SQuORE and the quality of software development

Boris Baldassari,
Université de Lille 3, Lille, France
boris.baldassari@gmail.com

1 Introduction

Software quality is a common subject among practitioners and users of software products. But although it generates a great deal of discussion and interest, quality assurance is often not well understood, nor considered worth investing in by managers and developers. As of today, many people still consider it a hazardous and expensive part of the art of development.

Bad quality has a price however, or rather, many cost factors. Firstly, there are the costs that can easily be estimated such as when a single bug shuts down a whole server or system, interrupts transactions, make a flying module explode. Secondly there are the costs hidden in the maintenance of the product which are more difficult to estimate. The former are well-known and quite impressive: the explosion of the Ariane 5 shuttle (500M$), the AT&T network crash (60M$ traced back to a missing break), or the destruction of the Mars Climate Observer space module. The latter costs are less visible, but have far greater impact: each and every software project has its own debt, with the associated recurring interest.

In this article, we propose a few ideas to help the practitioner define the objectives and requirements of quality, how to setup a quality evaluation process, and how to implement it using SQuORE[1], a multi-purpose tool used for the evaluation of software projects.

2 A few words about quality

Defining quality

To state it up front, there is no single definition of quality that fits every domain and every person. When it comes to software development, the most often cited definitions target customer satisfaction[a], fitness for purpose[b], and conformance to requirements[c]. In real life situations the concept depends on the chosen perspective – e.g. stakeholder, developer, or end-user – and on the domain – the requirements are not the same for a critical embedded system as they are for a desktop text editor.

From this we propose to define, or redefine, what quality is in every specific context, following the perspectives on quality enumerated by Garvin[d], to adapt the measurement and analysis process to the specific requirements and needs of the situation.

1 http://www.squoring.com/en/

Measuring quality

Another great pitfall is the pertinence of measures of software characteristics: how do we know that we measure what we really intend to, what is the semantic and accuracy shift introduced, and what impact does the method have on the measure itself? Several authors worked on this concern (Fenton[e], Kaner[f], Pfleeger[g]), demonstrating how devastating or inaccurate a metric can be if badly measured or understood, and proposing hints to avoid the most common issues.

One good example of this is how to measure productivity: some use the number of lines of code written, but this has a bad side effect: files often end up being artificially long, for mediocre quality (since people know they will be judged for the length, not the content). For the same reasons, maintainability or reliability do not depend directly on the size or complexity of the code base, although they are highly correlated. The former typically depends on the population of developers (experience, background) and the conventions used in the project. The latter can hardly be estimated through the number of bugs entered in the issue tracking system, because there is no proven relationship between the number of bugs entered and the real number of bugs inside the product: if a product provides an easy way to report bugs, there will be more bugs entered in the issue tracking system, which will be identified and corrected, thus patently improving reliability. On the other hand a product which has not been thoroughly tested will have a small number of registered bugs, but will undoubtedly be less mature and reliable.

The main approach used today is multi-dimensional analysis: attributes of quality are estimated through a large set of software measures, which are then weighted and aggregated to form a comprehensive result. These measures have to be understood and accepted by the various actors of the development process, and using methods like Basili's Goal-Question-Metric approach helps to preserve the semantic and consistency of the measurement process.

Quality Models

When it comes to measuring several characteristics of a product or process, one has to analyse and decompose the main definition of quality and refine it in sub-characteristics called attributes of quality. These represent various aspects and concerns of the system under analysis, like support reactivity, predictability of outputs, readability or stability of code. Figure 1 illustrates such a quality model derived from the ISO 9126 standard, defining its main characteristics and sub-characteristics down to the computation of base measures.

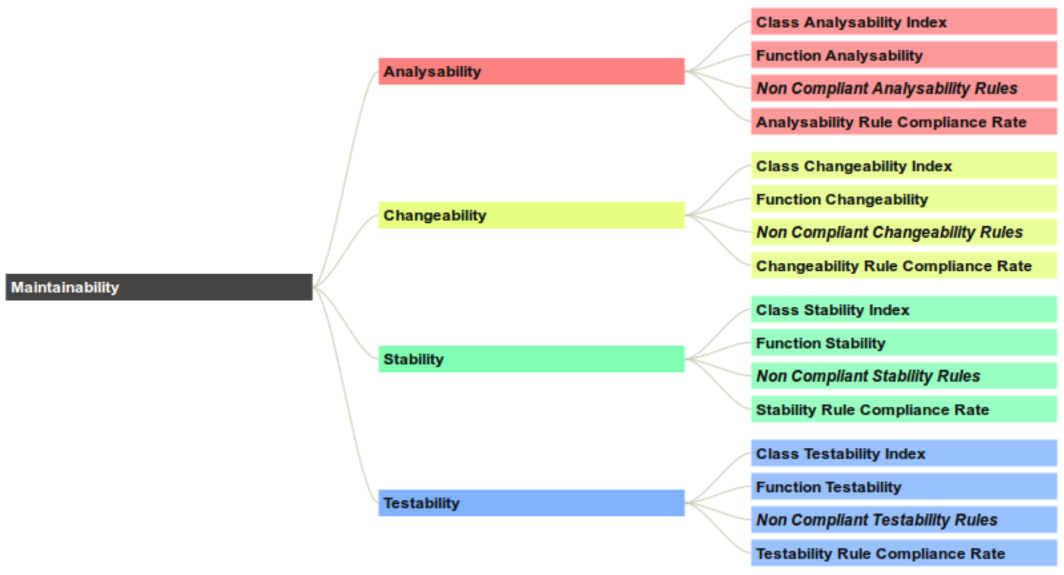

Maintainability
Analysability
Class Analysability Index
Function Analysability
Non Compliant Analysability Rules
Analysability Rule Compliance Rate
Changeability
Class Changeability Index
Function Changeability
Non Compliant Changeability Rules
Changeability Rule Compliance Rate
Stability
Class Stability Index
Function Stability
Non Compliant Stability Rules
Stability Rule Compliance Rate
Testability
Class Testability Index
Function Testability
Non Compliant Testability Rules
Testability Rule Compliance Rate

Figure 1: A product-oriented quality model (SQuORE)

Within a same company, different services may, and will, have different requirements of quality, and will probably measure it via different means: e.g. number of shipped and returned products, customer satisfaction, and sales results. Thus different quality models, serving different purposes, may co-exist in the organisation. They may well have different quality attributes, weights, measures and aggregation methods.

As an answer to these various requirements and perspectives on quality, a comprehensive amount of standards have been published to help practitioners express and structure their views. Examples include ISO 15504 (Spice) and CMMi for process evaluation, and SQUALE, ISO 9126 and ISO 250xx (SQuaRE) for product quality. These standards bring a safe foundation to build upon, with clear definitions of terms, a normalised structure for the evaluation, and even propose in some cases a method and recommendations to help people setup the quality evaluation process.

Furthermore a great ecosystem contributes to the maturity of the domain, with abundant literature and documentation (usage, criticism, discussions) around these standards, and a comprehensive offering of services (audit or counsel).

3 SQuORE

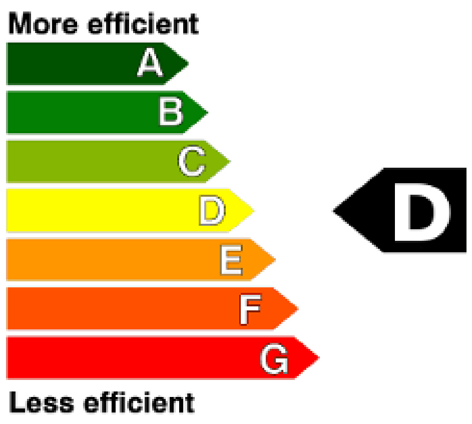


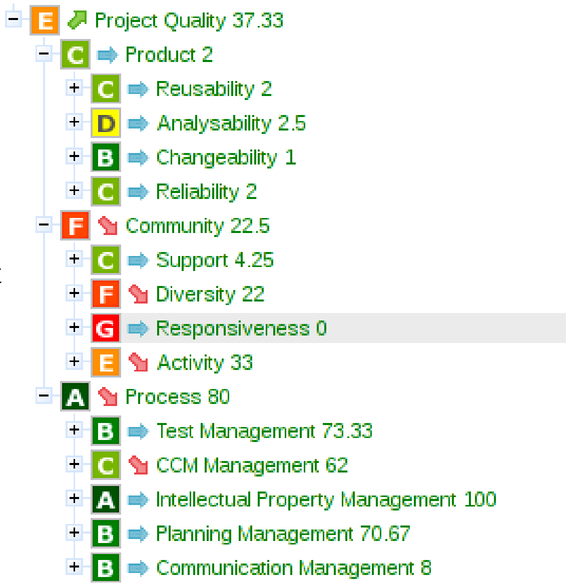
Illustration 2: The SQuORE notation

SQuORE aims to provide a ready-to-use quality assessment framework that can be entirely configured for the distinctive features of a given process. Several quality models are provided out-of-the-box, following the structure of established standards like ISO 9126 or the technical debt. Starting from such safe roots, one can then tune the quality model, add or remove measurement inputs to fit the data and requirements for the company.

SQuORE Quality Models

SQuORE uses modular and reusable XML configuration files to describe the structure of the quality model and the aggregation method used to sum measures up to the quality attributes. Thus it is easy to add a new quality attribute (say customer support reactivity) and its computation method (based, say, on the mean time to first answer and a satisfaction survey).

A comprehensive set of operators is available to compute measures at different levels, from functions up to files, folders, applications, department or service. This allows administrators to customise the quality model to fit the unique cultural environment of the company. Figure 4 presents an example of project-oriented quality model, composed of three custom axes: product, community, and process.



- E Project Quality 37.33
  - C Product 2
    - C Reusability 2
    - D Analysability 2.5
    - B Changeability 1
    - C Reliability 2
  - F Community 22.5
    - C Support 4.25
    - F Diversity 22
    - G Responsiveness 0
    - E Activity 33
  - A Process 80
    - B Test Management 73.33
    - C CCM Management 62
    - A Intellectual Property Management 100
    - B Planning Management 70.67
    - B Communication Management 8

Illustration 3: An example of project quality breakdown.

Types of data

SQuORE has its own multi-language analyser for source code analysis, and generates the usual code metrics: size, complexity, Halstead measures, and object-oriented metrics like the depth of inheritance tree. But SQuORE's main strength is its ability to connect to the outside world. It can execute a rule-checking tool (like CheckStyle, PMD, FindBugs, Coverity, or Polyspace) or a custom Perl script, and parse its output to include its results in the quality report. All standard formats are supported: CSV, JSON, XML, and even data bases. This feature allows to include unusual types of data from various areas of the development process and environment: tests, requirements, or the number of shipped products.

Action Items

Action items (AI) are defined by a combined set of thresholds on various metrics. They allow to identify complex patterns, relying on several conditions, and generate warnings accordingly. As an example, consider a long file (high SLOC) with a low comment rate and several practices violated inside. Each of these characteristics are not really harmful considered individually, but have a devastating effect when combined. For each action item SQuORE provides a clear explanation of the problem with the associated threshold values, and proposes pragmatic improvement steps.

Visualisation

At the end of the process, end-users have to really understand the results of the analysis and the way it is conducted, to take corrective actions that really matter. Pictures are often worth a thousand words, and there is a good graphic for every concern. Time series show the evolution of a pathology and help understand how things have evolved to produce the actual situation (figure 4, left). Scatterplots are helpful to identify abnormal and extreme values. Control graphs make visible the complexity of a function or a file (figure 4, right).

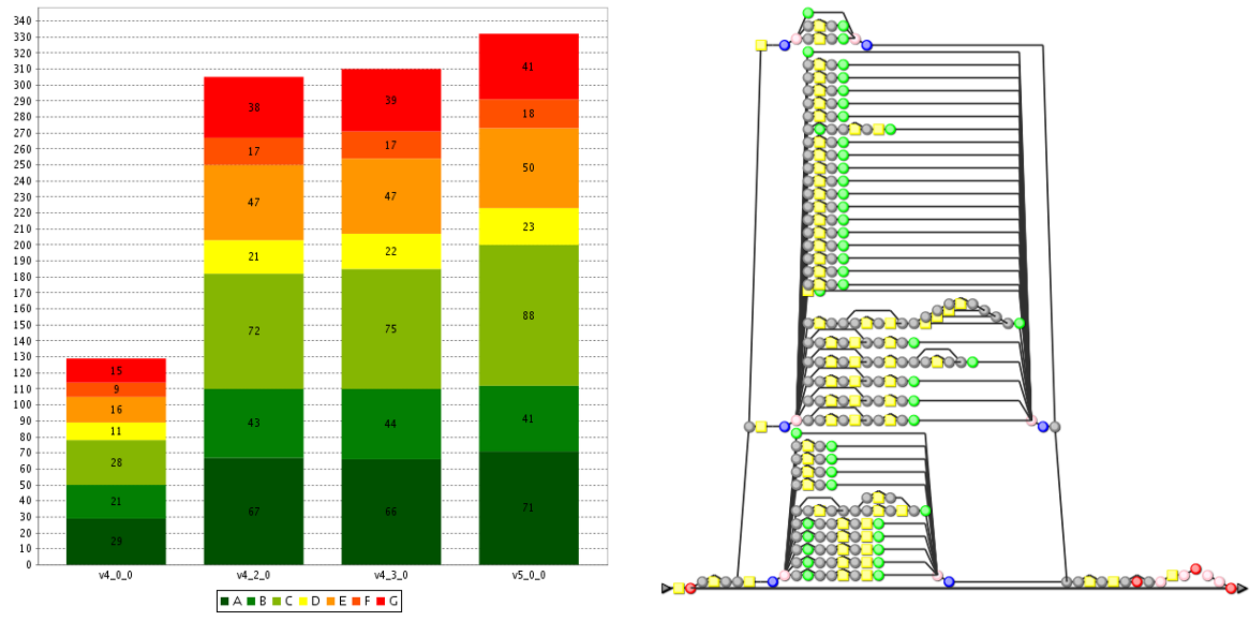


Illustration 4: Examples of visualisation (SQuORE)

Using nice and useful visualisation serves two main purposes. Firstly for the end users: they have to understand what the evaluation means for them and what good and bad practices it involves, and also how to improve it all. Secondly for managers to monitor effectively the project, forecast its behaviour and anticipate upcoming difficulties.

This last link of the chain has a fundamental importance, because the whole process is pointless if the users can not understand and use the results, and if there is no pragmatic and visible improvement in the project's quality. This also increases the confidence in the measurement and analysis process, demonstrates the benefits of the quality assurance program, and justifies further investment.

4 Conclusion

In this article we presented some fundamental concepts of a software quality assessment and improvement program, from measurement to the structure of the quality model. Most common issues have been presented, along with means to solve them. We also presented the basic principles of SQuORE, and how it can be successfully used in the context of a specific quality assurance program.

As a conclusion, one should not look for a universally accepted definition of software quality, neither for a panacea to software development problems. Each and every situation has its own quality requirements, measures, and cultural and domain biases. Nevertheless, by carefully reviewing the quality requirements and constraints of projects, by setting up meaningful measures, and by involving the users of the analysis process in its definition so they really understand what is measured and how, one will not only have a sound measurement process but will also be able to pro-actively help improve it according to the selected axes.

a Deming, W. E. (1988). Out of the crisis: quality, productivity and competitive position. Cambridge University Press.

b Juran, J. M., Godfrey, A. B., Hoogstoel, R. E., & Schilling, E. G. (1999). Juran’s Quality Handbook (Vol. 2). McGraw Hill New York.

c Crosby, P. B. (1979). Quality is free: The art of making quality certain (Vol. 94). McGraw-Hill New York.

d Kitchenham, B., & Pfleeger, S. (1996). Software quality: the elusive target. IEEE Software, 13(1), 12–21.

e Fenton, N., & Pfleeger, S. (1991). Software metrics.

f Kaner, C., & Bond, W. P. (2004). Software engineering metrics: What do they measure and how do we know? Methodology, 8(6), 1–12.

g Fenton, N., & Pfleeger, S. (1998). Software metrics: a rigorous and practical approach. Brooks/Cole Pub Co.