29 April 2016

GN4-1 White Paper: Supporting the Service Validation and Testing Process in the GÉANT Project

Supporting Documentation

Work Package/Activity:	8/SA4
Task Item:	T1
Dissemination Level:	PU (Public)
Lead Partner:	PSNC
Document Code:	GN4-1-16-106BC68
Authors:	Marcin Wolski (PSNC), Bartosz Walter (PSNC), Szymon Kupiński (PSNC), Patryk Promińsk (PSNC), Ivana Golub (CARNet)

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 691567 (GN4-1).

Contents

Table of Figures

1 Introduction

1.1 Quality in software development

Software quality models and frameworks define various perspectives that present views on selected desired properties that a software system should demonstrate. The plenitude of quality models that have been proposed in recent 30 years results from differences in understanding what quality actually is, and how it can be interpreted in terms of software systems and services.

From various definitions and philosophies that stand behind quality, we briefly describe two, that are important to the modern meaning of quality.

Philip Crosby defined quality as conformance to requirements: every property that a user expects the product to possess should be reflected by the definition of quality. Moreover, in his opinion, such a property has to be measurable and manageable, so that it could be controlled and evaluated. Additionally, quality management should rely mainly on prevention, not the post-hoc appraisal.

However, Edward Deming, the father of statistical process control, presents a slightly different point of view. He outlined the importance of customer satisfaction to the perception of quality. This vision is different from the Crosby's plain conformance to requirements, because the customer is often unable to fully, completely and consistently specify the requirements, end expresses only vague or contradictory expectations. Therefore, in the Deming's opinion, quality is inherently subjective.

Various views on software quality result from different perception of quality. However, in order to evaluate the quality of a system, we need at least a simplified model that could reflect the desired properties of it.

1.2 Motivation

Recently, we observe an increasing interest in defining and applying quality models in software industry. This observation could be attributed to several factors: a quest for a competitive advantage on a market, pressure on reducing cost of software development and maintenance, or a need of a control and management framework that could guide managers in supervising the processes. Moreover, quality models help organisations to clearly express their goals, choose appropriate means for attaining the goals, and monitoring and evaluating the efforts made to reach them.

A quality model is a particularly important element of software governance for large organisations and enterprises, like the GÉANT project. In these cases, the model is expected to:

- Define a uniform approach for evaluating software development and maintenance, based on recognised standards, and supplied by evidence from the analysed processes.
- Provide better alignment of business goals with software development products.
- Facilitate monitoring the quality characteristics for a single product and comparing them between various products.

The model can help to normalise the assessment of software products across the whole organisation. As an example, we can consider test coverage feature. It roughly reflects the degree to which various qualities and functions of a system have been examined. Collecting data on this metric helps to track progress, but also to relate the results to other projects. Therefore, it contributes to a more objective assessment of how the system is developed and maintained. As a result, the root causes of the possible deviations could be identified and appropriate corrective actions can be taken to remedy the anomalies.

1.3 Intended use of the model

There are three main groups of stakeholders: project managers, service operation managers and software developers, involved in the software development process. They have different perspectives on software quality in the ecosystem, at different level of detail.

Below we summarise the prospective benefits for each group coming from the unified quality model:

- **Project managers** are given a tool presenting high-level software characteristics, which provide them with quantitative rationale for decision making, in particular the strategic ones, concerning product management. Understanding the role, scope and rationale for measurement are other advantages.
- **Service operation managers** receive a method for systematic approach to software audits processes, including, for example, artefact review against the quality and security. The results can be collected, archived and compared with historical data, which would give them better insight in software development practices and the trends.
- **Service developers** are provided with fine-grained data about the relevance of their efforts with respect to the project objectives.

As a result, the business goals for defining the quality model, are threefold:

- Identify quality-sensitive processes and products, relevant for various stakeholders.
- Provide a definition of quality, understood as a reference point for evaluation.
- Provide a means to control and track quality characteristics.

Moreover, this intended model is meant to be consistent with the ITIL CSI concept (Continual Service Improvements). The need for software improvements is identified and implemented, based on a systematic approach to measurement and analysis of stakeholder expectations, which follows the Deming's cycle: *plan/do/check/act*.

2 Overview of software quality models

2.1 McCall's model

The first public model for evaluating software quality was proposed by McCall in 1977 for US Department of Defence [1]. The primary objective of the model was to help developers in constructing better software by improving the development process, and to bridge the gap between the development and users by outlining the shared points of interests.

The McCall's model comprises three perspectives: product revision (understood as the ability to change), product transition (thought as the ability to run in various environments) and product operation (reflecting the way in which system is operated).

The model defined a hierarchical structure made of three levels:

- **11 quality factors** (in three groups), which define the user view of quality.
- **23 quality criteria** that reflect the developer's perspective.
- **A set of metrics**, which provide quantifiable means for measuring and controlling the qualitative properties of interest.

The quality factors are linked to quality criteria with non-unique relationships, which means that a single criterion can support several factors. For example, reliability (a factor) is backed by accuracy, fault tolerance and consistency, but consistency is also connected to correctness – another factor in the model.

The general concept behind the model is that a synthesis of quality factors should provide a complete perspective of the software quality.

2.2 Boehm's model

A similar philosophy is present in the model proposed by Boehm in 1978 [2]. It also defines a hierarchy of desired properties, criteria and metrics, but is more focused on qualitative evaluation of the subject system with a set of measurable attributes and metrics. At the top of the model, three high level characteristics are present: utility, maintainability and portability, which reflect the user's point of view. They are supported by **seven quality factors**: reliability, usability, testability, understandability or flexibility, which constitute the internal qualities of the system. Subsequently, they are related to more primitive characteristics, reflected by a number of metrics. Such a structure defines a clear meaning of each characteristic and its contribution to the overall system quality.

2.3 ISO 9216

A slightly different perspective is presented in ISO 9126 standard (or its successor, ISO 25010). It is partially based on the McCall's and Boehm's models, inheriting their structure and several features. This standard also explicitly identifies internal and external quality characteristics as separate groups. Specifically, there are **six quality factors**: reliability, efficiency, maintainability, portability, usability and functionality.

This model, unlike its other models, explicitly identifies their internal and external quality characteristics of a software system.

2.4 Tailored models

Tailored models emerged in response to the observed shift in software development from constructing systems from scratch to connecting ready components delivered by third-party vendors. The components actually are not developed, by rather composed and configured, which significantly affects the entire development process and the responsibility of the system developers. As a result, not all quality characteristics present in other models are relevant for such systems, but, on the other hand, also new quality attributes emerge and gain importance. Tailored models allow for selecting the interesting high-level characteristics and organising the data collection process so that it supplies the model only with data relevant for the model.

2.5 Open-source models

The rising popularity of open source software, which offers various levels of freedom in acquiring, reading and modifying code of software libraries, resulted in development of quality models targeted at this specific method of constructing and maintaining software.

CapGemini OS Maturity Model [9] is based on the maturity indicators that belong to two groups: product and application-related.

QualOSS Model [10] is different in that it comprises three characteristics, related to product, community and process. These characteristics are backed by a number of sub-characteristics and individual metrics. Moreover, this model underlines the importance of the context and the users for the actual quality perception.

3 Characteristics of GÉANT-originated software systems

GÉANT covers a series of EU-funded scientific projects to support the design, implementation, operation and maintenance of a pan-European high-capacity network for scientists. As part of the project to install and manage the resulting network, a number of software applications and tools have been developed. These are currently in use within and outside the project and have been extended and actively maintained.

Along with the process of maturation of the available software components in the GÉANT ecosystem, a need emerged to create a model that would assist managers, developers and users in assessing the current state of the software quality, planning for Continual Service Improvements (CSI) activities or providing necessary security and risk-mitigation actions.

4 Outline of the software quality model

4.1 Requirements specific to GÉANT

Software development in GÉANT has a number of distinctive features which have their origins in the network domain the software products target, but also in the characteristic of software developers and users [3], which includes the following:

- Software development commitment – Software developers and NRENs initially declare the expected effort they are willing to contribute to the project. The effort is usually constant within the scope of the project, but it may change, depending on the internal schedule, preferences and commitments of the entities to other projects.
- Service-orientation – Software systems are usually used as an infrastructure for delivering network services to end-users. The recipient of the service is not always equivalent to a user of the OSS component supporting that service.
- Diversity of approaches to software development – GÉANT imposes a general framework for software development, but particular teams can choose different styles and approaches to software development.
- Expectation of high quality solutions – Software products in GÉANT are for demanding and knowledgeable users, maintainers and owners, and are based on highly developed advanced networks and systems. The operational environment, including people and existing systems, is conservative in accepting new solutions for daily use.
- Limited end-user involvement – End-users are usually not actively involved in the software development throughout its lifecycle. They are often domain experts with limited time available to contribute to the project.
- Dispersed location – Development teams are widely distributed geographically [4].

4.2 Structure

The proposed GÉANTQM model inherits the structure and several individual characteristics from the Boehm/McCall/ISO9216 models. It combines selected features from other models in open source and commercial domains, but also supplements them with items specific to the externally-funded systems with a limited number of users. As a result, it is a good fit to the specific requirements and quality perspective mandated by GÉANT-related projects.

The structure is composed of three tiers:

- **Tier one** defines and describes the high-level quality characteristics that are crucial for the success of a project. These characteristics cannot be directly measured.
- **Tier two** defines middle-level quality sub-characteristics that reflect how the characteristics from tier one are addressed and evaluated. Sub-characteristics are subject to measurement.
- **Tier three** identifies metrics supporting the Tier two sub-characteristics. Some of the metrics play the role of Key Performance Indicators (KPIs).

The general structure of the model is presented in Figure 4.1:

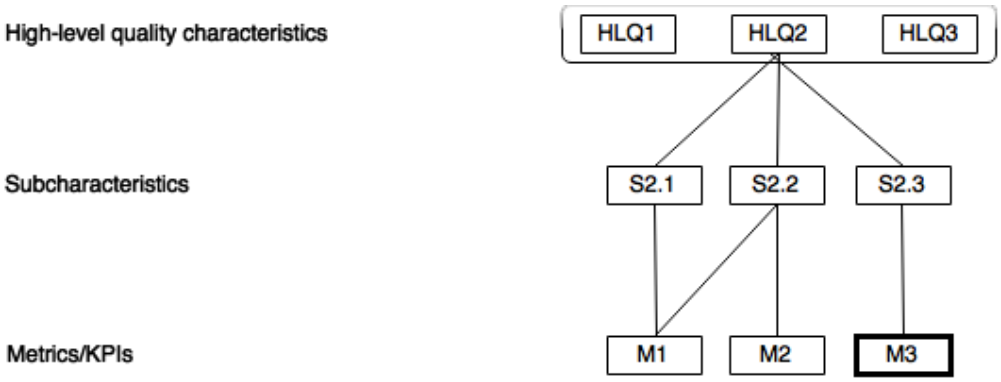

Figure 4.1: The GÉANTQM model

4.2.1 Tier one: high-level quality characteristics

The classical models usually define very general objectives for quality. Rather than providing detailed information, they envision main directions or perspectives that are important for the system stakeholders.

The specific of GÉANT-originated projects needs to be reflected by the selection of the high-level characteristics: some of the classical ones are absent, and the priority of some other is changed.

The model comprises five characteristics:

A. **Functionality**, which reflects the presence and availability of a set of functions that satisfy the explicitly stated or implied needs of users.
B. **Maintainability**, which comprises a set of attributes that bear on the effort needed to make specified modifications.
C. **Marketability**, which reflects the competitive strength of the system on the given market.
D. **Reliability**, which describes the capability of software to maintain its level of performance under stated conditions for a stated period of time.
E. **Usability**, which reflects the effort needed for using the subject system by a specified set of users.

4.2.2 Tier two: middle-level quality sub-characteristics

The second tiers decomposes characteristics into measurable, narrowly defined properties that are vital to the quality of the system. They define various possible meanings of the characteristics, but still without providing direct means of measurement (which are defined in Tier three).

A. Functionality

- **A.1** Suitability: appropriateness of the system's functions.
- **A.2** Accuracy: the required level of precision available in the system.
- **A.3** Interoperability: the system's ability to interact with other systems (in terms of protocol and format compliance).
- **A.4** Security: the control of unauthorised access to system's functions and services.

B. Maintainability

- **B.1** Stability: the ability to handle unexpected events during the operation.
- **B.2** Analysability: the ability to identify the root cause of a failure within the software.
- **B.3** Changeability: the degree to which the system can be changed or fixed.
- **B.4** Testability: the degree to which the system can be verified against specification.
- **B.5** Supportability: the ability to provide users with appropriate help or support.

C. Marketability

- **C.1** Market survivability: the ability to survive on the given market.
- **C.2** Market penetration: the share of market.
- **C.3** Market development: the ability to extend the market share.
- **C.4** Market adaptability: the ability to adjust the system to changing market needs.
- **C.5** Development velocity: the amount of functionality delivered in a time unit.

D. Reliability

- **D.1** Maturity: the frequency of failures.
- **D.2** Recoverability: the ability to restore operation of a failed system.
- **D.3** Fault tolerance: the ability to continue operation in case of system's failure.

E. Usability

- **E.1** Learnability: the ability to learn how to operate the software.
- **E.2** Understandability: the ease of which the systems functions can be understood.
- **E.3** Operability: the ability to easily operate the system by a given user in a given environment (user friendliness).

4.2.3 Tier three: metrics and KPIs

Tier three provides measures that quantify the sub-characteristics and convert them into numbers. The choice of metrics depends on the availability of data.

Some of the metrics can also play the role of Key Performance Indicators (KPIs). They define quantifiable measures that are correlated with the objectives of the organisation. In most cases, they reflect the business performance, but some process-related KPIs are also used. KPIs help to identify the core factors that contribute to success of the organisation, and focus maximising them. The number of KPIs in an organisation may vary, although it is recommended to have no more than 20 items.

The metrics are collected in two groups: Business-oriented (or external metrics), and Development-oriented (or internal metrics).

4.2.3.1 *External metrics*

The external metrics are concerned with the quantities directly related with the product and its market. Generally, they reflect the customer's point of view and their perception of quality.

4.2.3.2 *Internal metrics*

The internal metrics are focused on parameters of the development process that are not visible to customers, but have been proven to impact the external quality of the resulting product.

4.2.3.3 *Remarks*

KPIs can have different scope. Some of them are relatively easy to interpret and compare in entire organisation, between projects. Other KPIs are meaningful only in the context of a single project, and their definitions need to be adjusted for every project (e.g. usability measures).

Moreover, some metrics definitions vary depending on the context and the standard that defines them. For example, the commonly used term "bug" in software development is decomposed into distinct categories of a defect and a problem in ITIL-originated processes. A defect describes the externally visible deviations from the normal operation, and a problem refers to the underlying cause of the defect. Therefore, the defect-related metrics support external, and the problem-related ones – internal KPIs.

4.2.4 Data sources

The choice of the metrics to be used for the project is guided by the availability of data. Since GÉANT has produced several different software tools, historical data is at least partially available, or it can be retrieved from the archived artefacts. In particular, data currently available includes:

- **Audits**: formal project reviews, conducted by qualified and trained staff, following a pre-defined checklist.
- **Code metrics**: several basic code metrics collected by regular automated reviewing software tool (such as the open-source SonarQube). The following metrics are included: LOC, CC, comments density, number of commits, number of unit test cases.
- **Reports of code smells and anomalies**: reports generated by automated reviewing software tools, including PMD and Checkstyle; they highlight issues in the code that should be given attention or that require action.
- **Issue tracking data**: data from Jira bug tracker, including number of open issues, dated issues, etc.

Data from other sources can be supplemented and collected if needed.

5 Quality evaluation

5.1 Quantification

Various metrics have different units, values and meaning. Therefore, they cannot be directly interpreted in a uniform way. On the other hand, for assessing the overall quality of a software system, we need a single measurement scale.

A generic scale can be used to transpose the original metrics into a uniform measurement. The scale has five steps, which partition the domain of the metric into thresholds. Every threshold is given a numeric score describing its impact on the analysed sub-characteristic:

Very Low	Low	Medium	High	Very High
1	2	3	4	5

The other approach, which was used during the initial model validation is to normalise values from within the range 0 to 1, based on historical data for the metric as minimum and maximum values (or by expert judgement, if the data is not available). Additionally, their monotonicity is adjusted, so that higher values of each calculated factor represent a more positive evaluation.

The process of defining the thresholds for every metric is called calibration and is based on the historical data. In particular, it should encompass the regression of the evaluation scores based on the previous evaluations conducted on the same project. We recommend collecting the threshold from the first three evaluations, and later from every third evaluation.

The thresholds are project-based, which means they can differ across projects. However, the values acquired from various projects can affect other projects.

As individual sub-characteristics can be supported by more than a single metric, we can take them into account by calculating a mean of their values.

As a result, every sub-characteristic can be described by a single value from within the range 1–5.

5.2 Aggregation of sub-characteristics

The high-level characteristics describe various and diverse dimensions of quality, so it is impossible to aggregate their values without observing negative effects (like compensation) and loss of information.

However, the sub-characteristics within a single high-level characteristic seem to be coherent enough to be effectively combined into a single evaluation. As the relative importance of the sub-characteristics is variable in different systems, a simple mean value is not suitable for aggregating them. A popular method is the weighted mean, where weights represent the relative importance of individual characteristics, and are defined based on the opinion of the evaluator. If several evaluators are employed, the mode of their evaluations is taken instead.

The use of weights in the metric aggregation process could be applied for profiles of different services types. Based on the nature of a service, for example user portal or a backend service for a Network Operations Centre (NOC), different sets of weights should be determined based on historical data. This is the customisation operation to take into account business requirements to support the comparison of services with the same characteristics.

When using the Likert scale [5], every high-level characterising is also described by a single value from range 1–5. The overall quality of the analysed system is described by a vector of values, which represent each high-level characteristics.

Such aggregation hinders comparison of the results of the evaluation, in particular between various projects. When comparing different releases of a single project, the proposed method of aggregation is valid if the weights and the set metrics remain unchanged. However, if a variation is inevitable and well justified, the evaluation scores for the previous releases should be recalculated and adjusted as well.

5.3 Visualisation

Visual presentation of the quality helps evaluators to understand the present state of the quality and recent changes. We recommend a spider web chart as a method of presenting quality evaluation results. It provides the following advantages:

- Spider web chart accommodates a number of dimensions (high-level characteristics) without mixing or confusing them.
- Spider web chart allows for representing reference values or a reference profile together with the current values.
- Spider web chart allows for presenting validity thresholds for each dimension.

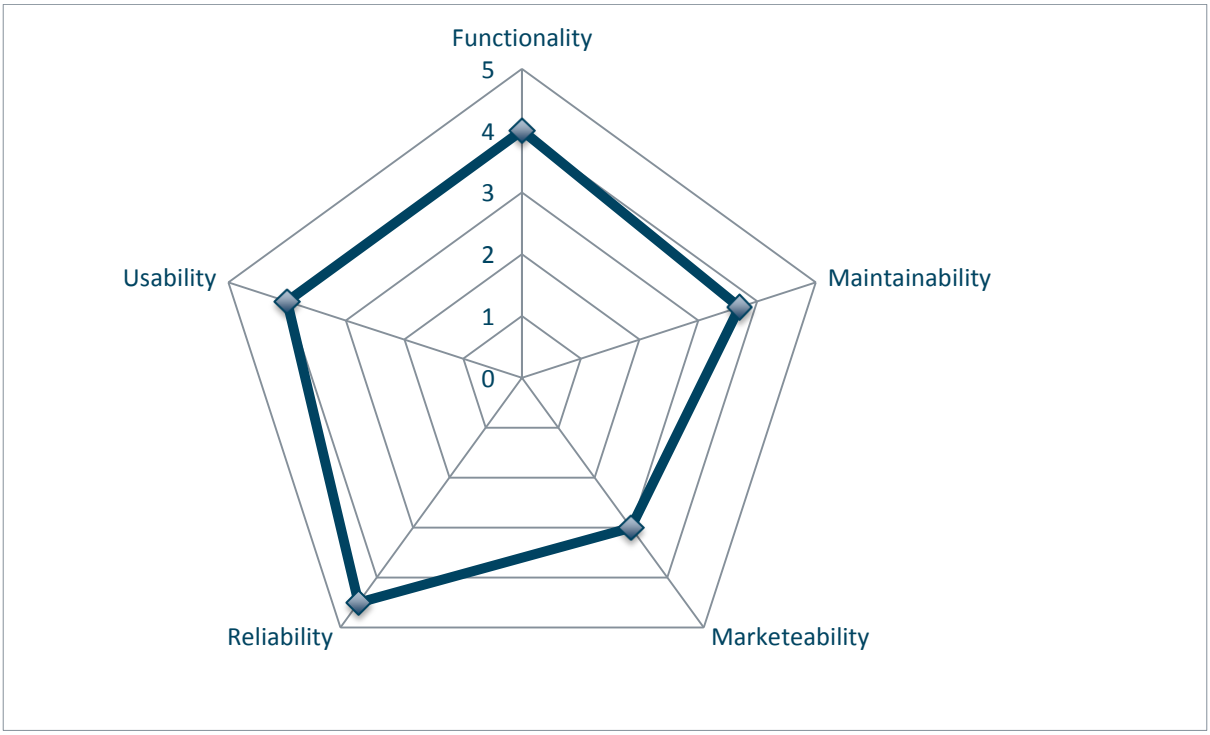

Figure 5.1: An example of the Spider Web chart

5.4 Summary and future work

The presented model has been developed and implemented within SA4 T1. For the first stage, the model was validated on the basis of two software products from the GÉANT ecosystem [8].

Early assessment of the framework performance shows that it is particularly useful in tracking the quality changes within individual projects, while comparison across projects produces ambiguous results. However, it helps in identifying quality areas requiring improvement. Furthermore, the quality of the framework itself relies strictly on the quality of the gathered data, e.g. software development process (Jira), collected answers from the survey stating about the availability or existing results from the software code audits. Thus in practice the number of useful and valuable metrics may vary and the framework should be tuned prior to the measurement and evaluation process.

Essentially, the framework would extend the current software quality evaluation process in the GÉANT project, which is a part of the Service Validation and Testing process [6], towards the objective software evaluation based on the metrics and measurements. Plans for further development of the framework include several enhancements. For example, the review and prioritisation of metrics could allow for defining various profiles of the framework, with metrics subsets classified to refer the project lifecycle (prototype, pilot and production), availability of the data sources (e.g. software audits) and other factors.

References

[1] James McCall: Factors in Software Quality: Preliminary handbook on software quality for an acquisition manager. Vol. 1-3. ADA049055. General Electric, Nov. 1977.

[2] Barry W. Boehm: Characteristics of software quality. Vol. 1/1978, NorthHolland Publishing Company, June 1978.

[3] Vilmos Bilicki et al.: Failure and success – how to move toward successful software development in Networking, TNC, Dublin, 2014.

[4] Anthony R. Hendrickson et al.: "Virtual teams: technology and the workplace of the future". In: The Academy of Management Executive (1993-2005) 12.3 (1998), pp. 17-29.

[5] Rensis Likert: The technique for the measurement of attitudes". Archives of Psychology, Vol. 140, pp. 1-55.

[6] M. Wolski (PSNC), I. Golub (CARNet), G. Frankowski (PSNC), A. Radulovic (MREN), P. Berus (PSNC), S. Medard (RENATER), S. Kupinski (PSNC), T. Appel (DFN/LRZ), T. Nowak (PSNC), S. Visconti (GARR), I. Smud (CARNet), B. Mazar (CARNet), B. Marovic (AMRES), P. Prominski (PSNC) "Deliverable D8.1 Service Validation and Testing Process"

[7] SA4 T1 generic questionnaires for users and developers of services, https://wiki.geant.org/display/gn41sa4/Surveys+and+questionnaires

[8] Appendix A – Software quality model validation. (The results document is stored on the geant.org intranet in SA4 space)

[9] Duijnhouwer FW, Widdows C. Open Source Maturity Model. Capgemini Expert Letter, 2003.

[10] Soto M, Ciolkowski M. The QualOSS open source assessment model measuring the performance of open source communities. Proc of ESEM 2009 Conference, IEEE Press, pp. 498-501 (doi: 0.1109/ESEM.2009.5314237)