SQuORE: a new approach to software project assessment.

Boris Baldassari
SQuORING Technologies
76, Alles Jean Jaurs,
31000 Toulouse - France
www.squoring.com
boris.baldassari@squoring.com

October 25, 2012

Abstract

Quality has a price. But non-quality is even more expensive. Knowing the cost and consequences of software assets, being able to understand and control the development process of a service, or quickly evaluating the quality of external developments are of primary importance for every company relying on software. Standards and tools have tried with varying degrees of success to address these concerns, but there are many difficulties to be overcome: the diversity of software projects, the measurement process – from goals and metrics selection to data presentation, or the user's understanding of the reports. These are situations where the SQuORE business intelligence tool introduces a novel decision-based approach to software projects quality assessment by providing a more reliable, more intuitive, and more context-aware view on quality. This in turn allows all actors of the project to share a common vision of the project progress and performance, which then allows efficient enhancing of the product and process. This position paper presents how SQuORE solves the quality dilemma, and showcases two real-life examples of industrial projects: a unit testing improvement program, and a fully-featured software project management model.

Key words: software quality, key performance indicators, trend analysis, measurement, quality models, process evaluation, business intelligence, project management.

Contents

1 Introduction

Despite an increasing interest in software quality, many still think about quality achievement as an expensive and unproductive process. On the other hand, as Shaw [23] pointed out a few years earlier, the software engineering discipline is currently in the process of maturing: in the past decade, new methods and new processes have grown, standards have been published, and the many years of experience gathered in the field brought much feedback and a new maturity to the discipline. The time has come for a new era in business intelligence [5] for software projects.

In the second section of this paper, we lay down the ground foundations of our approach to software measurement and present the state of practice, along with some terms and concepts about software development quality and management. In section three, we discuss the SQuORE approach to software projects assessment, with its features and benefits. Section four further expands the scope of quality assessment by showing two real-life implementations that demonstrate the use of SQuORE, with unit testing prioritisation and project- and schedule dashboards and models.

2 State of practice

2.1 The cost of failures

There are many examples of software project failures, and their associated costs – either in human or financial losses. All of them originate from non-quality or lack of control on development. Some well-known example bugs in the past decades include:

- Therac-25: six deaths before beeing fixed, took two years to diagnose and fix [15].
- Mars Climate Orbiter: race conditions on bus priority, system rebooted continuously and robot eventually crashed[19].
- Patriot missile target shifted 6 meters every hour due to float precision bug. 28 soldiers killed in Dhahran [25].
- Ariane 5 infamous buffer overrun crash due to abusive reuse of Ariane 4 software[16]: 500 millions $ pure loss.
- AT & T failure of 1990: software upgrade of switch network led to a 9 hours crash, traced back to a missing break[20]. 60 Million $ lost revenue.

According to a study conducted by the U.S. Department of Commerce's National Institute of Standards and Technology (NIST), the bugs and glitches cost the U.S. economy about 59.5 billion dollars a year[21]. The Standish Group CHAOS 2004 [24] report shows failure rates of almost 70%.

2.2 Standards for software products and processes

Many quality-oriented standards and norms have been published in the last decades, some focusing on product quality (from Boehm [1] and McCall [18], further simplified and enhanced by ISO 9126 [9]), while other rather consider the process quality (e.g. ISO/IEC 15504 [6] and CMMi [2]). More recently, two quality assessment standards have been developed: SQALE [14, 13], a generic method independent of the language and source code analysis tools, mainly relying on technical and design debts, and ISO SQuARE [7], the successor of ISO 9126, which is still being developed. Furthermore, some domains have their own de-facto standards: HIS and ISO 26262 [8] for the automotive industry, or DO-178 [22] for the aeronautics and critical embedded systems.

But some objections have been opposed to established standards, because:

- they may be seen as mere gut-feeling and opinions from experts, as pointed out by Jung et al [10] for the ISO 9126, and
- they dont fit well every situation and view on quality, as they are rather scope-fixed [11].

Another point, also related to the complexity and diversity of software projects, is that published standards dont provide any pragmatic measures or tool references, which leads to misunderstanding and misconceptions of what is measured as an example, consider the thousands of different ideas and concepts behind the Mean Time To Failure metric [12].

2.3 Metrics for software quality measurement

There is a huge amount of software-oriented metrics available in the literature. Examples of weidly-used metrics include McCabe's cyclomatic complexity for control flow[17], Halstead's complexity for data flow[4], size or coupling measures. Each measure is supposed to characterise some attributes of software quality, and they have to be put together to give the complete picture.

Another mean to assess software quality is the number of *non-conformities* to a given reference. As an example, if naming or coding conventions or programming patterns have been decided, then any violations of these rules is supposed to decrease the quality, because it threatens some of the characteristics of quality, like analysability (for conventions), or reliability (for coding patterns).

The concept of *technical debt*, coined by Ward Cunningham in 1992 [3] and gaining more and more interest nowadays, can be considered as the distance to the desired state of quality. In that sense, it is largely driven by the number and importance of non-conformities.

The trend of software measurement globally tends to multi-dimensional analysis [12]: quality or progress of a software project or product is a composite of many different measures, reflecting its different characteristics or attributes. The next step is the way information can be aggregated and consolidated.

3 Principles of SQuORE

The purpose of SQuORE is to retrieve information from several sources, compute a consolidation of the data, and show an optimised report on software or project state. The rating of the application is displayed on a common, eye-catching 7-steps scale which allows immediate understanding of the results, as shown in Figure 1.

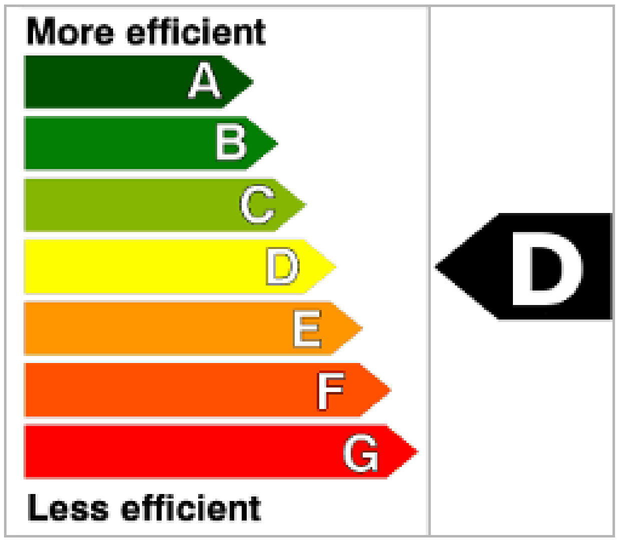


Figure 1: The 7-levels SQuORE rating

3.1 Architecture

SQuORE analysis process can be broken down in three separate steps: *data providers* that take care of gathering inputs from different sources, the *engine*, which computes the consolidation and rating from the base measures gathered by data providers, and the *dashboard* to present information in a smart and efficient way.

3.2 Data Providers

As stated in our introduction, there are nowadays many tools available, each one having a specific domain of expertise and an interesting, but partial, view on quality. SQuORE brings in the glue and consistency between them all, by importing this information any type of input is accepted, from xml or csv to Microsoft binary files and processing it globally.

The SQuORE analyser runs first. It is fast, does not need third-party dependencies, and constitutes a tree of artefacts corresponding to the items measured: source code, tests, schedule, hardware components, or more generally speaking any kind of item able to represent a node of the project hierarchy. In addition, the SQuORE engine adds the findings and information from external tools, attaching them to the right artefacts with their values and meaning.

3.3 Data Consolidation

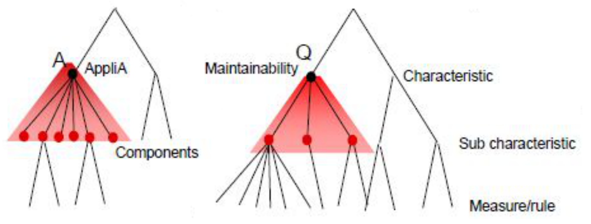


Figure 2: Artefacts and Quality Trees

Once the base measures have been collected, SQuORE computes derived measures as defined in the quality model and builds the quality hierarchy for every node of the artefact tree, as shown in Figure 2.

3.3.1 Quality Models

Quality models define how data are aggregated and summarised from the leaves up to the root artefact (usually the application or project): base measures collected by data providers are transformed into derived measures and associated to the different attributes of quality.

As stated before, there is no silver bullet for quality models[11]: one has to tailor the assessment method, considering the specific needs and goals of the development. In many cases existing models[1] constitute a good start, and they should simply be fine-tuned to fit most common needs. But for specific or uncommon situations, SQuORE proposes everything one would need in such a task, from basic operations on measures to complex, cross-tree computations.

[1]The default SQuORE setup proposes several models and standards: SQALE, ISO 9126 Maintainability and Automotive HIS are available right out-of-the-box.

3.3.2 Metris, Scales, Indicators

Raw measures give the status of a characteristic, without qualitative judgement. *Indicators* give this information, by comparing the measure to a scale. *Scales* define specific levels for the measure and their associated rating, which allow fine-tuning the model with specific thresholds and weights. As an example, the well-known cyclomatic complexity metric [17] for functions could be compared to a four levels scale, such that:

- from 0 to 7 rating is A (very good) and weight for technical debt is 0,
- from 7 to 15 rating is B (ok) and weight is 2,
- from 15 to 25 rating is C (bad) and weight is 4,
- above 25 rating is D (very bad) and weight is 16 because you really should refactor it.

Considering this, the cyclomatic complexity indicator gives at first sight the status of the intended meaning of the metric.

3.3.3 Action Items

Quite often, the dynamics of development depend on many different factors: as an example, if a function is quite long, has an important control complexity and many non-conformities, and a poor comment rate, then it should be really looked at although none of these individual indicators, taken separately, would be worse rising it. Action items serve this goal: the quality officer can define triggers, which can be any combination of indicators on the artefact tree, and SQuORE will create action items if one or all criteria are met. A helpful description about the problem and its resolution is displayed as well. Because they can be applied on any artefact type, the possibilities of action items are almost limitless. They are often the best way to implement experience-based heuristics, and automate long, tedious, and error-prone checking processes.

3.4 From Quality Assessment to Project Monitoring

Quality assessment, as a static figure, is the first step to project control: if you don't know where you are, a map won't help. The next step is to monitor the dynamics of the project, by following the evolution of this static status across iterations – this can be thought of as search-based decision making, as described by Hassan et al. in [5]. SQuORE proposes for this several mechanisms:

- Trends show at first sight how an artefact or attribute of quality did evolve.
- Drill-downs, sorts and filters help identify quickly *what artefacts* actually went wrong.
- The quality view helps understand *why* the rating went down, and what should be done to get it back to a good state.
- Action items help identifying complex evolution schemas, by specifying multiple criteria based on artefacts, measures and trends.

4 Use Cases

4.1 General Feedback

From our experience, there are some common reactions to a SQuORE evaluation:

- People are able to quickly identify issues and are not overwhelmed by the amount of information, which allows finding in a few minutes serious issues like missing breaks. Specialised tools are indeed able to uncover such issues, but due to the sheer volume of results they generate, it is not uncommon for end users to miss important results.
- People are glad to see that their general feeling about some applications is verified by pragmatic evidence. This re-enforces the representativeness of measurement and puts facts on words[2].
- Developers are concerned by their rating: the simple, eye-catching mark is immediately recognised as a rating standard. Further investigations help them understand why it is so, and how they could improve it[3].

[2]In other words: *I told you this code was ugly!*

4.2 Unit Test Monitoring

One of our customers needed to know what parts of software had to be tested first for maximum efficiency. Until now, the process was human-driven: files to be tested were selected depending on their history and recent changes, complexity of their functions, and their number of non-conformities. The team had developed home-grown heuristics gathered from years of experience, with defined attributes and thresholds.

We built a model with inputs from two external tools, QAC and Rational Test Real Time, and the SQuORE analyser. From these, we defined four main measures: non-conformities from QAC, RTRT, and SQuORE, plus cyclomatic complexity. Weights and computations were then selected in such a way that the files would get exponential-like ratings: as an example, if a file had only one of these main measures marked as bad, it would get a weight of 2. For two, three or four bad measures, it would get resp. a weight of 8, 16 or 32. This allowed quickly identifying the worst files in the artefact hierarchy.

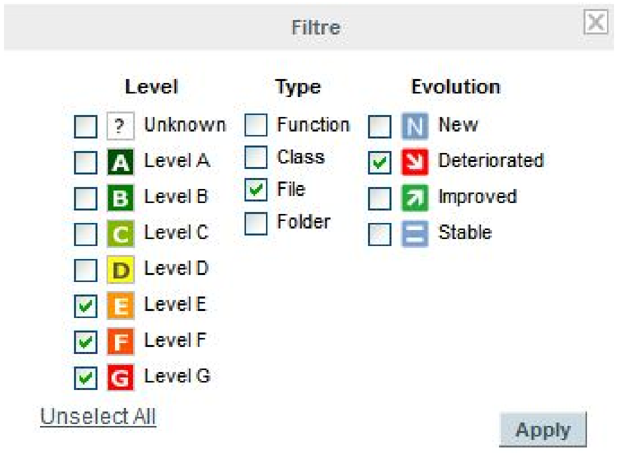
Folder ratings were computed according to the number of bad files they had under their hierarchy and their relative badness, which allowed focusing on worst components easily by sorting the artefact tree by folder ratings.

Action items were setup for the files that really needed to be checked, because they had either really bad ratings, or cyclomatic complexities or number of non-conformities that exceeded by far the defined criteria. Such action items allowed identifying problematic files hidden in hierarchies of good or not-so-bad files.

We were able to achieve the following goals:

- Define a standardised, widely-accepted mean of estimating testing efforts.

[3]In other words: *I suspected this part needed refactoring.*



Filtre

Level	Type	Evolution
☐ ? Unknown	☐ Function	☐ N New
☐ A Level A	☐ Class	✓ Deteriorated
☐ B Level B	✓ File	☐ Improved
☐ C Level C	☐ Folder	☐ Stable
☐ D Level D		
✓ E Level E		
✓ F Level F		
✓ G Level G		

Figure 3: Artefact filters

- Reproduce some gut-feeling mechanisms that had been thoroughly experienced and fine-tuned along the years by human minds, without having been explicitly formalised until now.

Dashboard graphs were setup to get immediate visual information on the rate of bad files in components. The evolution of component ratings also helped to identify parts of software that tended to entropy or bad testability, and take appropriate actions with development teams.

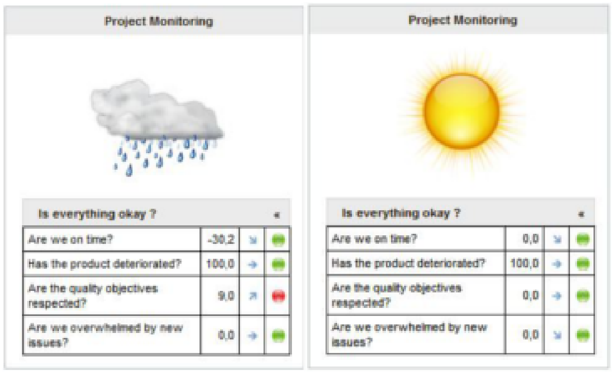
4.3 Project Monitoring

Another experience encompassed a full software project monitoring solution: the customer wanted to have a factual and unified vision on the overall progress of his developments.

SQuORE analyser was used for the source code quality assessment. Additional data providers were defined to retrieve data from change management, tests, and scheduling (as the number of open/closed tasks) tools.

Considering these entries, we defined the following axes in the quality model, as shown in Figure 4:

- *Are the quality objectives respected?* – based on the ISO 9126 maintainability assessment of code.



Is everything okay ?

Are we on time?	-30,2	↘	🟩
Has the product deteriorated?	100,0	→	🟩
Are the quality objectives respected?	9,0	↗	🟥
Are we overwhelmed by new issues?	0,0	→	🟩

Is everything okay ?

Are we on time?	0,0	↘	🟩
Has the product deteriorated?	100,0	→	🟩
Are the quality objectives respected?	0,0	→	🟩
Are we overwhelmed by new issues?	0,0	↘	🟩

Figure 4: Example scorecards

- *Are we on-time?* – an agile-like reporting on tasks completion.
- *Are we overwhelmed by new issues?* – the amount of defect reports waiting for treatments.
- *Was the product deteriorated?* – which reflects the state of test coverage and success: regression, unit, and system tests.

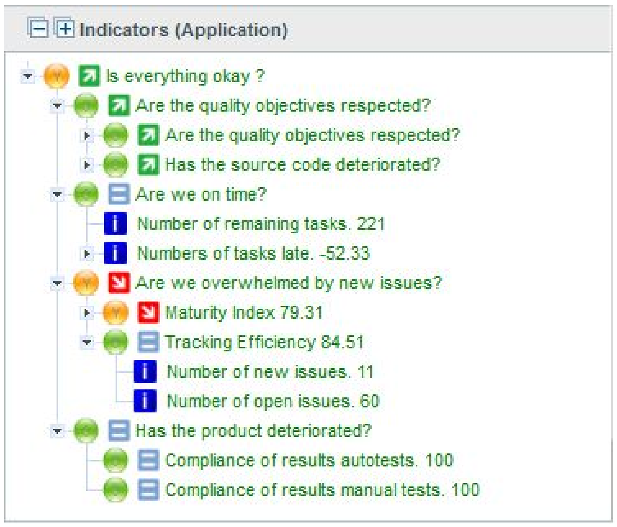
These axes are summarised in a global indicator (Figure 5), showing the composite progress of the project. In this case, we chose to take the worst sub-characteristic as the global rating to directly identify problems on progress.

Action items were developed to identify:

- Parts of code that had a bad maintainability rating, were not enough covered by tests, and got many defect change requests.
- Schedule shifts, when the number of opened tasks was too high for the remaining time and many change requests were incoming (opened).

The Jenkins continuous integration server was used to execute SQuORE automatically on a daily basis. Automatic retrieval of data and analysis was needed to ensure reliability and consistency of data – the constant availability being one of the keys to meaningful data.

Dashboards were defined for the main concerns of the project: evolution of the maintainability rating



Indicators (Application)

- Is everything okay ?
  - Are the quality objectives respected?
    - Are the quality objectives respected?
    - Has the source code deteriorated?
  - Are we on time?
    - Number of remaining tasks. 221
    - Numbers of tasks late. -52.33
  - Are we overwhelmed by new issues?
    - Maturity Index 79.31
    - Tracking Efficiency 84.51
      - Number of new issues. 11
      - Number of open issues. 60
  - Has the product deteriorated?
    - Compliance of results autotests. 100
    - Compliance of results manual tests. 100

Figure 5: Project Quality model

on files, evolution of change requests treatment, and failing tests. The scheduling information was represented using burn-down and burn-up charts, and dot graphs with trends and upper- and lower- limits.

We were able to:

- Provide real-time information about current state and progress of project. Web links to the Mantis change management system even allowed knowing exactly what actions are on-going.
- Get back confidence and control on project to team leaders and developers by enabling a crystal-clear, shared and consistent vision.

References

[1] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605, San Francisco, California, United States, 1976. IEEE Computer Society Press.

[2] CMMI Product Team. CMMI for Development, Version 1.3. Technical report, Carnegie Mellon University, 2010.

[3] Martin Fowler. Martin Fowler on Technical Debt, 2004.

[4] Maurice H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY, USA, 1977.

[5] Ahmed E Hassan and Tao Xie. Software Intelligence: The Future of Mining Software Engineering Data. In *Proc. FSE/SDP Workshop on the Future of Software Engineering Research (FoSER 2010)*, pages 161–166. ACM, 2010.

[6] ISO IEC. Iso/iec 15504-1 – information technology – process assessment. *Software Process: Improvement and Practice*, 2(1):35–50, 2004.

[7] ISO IEC. Iso/iec 25000 – software engineering – software product quality requirements and evaluation (square) – guide to square. *Systems Engineering*, page 41, 2005.

[8] ISO. ISO/DIS 26262-1 - Road vehicles Functional safety Part 1 Glossary. Technical report, International Organization for Standardization / Technical Committee 22 (ISO/TC 22), 2009.

[9] ISO/IEC. *ISO/IEC 9126 – Software engineering – Product quality*. 2001.

[10] Ho-Won Jung, Seung-Gweon Kim, and Chang-Shin Chung. Measuring software product quality: A survey of iso/iec 9126. *IEEE Software*, 21:88–92, 2004.

[11] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[12] C Kaner and Walter P. Bond. Software engineering metrics: What do they measure and how do we know? In *10Th International Software Metrics Symposium, METRICS 2004*, pages 1–12, 2004.

[13] Jean-louis Letouzey. The SQALE method for evaluating Technical Debt. In *2012 Third International Workshop on Managing Technical Debt*, pages 31–36, 2012.

[14] Jean-louis Letouzey and Thierry Coq. The SQALE Analysis Model An analysis model compliant with the representation condition for assessing the Quality of Software Source Code. In *2010 Second International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, pages 43–48, 2010.

[15] Nancy Leveson and Clark S. Turner. An Investigation of the Therac-25 Accidents. *IEEE Computer*, 26(7):18–41, 1993.

[16] J.L. Lions. ARIANE 5 Flight 501 failure. Technical report, 1996.

[17] TJ McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320, 1976.

[18] J.A. McCall. *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*. Information Systems Programs, General Electric Company, 1977.

[19] National Aeronautics and Space Administration. Mars Climate Orbiter Mishap Investigation Report. Technical report, National Aeronautics and Space Administration, Washington, DC, 2000.

[20] Peter G. Neumann. Cause of AT&T network failure. *The Risks Digest*, 9(62), 1990.

[21] National Institute of Standards and Department of Commerce. Technology (NIST). Software errors cost u.s. economy $59.5 billion annually, 2002.

[22] RTCA. DO-178B: Software Considerations in Airborne Systems and Equipment Certification. Technical report, Radio Technical Commission for Aeronautics (RTCA), 1982.

[23] Mary Shaw. Prospects for an engineering discipline of software. *IEEE Software*, (November):15–24, 1990.

[24] The Standish Group International Inc. The Standish Group International Inc. Chaos Technical report. Technical report, 2004.

[25] United States General Accounting Office. Patriot Missile Defense: Software Problem Led to System Failure at Dhahran , Saudi Arabia. Technical report, United States General Accounting Office, 1992.