirements that relate to the source code's internal attributes. These requirements usually depend on the software's context and language.

Figure 3.1. The SQALE Quality Model's General Structure



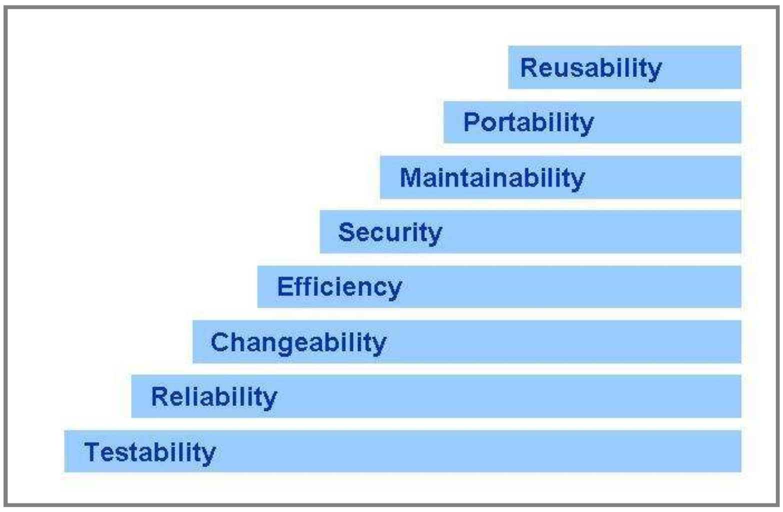
2. The Characteristics Level

The SQALE Quality Model has eight Level 1 characteristics, which are presented in Figure 3.2. They have been deduced from the theoretical cycle of a source file [Ref. 1].

These characteristics are "abilities" resulting from the ISO 9126 standard [Ref 4]. They have been selected, on the one hand because they depend on the code's internal properties, and on the other because they directly impact the typical activities of a software application's lifecycle. We have added "Reusability", which was missing from the ISO 9126 model.

The SQALE Quality Model can therefore be regarded as a projection of the ISO 9126 model in the chronology of a software application's lifecycle.

Figure 3.2. The SQALE Quality Model's Characteristics



3. The Subcharacteristics Level

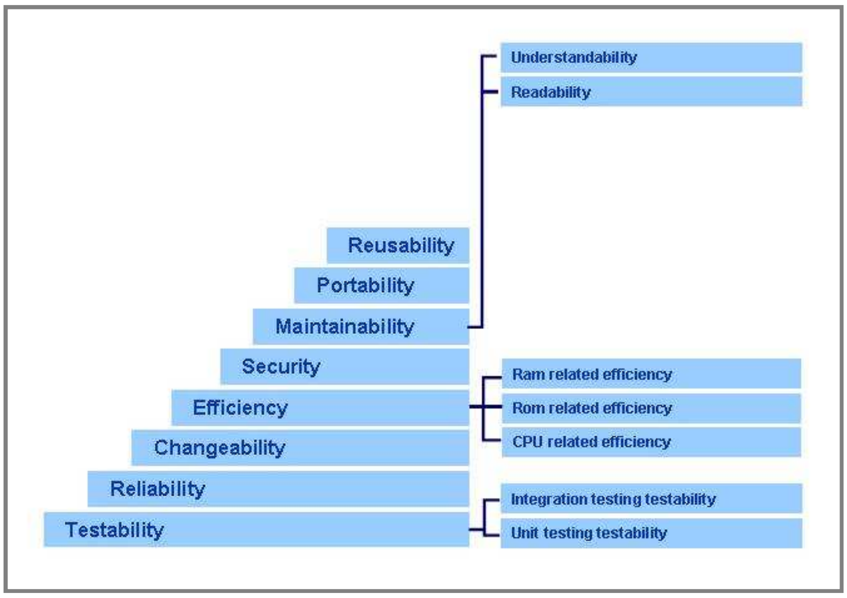
Each characteristic is broken down into subcharacteristics. Subcharacteristics are used to combine the requirements into medium-sized groups so that “drill down” analyses can be carried out.

There are two types of subcharacteristic:

- Subcharacteristics corresponding to lifecycle activities such as unit test, integration test, optimisation of processor usage and optimisation of the size of the generated code. These are represented in Figure 3.3. They are presented in their chronological order.

- Subcharacteristics resulting from generally recognised taxonomies in terms of good and bad practices relating to the software’s architecture and coding. This classification is an implementation choice. Inspiration can be drawn from existing classifications such as the one carried out by D. Spinellis [Ref 5].

Figure 3.3. The Subcharacteristics Resulting from Lifecycle Activities



A subcharacteristic is attached to only one characteristic, the first in the chronology of the characteristics.

The appendix contains an example of characteristic and subcharacteristic hierarchy.

4. The Requirements Level

This level of the model contains all of the source code’s quality-related requirements. They are formulated by respecting the quality criteria (atomicity, clarity, and so on) presented in the Fundamental Principles section.

Requirements relate to the artefacts that compose the software’s source code, e.g. software applications, components, files, classes, and so on and so forth.

Each of the requirements will be attached to the lowest possible level, i.e. in relation to the first quality characteristic to which it chronologically contributes.

For example, let us suppose that we are expressing a requirement relating to cyclomatic complexity [Ref. 7]. This internal property impacts the unit test efforts (therefore the Testability) and the understanding (therefore the maintainability). We will attach the requirement to the first subcharacteristic, which is the unit test and therefore to the testability.

A typical requirement, would be to be within a threshold limit for a structural measure, or the respect of a syntactical rule. We provide as an appendix some Quality Model examples for various languages.

5. SQALE Quality Model Tailoring

To preserve all of the SQALE Method's properties and benefits, its Quality Model adaptation and customisation possibilities are limited, these being the following:

5.1. Tailoring the Characteristics List

In a context that justifies it, it is possible to regard certain characteristics as non-applicable. In this case, the associated subcharacteristics and requirements are deleted from the Quality Model.

The characteristics which can be regarded as non-applicable are the following:

- Efficiency
- Security
- Maintainability
- Portability
- Reusability

In every case, the order of the characteristics must remain unchanged in relation to the SQALE Quality Model presented in Figure 3.2.

5.2. Tailoring the Subcharacteristics List

For the subcharacteristics resulting from the lifecycle activities and if the context justifies it, some of them can be deleted if they correspond to a non-applicable activity in the context of the software that is being assessed. For example, the optimisation of the ROM memory.

For the other subcharacteristics (corresponding to requirement categories), some of them can be deleted, reorganised or added according to the context of the software that is being assessed. The guideline is to obtain subcharacteristics that group the requirements according to logic that contributes to the analysis, the understanding or the remediation of nonconformities.

5.3. Tailoring the Requirements List

All of the requirements represent the software's source code's quality objectives. They therefore have to be adapted according to the expectations and needs of the project or organisation.

To do so, they have to be attached to one subcharacteristic of the model in compliance with the previously stated principles.

4. The SQALE Analysis Model

1. Structure

The SQALE Analysis Model contains on the one hand the rules that are used for normalising the measures and the controls relating to the code, and on the other the rules for aggregating the normalised values.

The SQALE Method normalises the reports resulting from the source code analysis tools by transforming them into costs. To do this remediation functions and non-remediation functions are used. These concepts are described in the following paragraphs.

The SQALE Method defines rules for aggregating costs, either in the Quality Model's tree structure, or in the hierarchy of the source code's artefacts.

2. Remediation Functions

When a Quality Model's requirement is formulated as a rule that has to be obeyed, the associated measure, collected by the analysis tools, is the detected number of violations per artefact.

The remediation function's goal is to normalise findings related to one requirement into remediation costs. This normalization is performed from the technical team point of view.

In order to do this normalisation, one can simply use a multiplicative factor that corresponds to the average remediation cost unit for bringing the code into conformity. The value of this factor will depend on the activities that have to be carried out in order to remedy the non-conformity.

This value can be very low in the case, for example, where it is a question of a violation relating to one of the code's formatting requirements. The remediation can be carried out by an IDE script or functionality. As this change of the code does not impact the code that is generated, it will inevitably not require a verification and validation cycle.

This value can, in other cases, be much higher because the code will have to follow a verification and validation cycle that is both complete and costly. It is for example the case when the code's structure or architecture has to be changed in order to make it conform to the Quality Model.

In order to get a more precise estimation of the remediation cost, it possible to normalize with a more complex function that use as parameters some code's internal attributes as well as the measured value. This is particularly the case when a Quality Model requirement consists of keeping a measure within a certain interval (example: a function must have a cyclomatic complexity below 15)

The set of remediation functions associated to a Quality Model constitutes a "**Technical Debt"** estimation model.

In order to respect the representation condition, a remediation function must:

- Be strictly monotonic on the interval (or the intervals) corresponding to the non-conforming values.

- Be constant or equal to 0 on the interval (or the intervals) corresponding to the conforming values.

Remediation function examples are given in the appendix.

3. Non-remediation Functions

The decision to fix a non-conformity has an impact on the development plan and an associated remediation cost. Symmetrically, the decision to keep and deliver a non-conformity has a negative impact on the business plan. This negative impact has an associated cost which is the non-remediation cost of the non-conformity.

The non-remediation function’s goal is to normalise findings into non-remediation costs. This normalization is performed from the Business or Product Owner point of view.

The set of non-remediation functions associated to a Quality Model constitutes a “**Business Impact”** estimation model.

This Business Impact should represent all the damages inferred by the non-conformity, damages that can be numerous. This includes, but is not limited to:

- A more expensive remediation cost that need to be spent in a later phase of the lifecycle
- A more expensive maintenance cost (enforced by example by duplicated code)
- Additional resource costs, such as CPU power, memory, disk space (enforced by example by violations of efficiency related requirements)
- Extra man power to operate the software (longer installation, configuration, back-ups…)
- Costs for investigating and fixing a bug resulting from a non-conformity
- A provision to cover the related risks of leaving non-conformities

All these inferred costs may be difficult to estimate and to model within a non-remediation function. Relevant stakeholders should be involved in the estimation of the non-remediation cost to each requirement, for example maintenance team (cost of bugs, cost of lack of comments, cost of duplicated code…), production team (cost of extra hardware), business team (cost of unavailability, cost of low performances…). It is possible to use a global approach based on the penalty concept. It is the penalty that the product owner will ask as a compensation for delivering the product with a violation. Below this amount, he/she won’t accept the violation. For this amount, he/she considers that the penalty is an acceptable compensation and that it covers all the damages, of all types, that result or may result from the violation.

The set of non-remediation functions shall represent, from the Product Owner point of view, the relative importance of the different non conformities that developers may leave in the source code.

In order to respect the representation condition, a non-remediation function must:

- Be strictly monotonic on the interval (or the intervals) corresponding to the non-conforming values.

- Be constant or equal to 0 on the interval (or the intervals) corresponding to the conforming values.

4. Aggregation Rule

Within the SQALE method, all aggregations are made by additions. This applies to remediation costs and non-remediation costs.

This is the case in the Quality Model’s requirement hierarchy. By application of the principle that in order to estimate the remediation or non-remediation cost of a characteristic or a subcharacteristic, all the estimates of the costs of the related non-conformities will be added.

This is also the case in the artefact hierarchy of the source code. In order to estimate an artefact’s remediation or non-remediation cost, the estimated costs for all of its constituent elements have to be added.

5. Tailoring the Analysis Model

5.1. Tailoring the Remediation Functions

Remediation costs are depending on the local rules and processes of the organisation or project concerned. The remediation functions can be tailored in order to correspond as closely as possible to the real remediation costs for the evaluated software. This customisation must be carried out in compliance with SQALE’s fundamental principles.

5.2. Tailoring the Non-remediation Functions

Non-remediation costs are depending on the local rules, processes and business of the organisation or project concerned. The non-remediation functions can be tailored in order to correspond as closely as possible to the real non-remediation costs for the evaluated software. This customisation must be carried out in compliance with SQALE’s fundamental principles.

5.3. Tailoring the Aggregation Rules

The SQALE aggregation rules cannot be tailored.

5. The SQALE Indices

All of the SQALE indices represent costs. All the indices are measured in relation to the same unit, which is either a monetary unit, a work unit or a symbolic unit. In every case, the indices have values on a scale of the ratio type. They can therefore be manipulated with all the operations that are authorised for a scale of this type [Ref. 6, Page 57].

1. Absolute Indices

1.1. Characteristic Indices

For any element of the artefact hierarchy of the source code's, the remediation cost relating to a given characteristic can be estimated by adding up all of the remediation costs linked to the detected non-conformities noted with regard the requirements linked to the characteristic.

The SQALE Characteristic Indices are the following:

SQALE Testability Index: STI

SQALE Reliability Index: SRI

SQALE Changeability Index: SCI

SQALE Efficiency Index: SEI

SQALE Security Index: SSI

SQALE Maintainability Index: SMI

SQALE Portability Index: SPI

SQALE Reusability Index: SRuI

1.2. Quality Index: SQI

For any element of the artefact hierarchy of the source code, the remediation cost relating to all of the Quality Model's characteristics can be estimated by adding up all of the remediation costs linked to all of the Quality Model's requirements.

This derived measure is called: the SQALE Quality Index: SQI.

The SQALE Quality Index is a precise implementation of the concept of "Technical Debt" commonly used to manage agile projects. The SQI represents the Technical Debt associated to the source code.

1.3. Consolidated Indices

In order to give a synthesised representation of the source code's quality, SQALE defines consolidated indices. These indices aggregate the Characteristic Indices in the following way:

- The consolidated index of a given characteristic is equal to the sum of its index with all the indices of the previous characteristics contained in the Quality Model that is being used.

For example, in the case where the Quality Model has kept the eight characteristics of Figure 3.2:

SQALE Consolidated Reliability Index: SCRI = STI + SRI

SQALE Consolidated Changeability Index: SCCI = STI + SRI + SCI

SQALE Consolidated Efficiency Index: SCEI = STI + SRI + SCI + SEI

SQALE Consolidated Security Index: SCSI = STI + SRI + SCI + SEI + SSI

SQALE Consolidated Maintainability Index: SCMI = STI + SRI + SCI + SEI + SSI + SMI

SQALE Consolidated Portability Index: SCPI = STI + SRI + SCI + SEI + SSI + SMI + SPI

SQALE Consolidated Reusability Index: SCRuI = STI + SRI + SCI + SEI + SSI + SMI + SPI+ SRuI

For reasons of coherency, the Consolidated Testability Index is defined as:

SCTI = STI

The use and the interpretation of these consolidated indices are detailed in the appendix. They are used to feed the “SQALE Pyramid” indicator.

1.4. Business Impact Index: SBII

For any element of the artefact hierarchy of the source code, the non-remediation cost can be estimated by adding up all the non-remediation costs linked to all the Quality Model’s requirements.

This derived measure is called: the SQALE Business Impact Index: SBII.

The SQALE Business Impact Index represents the Business perspective of the non-conformities of the source code.

2. Index Densities

2.1. Code Size Measurement

A density index has to be associated with each absolute index defined in the previous paragraph. To do this, the absolute index has to be divided by a measure representing the size of the concerned artefact.

To date, there is no standardised measure enabling the size of the source code in the context that concerns us, in this instance, the produced cost for the work, to be representatively measured.

Failing this, the user or the implementer has to choose, according to the context, a measure that approximates this objective, for example the number of effective code lines, the cyclomatic complexity as per McCabe [Ref. 7], the number of instructions, or the Halstead volume [Ref. 8].

2.2. Index Density

A density index has to be associated with each absolute index defined in the previous paragraph. The code size measure that is adopted for calculating these densities has to be specified.

Index Density Table:

Index Acronym	Density Index Acronym	Density Index Name
STI	STID	SQALE Testability Index Density
SRI	SRID	SQALE Reliability Index Density
SCI	SCID	SQALE Changeability Index Density
SEI	SEID	SQALE Efficiency Index Density
SSI	SSID	SQALE Security Index Density
SMI	SMID	SQALE Maintainability Index Density
SPI	SPID	SQALE Portability Index Density
SRuI	SRuID	SQALE Reusability Index Density

Index Acronym	Density Index Acronym	Density Index Name
SBII	SBIID	SQALE Business Impact Index Density

For example: The SQALE Testability Index Density

STID: SQALE Testability Index Density = STI/KSLOC

Where KSLOC is the size of the artefact expressed in thousands of instruction lines.

Comment:

Although one can in theory calculate index densities on artefacts or groups of artefacts of any size, one should in practice be careful with densities calculated on a single file. A file of very small size can then have a very high density.

6. The SQALE Indicators

The SQALE Method defines three synthesised indicators. Each user or implementer can define others according to his or her information needs. The appendix contains some examples of indicators.

These three SQALE indicators, which relate to the quality characteristics, enable a highly synthesised representation of an application's quality to be given.

1. The SQALE Rating

The rating consists of producing a derived measure on an ordinal scale, for example: Good, Average, Bad.

In the SQALE Method, this one is done on a scale of five values or more (for example with five values: A, B, C, D, and E).

The rating for a given characteristic and a given artefact results from a comparison of the artefact's estimated remediation cost for the characteristic with an estimate of the artefact's development cost.

To do this, one defines in advance a grid such as that in Figure 6.1, conveying the degree of acceptability of the remediation cost in relation to the development cost.

Figure 6.1. SQALE Rating Grid Example

Rating	Up to	Color
A	1%	🟩
B	2%	🟩
C	4%	🟨
D	8%	🟧
E	∞	🟥

Detailed example:

If one has chosen to express the cost in work units (WU) and the size in KSLOC, and if the average cost of 1 KSLOC has been estimated at 100 work units. The grid in Figure 6.1 is used as follows:

An artefact that has a development cost estimated at 500 WU (because its size is 5 KSLOC) and a SQALE Maintainability Index (SMI) of 15 WU, will have a ratio of 15 WU/500 WU = 3%. Its maintainability rating will be "C".

2. The Kiviat

For any artefact of the hierarchy, the various index densities for the various characteristics selected in the applied Quality Model are represented on a Kiviat graph. A SQALE Kiviat graph has the following characteristics:

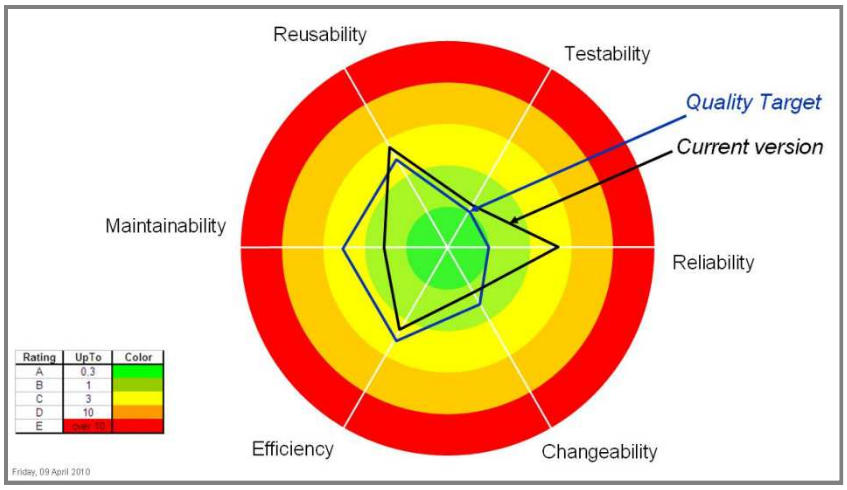
- The "0" values are in the middle.

- There are as many concentric zones as there are values selected for the SQALE Rating (at least five).

- The order of the characteristics is the same as that of the Quality Model.

On the Kiviat, one can add, if one so wishes, the objectives that have been targeted for the analysed software.

Figure 6.2. SQALE Kiviat Example with 6 characteristics



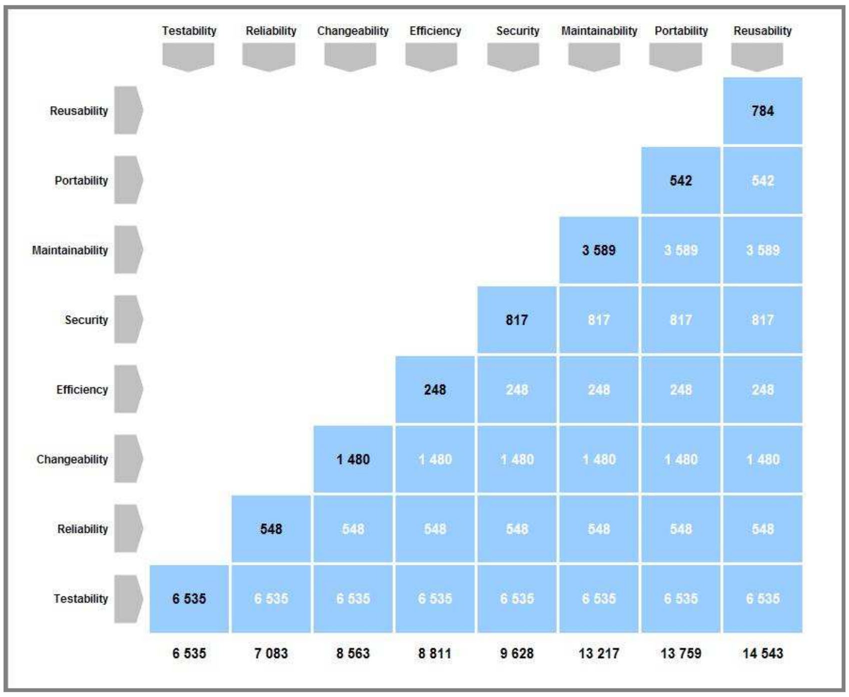
Rating	UpTo	Color
A	0.3	🟩
B	1	🟩
C	3	🟨
D	10	🟧
E	over 10	🟥

3. The SQALE Pyramid

This graph represents, for a given software application, all of the collected indices. In view of the dependence of the model's quality characteristics on the lifecycle, it helps appropriate decisions to be made.

This graph contains on the one hand the absolute indices, and on the other, the consolidated indices, following the chronology of the Quality Model's characteristics. This indicator has the following form:

Figure 6.3. SQALE Pyramid Example



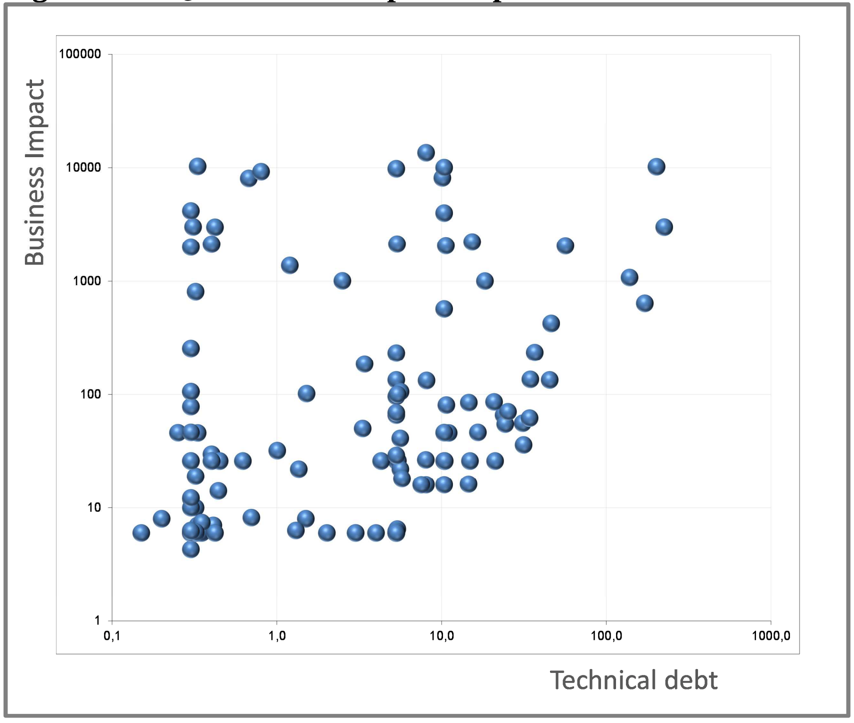
	Testability	Reliability	Changeability	Efficiency	Security	Maintainability	Portability	Reusability
Reusability								784
Portability							542	542
Maintainability						3 589	3 589	3 589
Security					817	817	817	817
Efficiency				248	248	248	248	248
Changeability			1 480	1 480	1 480	1 480	1 480	1 480
Reliability		548	548	548	548	548	548	548
Testability	6 535	6 535	6 535	6 535	6 535	6 535	6 535	6 535
	6 535	7 083	8 563	8 811	9 628	13 217	13 759	14 543

4. The SQALE Debt Map

This graph represents artefacts of the evaluation scope plotted on two dimensions. The first one (X axis) is the Technical Debt axis (SQI), the second one (Y axis) is the Business Impact axis (SBII). This graph uses logarithmic scales.

When the information is available, the size of dots may depend on the business value of the artefacts.

Figure 6.4 SQALE Debt Map Example



5. Steering a project with the appropriate indicators

When monitoring the code quality of a development or maintenance project, one will mainly use the Technical Debt represented by the SQALE quality index. It allows at any time to know the quality status and to monitor the progress. It estimates the cost to spend in order to achieve the pre-set quality goals. This will be normally performed by monitoring and analysing the technical debt of the project (SQI).

Because this index is a cost, it allows comparison with the status and progress regarding the functional requirements.

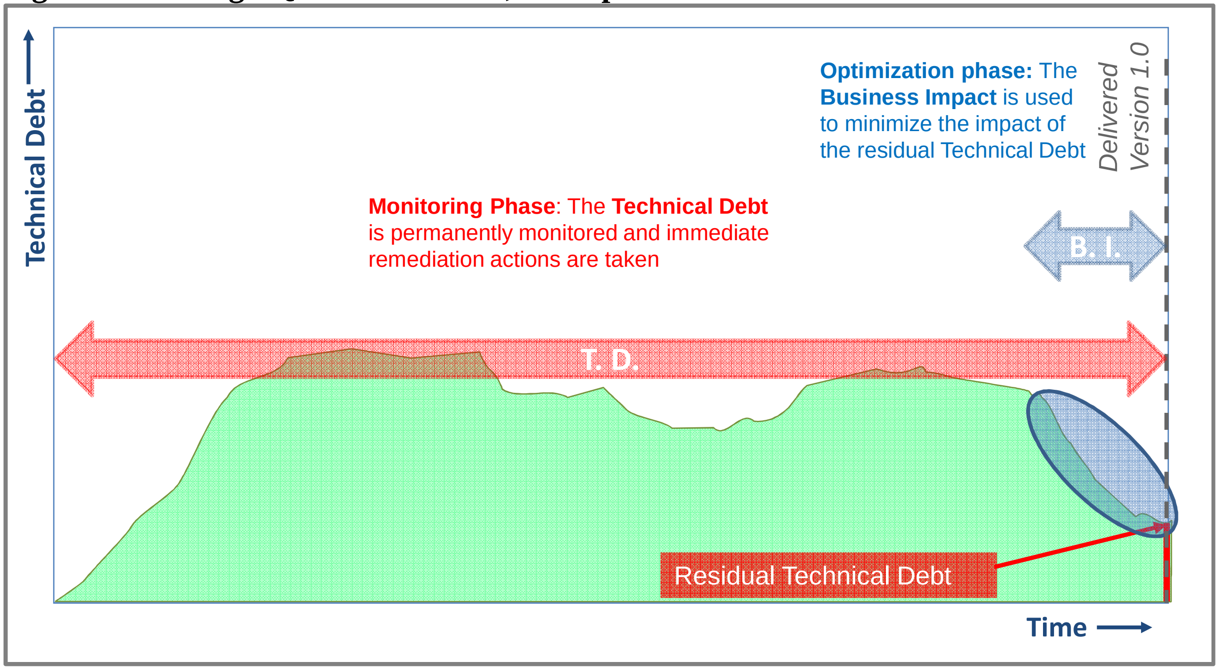
If everything goes well, and the code quality requirements are fulfilled, the code will be delivered without any Technical Debt and associated Business Impact.

When the planned delivery date is imminent and when it clearly appears that the remaining time is incompatible with the fulfilment of all the code requirements, it will be necessary to find a compromise and deliver the code with some remaining Technical and associated Business Impact.

In this case, and at that moment, it is recommended to take into account the Business Impact in order to prioritize remediation actions. During this optimisation phase, the goal will be to decrease as much as possible the Business Impact (SBII) in order to limit the damage resulting from the delivered non-conformities. This will be normally performed by using the SQALE Debt Map indicator. It may be relevant to attribute the highest priority to artefacts located in the upper left quadrant of this map.

Figure 6.5 illustrates an example of SQI and SBII usage within an iterative development project.

Figure 6.5. Using SQALE indicators, Example


Technical Debt
Optimization phase: The Business Impact is used to minimize the impact of the residual Technical Debt
Delivered Version 1.0
Monitoring Phase: The Technical Debt is permanently monitored and immediate remediation actions are taken
B. I.
T. D.
Residual Technical Debt
Time

7. The SQALE Conformity Criteria

In order to be in conformity with the Version 1.0 of the SQALE Method, an implementation must meet the following criteria:

- Support the SQALE Method’s fundamental principles
- Enable a SQALE Quality Model to be defined and set up
- Enable a SQALE Analysis Model to be defined and set up
- Produce the SQALE Characteristic Indices (absolute and consolidated) corresponding to the models set up
- Produce the SQALE Quality Index (SQI) corresponding to the models set up
- Produce the SQALE Business Impact Index (SBII) corresponding to the models set up

The other elements of the method, such as the index densities and the indicators, are optional.

Appendices

1. The representation condition

The representation condition is an important component of the theory of measure.

As a reminder and as explained in [Ref 6 p28]:

"Measurement is defined as the mapping between the empirical and the formal world. A measure is a number or symbol assigned to an entity in order to characterise an attribute."

In order to set up a quality measure system, some rules have to be observed:

"We want the behaviour of the measures, in the number system to be the same as the corresponding elements in the real world, so that by studying the numbers, we learn about the real world. Thus we want the mapping to preserve the relation. (...)

The representation condition asserts that a measure mapping M must map entities into numbers and empirical relations into numerical relations in such a way that the empirical relations preserve and are preserved by the numerical relations." [Ref. 6 p31]

2. Example of Breakdown as Subcharacteristic

The following figure presents an example of a SQALE Quality Model with seven characteristics and the selected subcharacteristics:

Figure 8.1. Example of Levels 1 and 2 of a SQALE Quality Model

Characteristic	Subcharacteristic
Reusability	Extractability
Reusability	Conciseness
Reusability	Stability
Portability	Language related portability
Portability	Time zone related portability
Portability	Hardware related portability
Portability	External application related portability
Portability	Compiler related portability
Portability	OS related portability
Maintainability	Understandability
Maintainability	Readability
Security	OS related security
Security	User related security
Security	Statement related security
Efficiency	ROM related efficiency
Efficiency	RAM related efficiency
Efficiency	CPU related efficiency
Changeability	Architecture related changeability
Changeability	Logic related changeability
Changeability	Data related changeability
Reliability	Fault tolerance
Reliability	Architecture related reliability
Reliability	Resource related reliability
Reliability	Synchronization related reliability
Reliability	Statement related reliability
Reliability	Logic related reliability
Reliability	Data related reliability
Testability	Integration Testing testability
Testability	Unit Testing testability

3. Quality Model Examples

The following figure presents an example of a SQALE Quality Model for the C++ language.

Figure 8.2. SQALE Quality Model Example for the C++ Language

Characteristic	SubCharacteristic	Generic Requirement Description
Maintainability	Understandability	No use of double pointer
Maintainability	Understandability	No unstructured statements (goto, break outside a switch...)
Maintainability	Understandability	No use of "continue" statement within a loop
Maintainability	Understandability	No use of ternary operators
Maintainability	Understandability	File comment ratio (COMR) > 25%
Maintainability	Readability	Capitalization rules for identifying code elements are followed
Maintainability	Readability	The code follow consistent block formatting rules
Maintainability	Readability	The code follow consistant whitespace rules
Maintainability	Readability	The code follow consistant indentation rules
Maintainability	Readability	File size (LOC) <1000
Maintainability	Readability	All declarations are done in a consistent order
Maintainability	Readability	No commented-out code
Efficiency	ROM related efficiency	All statements are useful
Efficiency	RAM related efficiency	Class depth of inherítence (DIT) <8
Efficiency	RAM related efficiency	No unused variable, parameter or constant in code
Changeability	Architecture related changeability	Class weighted complexity (WMC) <100
Changeability	Architecture related changeability	Coupling between objects (CBO) <7
Changeability	Logic related changeability	All if, for, while structures have a clear scope delimitation
Changeability	Data related changeability	No explicit constants directly used in the code (except 0,1, True and False)
Reliability	Architecture related reliability	No multiple inheritance of implementation classes
Reliability	Resource related reliability	No use of freed or unallocated memory
Reliability	Statement related reliability	No ambiguous statement execution order
Reliability	Logic related reliability	No assignement within a condition
Reliability	Logic related reliability	Invariant iteration index
Reliability	Data related reliability	No use of unitialized variables
Reliability	Data related reliability	All types are safely converted
Reliability	Data related reliability	No module with a variable number of parameters
Reliability	Data related reliability	All types are explicitly declared (no Void *)
Testability	Unit Testing testability	All modules are reachable
Testability	Unit Testing testability	No duplicate part over 100 token
Testability	Unit Testing testability	Number of ind. test paths within a module (v(G)) <15
Testability	Unit Testing testability	Number of parameters in a module call (NOP) <7

The following figure presents an example of a quality model for the Java language.

Figure 8.3. SQALE Quality Model Example for the Java Language

Characteristic	SubCharacteristic	Generic Requirement Description
Maintainability	Understandability	No unstructured statements (goto, break outside a switch...)
Maintainability	Understandability	No use of "continue" statement within a loop
Maintainability	Understandability	File comment ratio (COMR) > 35%
Maintainability	Readability	Variable name start with a lower case letter
Maintainability	Readability	The closing brace '}' is on a standalone line
Maintainability	Readability	The code follow consistant indentation rules
Maintainability	Readability	File size (LOC) <1000
Maintainability	Readability	No commented-out code
Efficiency	RAM related efficiency	Class depth of inherítence (DIT) <8
Efficiency	RAM related efficiency	No unused variable, parameter or constant in code
Changeability	Architecture related changeability	Class weighted complexity (WMC) <100
Changeability	Architecture related changeability	Class specification does not contains public data
Changeability	Logic related changeability	If, else, for, while structures are bound by scope
Changeability	Data related changeability	No explicit constants directly used in the code (except 0,1, True and False)
Reliability	Fault Tolerance	Switch' statement have a 'default' condition
Reliability	Logic related reliability	No assignement ' =' within 'if' statement
Reliability	Logic related reliability	No assignement ' =' within 'while' statement
Reliability	Logic related reliability	Invariant iteration index
Reliability	Data related reliability	No use of unitialized variables
Testability	Integration level testability	No "Swiss Army Knife" class antipattern
Testability	Integration level testability	Coupling between objects (CBO) <7
Testability	Unit Testing testability	No duplicate part over 100 token
Testability	Unit Testing testability	Number of ind. test paths within a module (v(G)) <11
Testability	Unit Testing testability	Number of parameters in a module call (NOP) <6

4. Remediation Function Examples

In order to associate a remediation function with each of the model's requirements, one can do it individually by estimating the average remediation cost of a non-conformity for each requirement.

One can also apply a logic consisting of analysing the "remediation cycle" which will be engaged at the time of the remediation of the nonconformities.

In the following example, in a given organisation and in the light of its context, the requirements have been classified in five types as defined in the following table. All the requirements of the same type have been associated with the same remediation factor.

Figure 8.4. Example of Remediation Factors associated with various types of non-conformity

NC Type Name	Description	Sample	Remediation Factor
Type1	Corrigible with an automated tool, no risk	Change in the indentation	0.01
Type2	Manual remediation, but no impact on compilation	Add some comments	0.1
Type3	Local impact, need only unit testing	Replace an instruction by another	1
Type4	Medium impact, need integration testing	Cut a big function in two	5
Type5	Large impact, need a complete validation	Change within the architecture	20

Figure 8.5 shows an example of a remediation function that one can apply for calculating the remediation cost when the required comment ratio has not been reached. In this example, the threshold is 30%.

For a given artefact, Value "a" corresponds to the cost necessary for adding comments and for bringing into conformity an artefact that has no comment (and for carrying out the possible corresponding verification cycle). Value "a" depends on the size of the file and consequently on the number of comments line corresponding to a ratio of 30%.

Value "b" corresponds to the cost consisting of adding only one comment line (and of carrying out the possible corresponding verification cycle).

Figure 8.5. Example of a Remediation Function for the Comment Ratio

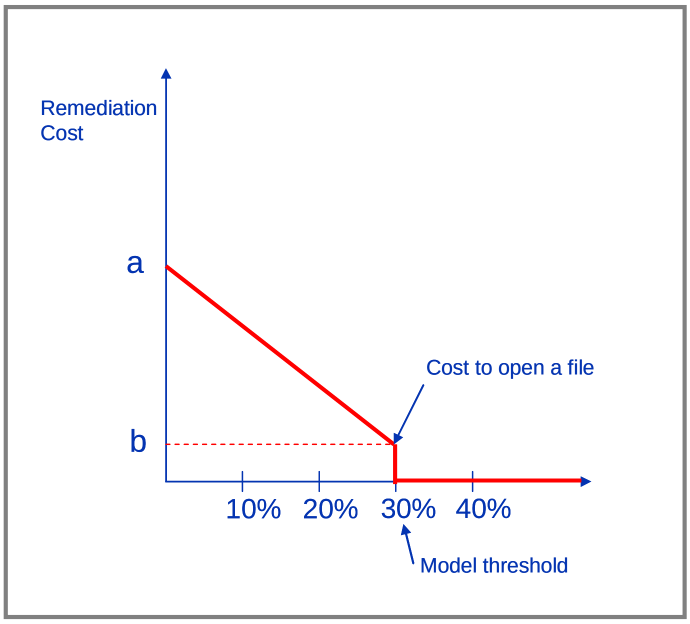
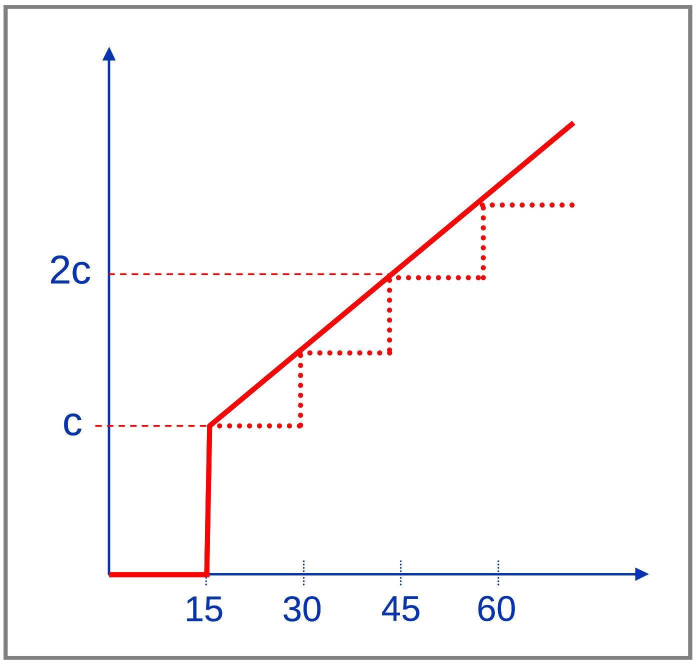


Figure 8.6 shows an example of a remediation function that one can apply to calculate the remediation cost when one cuts an artefact - into two or more in order to reduce its coupling or its cyclomatic complexity. In this

case, the threshold is 15. Value "C" is the cost corresponding to the cutting into 2. When the artefact has a complexity cyclomatic higher than three times the threshold, it has to be cut into 4, that is to say a cost estimated twice "C", which establishes the slope of the straight line.

Figure 8.6. Example of a Remediation Function for Cutting an Artefact



5. Non-Remediation Function Examples

In order to associate a remediation function with each of the model's requirements, one can do it individually by estimating the average non-remediation cost of a non-conformity for each requirement.

One can also apply a logic consisting of associating a criticality to each requirement.

In the following example, in a given organisation and in the light of its context, the requirements have been classified in five types as defined in the following table. All the requirements of the same type have been associated with the same non-remediation factor. This represents the priority of the different types of non-conformities as perceived by the Product Owner.

Figure 8.7. Example of Non-Remediation Factors associated with various types of non-conformity

NC Type	Description	Sample	Non-Remediation Factor
Blocking	Will or may result in a bug	Division by zero	5 000
High	Wil have a high/direct impact on the maintainance cost	Copy and paste	250
Medium	Will have a medium/potential impact on the maintainance cost	Complex logic	50
Low	Wil have a low impact on the maintainance cost	Naming convention	15
Report	Very low impact, it is just a remediation cost report	Presentation issue	2

6. Examples of additional Indicators to the SQALE Indicators

The following Figures give four examples of indicators built upon the SQALE indices.

Figure 8.8. Comparative Histogram of SQALE Maintainability Index Densities

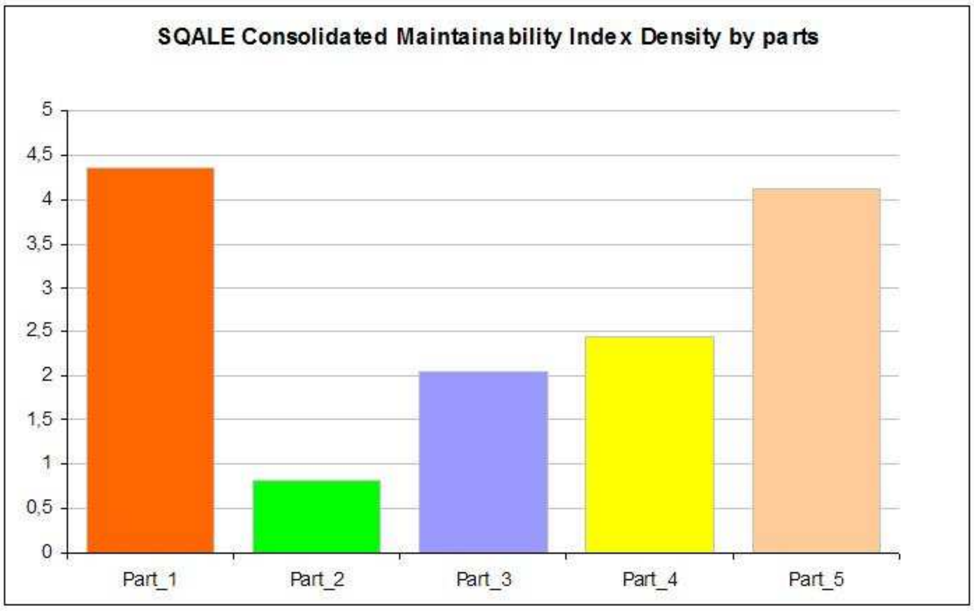


Figure 8.9. Historical Trend Graph of the Technical Debt (SQALE Quality Index) according to an application's various parts

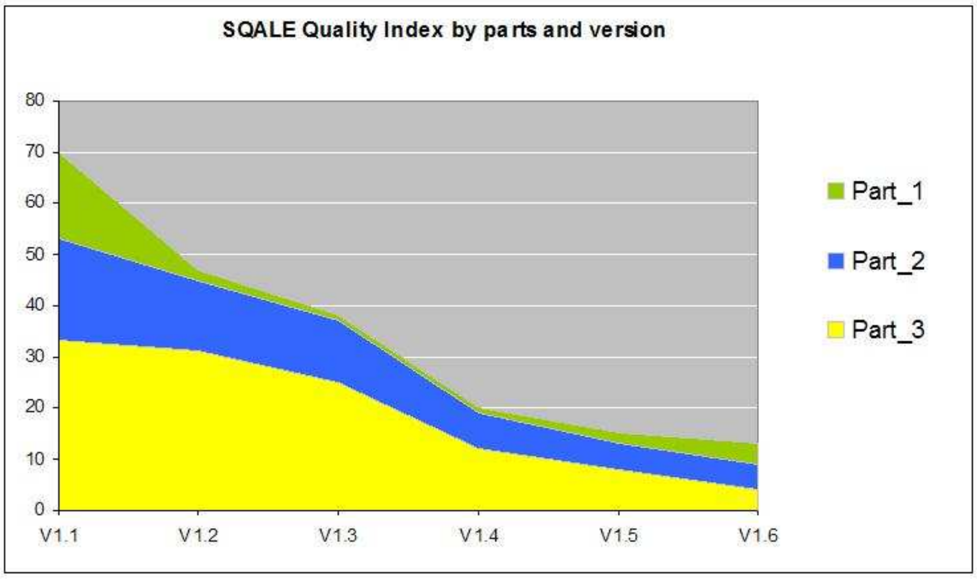


Figure 8.10. Historical Trend Graph of the Technical Debt (SQALE Quality Index) according to the severity of the violations

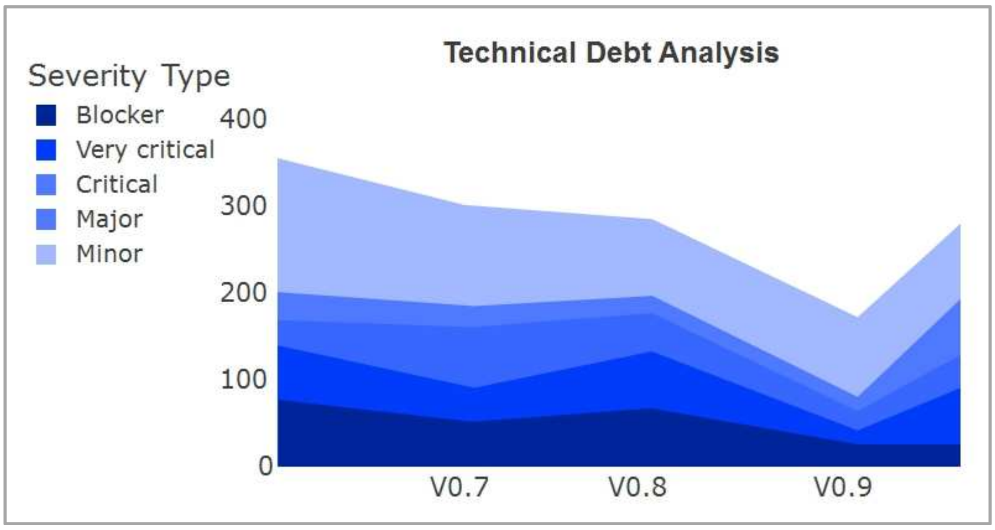
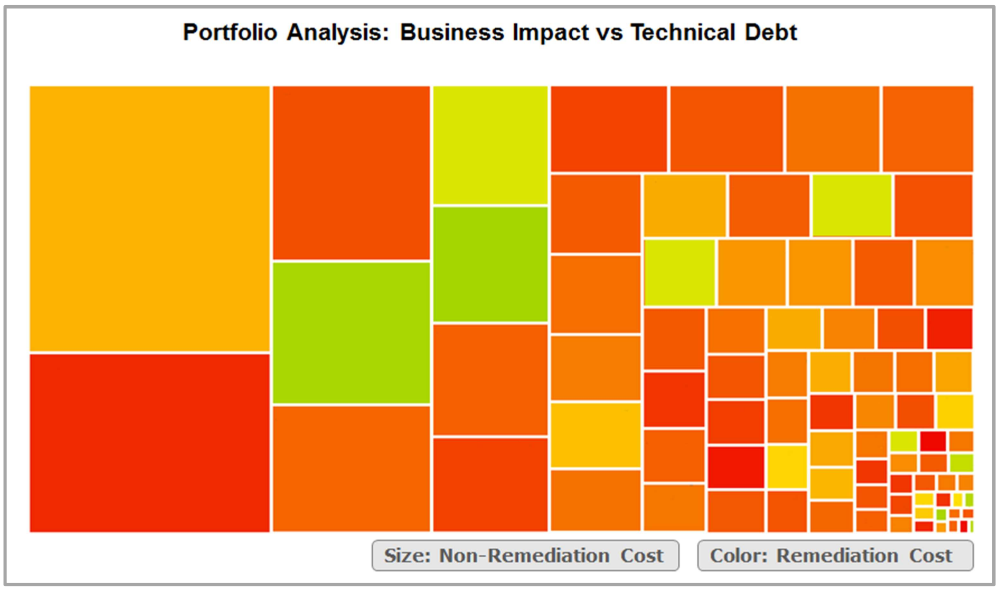


Figure 8.11. Application Portfolio Management with a Map representation



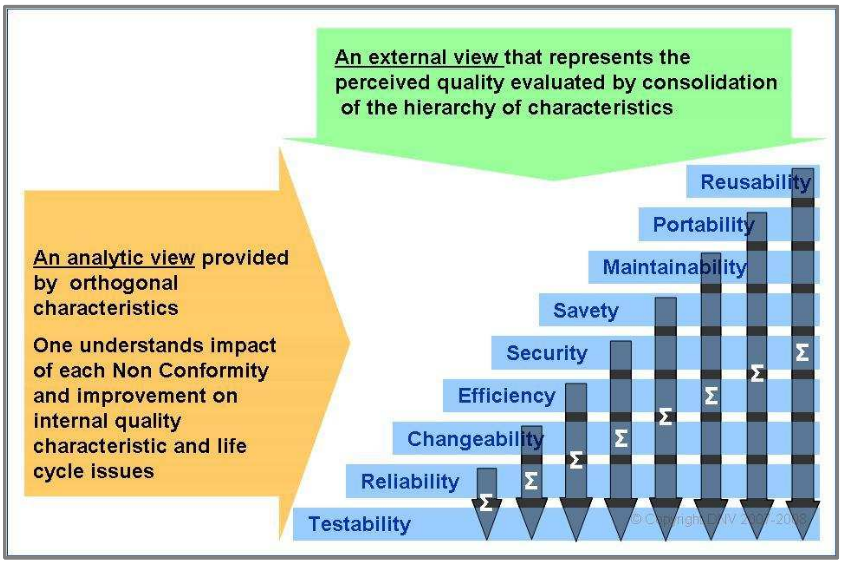
7. The SQALE Quality Model's Two Points of View

The SQALE Quality Model has the characteristic of supporting two points of view. On the one hand, the developer's point of view through the subcharacteristic and characteristic indices. Those indicating to him the presence of non-conformities impacting the precise activities of the lifecycle. On the other, the user's or the "owner's" point of view with the consolidated characteristic indices that represent the totality of the non-conformities to be remedied in order to reach the phase of the lifecycle concerned under the best conditions.

For example, the absolute maintainability index represents the remediation cost for the requirements directly linked to maintainability. But in order for a software application to be able to be easily maintained in its operating phase, it has to be testable, reliable, upgradable, performing and maintainable. It is therefore the consolidated maintainability index that best represents the software's status in relation to that objective.

This duality of the points of view is synthesised in Figure 8.

8.12. The SQALE Quality Model's Two Points of View


An external view that represents the perceived quality evaluated by consolidation of the hierarchy of characteristics
An analytic view provided by orthogonal characteristics
One understands impact of each Non Conformity and improvement on internal quality characteristic and life cycle issues
Reusability
Portability
Maintainability
Savety
Security
Efficiency
Changeability
Reliability
Testability
Σ
© Copyright DNV 2007-2008

8. Use Scenario of the SQALE Method: Example 1

This paragraph gives a detailed example of a SQALE Method use scenario. The chosen example describes how an organisation can use the SQALE Method for properly controlling the development of a project that has been subcontracted to a service company.

The main stages are the following:

1 - The project manager identifies the quality objectives for the application that he is going to have developed. These objectives are for example very great reliability and a good quality for the remainder of the characteristics of the source code.

If the organisation does not already have a SQALE Quality Model, it develops one that lists all of the requirements relating to the code's quality.

2 - When the contract is drawn up, the requirements relating to the code's quality are specified by contract. That results in the communication of the following elements:

- The SQALE Quality Model specifies what will be used as reference (i.e. the precise requirements for the language used)
- The remediation functions used to calculate the remediation indices
- The target thresholds for various index densities in agreement with the targeted objectives

For example, if the size is expressed in KSLOC and if reliability is a strong requirement:

- STID < 0.1
- SRID < 0.1
- Other characteristic SQALE indices densities < 3

3 - The subcontractor, during the course of the project, reports the value of the various indicators at the time of the steering committee meetings in order to assure the client that the code's quality objectives will be met.

4 – At the time of the delivery, the subcontractor provides the analysis results of establishing that the contractual thresholds have been met.

5 - If necessary, at the time of the acceptance test, the organisation checks with its own tools that the requirements have indeed been met.

9. Use Scenario of the SQALE Method: Example 2

This paragraph gives another detailed example of a SQALE Method use scenario. The chosen example describes how an organisation can use the SQALE Method for properly managing the quality of the code developed by an agile project. This is performed by continuously monitoring the Technical Debt of the project.

The main stages are the following:

1 - The project's team identifies the quality objectives and the associated requirements for the code that will be developed. Any violation of these requirements will create a Technical Debt.

If not provided by the organisation, the team develops the SQALE Quality Model and the SQALE Analysis model that will be used to estimate the Technical Debt accumulated by the project.

2 – The continuous integration system is configured to calculate the SQI (that estimates the Technical Debt) and

the associated SQALE indicators at each build.

3 - At each build the team monitors and analyses the Technical Debt of the project (SQI). This index is used to monitor the project in addition to other indicators like backlog and velocity. The SQALE indicators are used to analyse in detail the Technical Debt and decide the priorities and the best moment to start some refactoring activities. Normally, refactoring should start by addressing the issues related to the lowest layers of the SQALE Quality Model, which means Testability and Reliability.

10. Comments

Readers are invited to provide their comments or any problems noted in relation to this document, be they on technical or presentational aspects. This can be done on the website: http://www.sqale.org

11. List of Change Requests

A number of amendments have not been taken into account in the present version of this document. Here are the main ones:

CR1	Bring all the terms used in this document in line with one established terminology (OMG, IEEE, ISO ...).
CR2	Insert a glossary
CR4	Insert an ontology or a meta model of the concepts used by the method

12. Orientations

A number of proposals have been received for the possible extension of this document. No decision has been made concerning their adoption. Here are the main ones:

- Add information helping to implement the method
- Establish standard quality models per language and per type of business
- Establish standard analysis models per language and per type of business
- Propose target density thresholds for those standards according to the applications’ criticality
- Define standard formats, in the form of XML DTD, in order to exchange the models between organisations and between tools
- Define standard formats, in the form of XML DTD, in order to exchange the automatic analysis results between organisations and between tools.