XIX IMEKO World Congress
Fundamental and Applied Metrology
September 6–11, 2009, Lisbon, Portugal

SOFTWARE QUALITY CHARACTERIZATION OF THE FLEXIBLE FRAMEWORK FOR MAGNETIC MEASUREMENTS AT CERN

Pasquale Arpaia[1,2], Vitaliano Inglese[2,3], Giuseppe La Commara[1]

[1] Department of Engineering, University of Sannio, Benevento, Italy
arpaia@unisannio.it
[2] CERN, Dept. TE (Technology), Group MSC, Geneva, Switzerland
Vitaliano.Inglese@cern.ch
[3] Department of Electrical Engineering, University of Naples - Federico II, Napoli, Italy

Abstract – The paper deals with the measurement of software quality in frameworks for automatic test systems. In particular, the quality characterization of software components inside the Flexible Framework for Magnetic Measurements, developed at the European Organization for Nuclear Research (CERN), is illustrated with respect to the ISO 9126 reference model through the introduction of suitable metrics. Experimental results for code quality assessment are finally reported.

Keywords: Automatic measurement systems, software quality, software metrics.

1. INTRODUCTION

Quality is a key issue in software development. The quality of a system is the result of the quality of its elements and their interactions. Although software quality can be described from different perspectives [1], [2], [3], it can be defined in a general way as the capability of a software product to satisfy stated and implied needs when used under specified conditions [4]. Pursuing software quality is always worthwhile, since the cost of achieving a high quality level is widely overtaken by the cost of nonquality (having a software incapable of providing the required functionalities when needed).

The assessment of the software quality cannot be achieved without defining how to measure it in a quantitative way. For this reason metrics were introduced. The term *metric* is defined as a measure of the degree to which a process or product possesses a certain quality characteristic [5].

Even though the original motivations for deriving software measurements were almost entirely managerial (managers wanted to predict project costs at early stages in software lyfe-cycle and assess the productivity of the personnel [6]), very soon many researches concluded that no meaningful measures would be possible without consideration of the quality of the software produced. Anyway, metrics must be evaluated within the frame of a quality model to avoid their misuse. A *model* is an abstraction of reality, allowing to discard useless details and view an entity or a concept form a particular perspective [6], understanding the interactions among the parts forming the whole system of interest. In order assess the quality level achieved by an entity, a model defines (i) the entity and its attributes being measured, (ii) domain and range of the resulting measures, (iii) meaning of the single measures, and (iv) the relationships among several measurements.

Measures can be used to estimate future characteristics from previous ones or to determine the current condition of a process, product, resource. Therefore another characteristic of models is that they distinguish, as main aim of the measures, the prediction from the assessment.

A considerable amount of work has been devoted to the formulation of so-called quality models. One of the first was proposed by Gilb [7], according to whom any quality characteristic can be measured directly. The quality concept is broken into component parts until each can be stated in terms of directly measurable attributes. Other models were proposed by Boehm [5] and McCall [8]. These hierarchical models are based on the assumption that there are a number of important high level quality *factors* that are determined by lower level *criteria* supposed much easier to measure than the corresponding factors. Actual measures, *metrics*, are proposed for the criteria. The model describes all the relationships between factors and criteria, so that the former can be quantified in terms of measures of their dependent criteria. This conception of modeling quality was more recently at the basis of international efforts that led to the development of a standard for software quality measurement, defining a software quality model (ISO 9126 [9]-[12]), the software measurement process (ISO 15939 [13]), and the software evaluation process (ISO 14598 [14]). The standard recommends six quality characteristics, further refined in subcharacteristics, as basic set for quality evaluation.

Given a particular problem, techniques like the Goal-Question-Metric [15] can help identify which measures are to be taken into account to monitor and improve quality in the specific case.

In this paper, the approach proposed in the standard ISO 9126 is employed as reference model for the quality characterization of the Flexible Framework for Magnetic Measurements (FFMM [16]) developed at CERN in the frame of a cooperation with the University of Sannio. In the

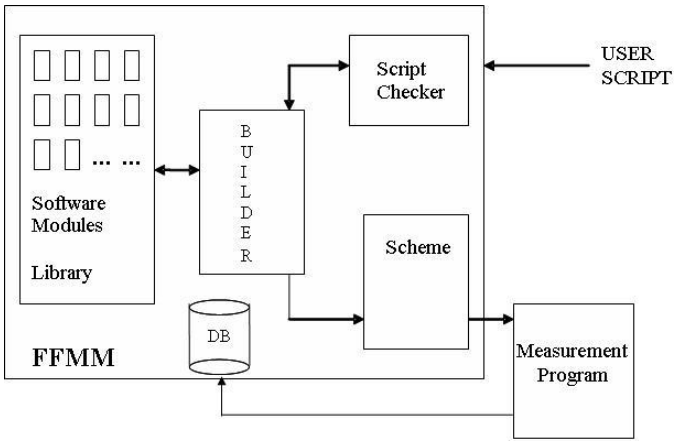

Fig. 1. The FFMM architecture [16].

following, Section 2 describes the architecture of FFMM, the reference quality model, and the metrics chosen for the characterization, Section 3 presents the experimental results obtained on the release 3.0 of FFMM.

2. SOFTWARE QUALITY IN FFMM

The FFMM is a software framework for magnetic measurement applications based on Object Oriented Programming (OOP), and Aspect-Oriented Programming (AOP) [17]. Its basic ideas and architecture are discussed in [18], [19]. In particular, FFMM aims at supporting the user in developing software for automatic measurement systems by maximizing quality in terms of flexibility, reusability, maintainability, and portability, without neglecting efficiency, vital in actual test applications. Moreover, the requirements for a wide range of magnetic measurement applications, such as needed for the test of superconductive magnets for particle accelerators, have to be satisfied.

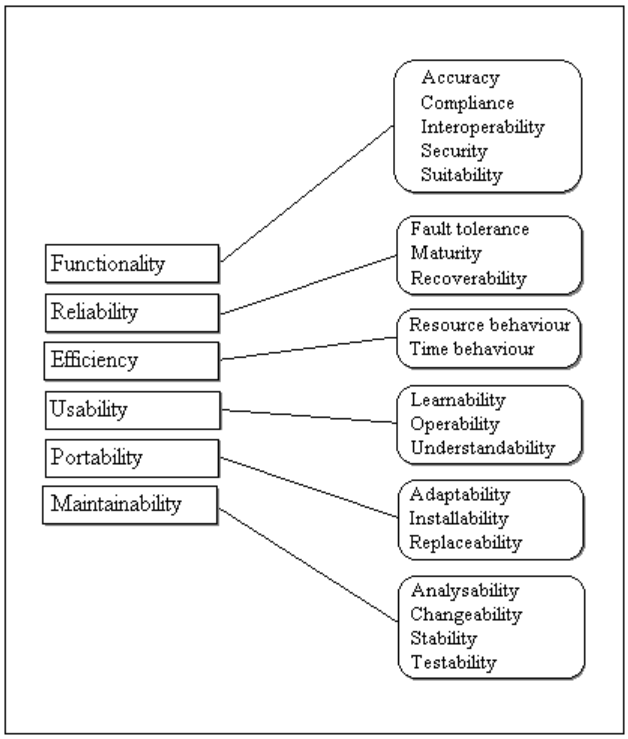

Fig. 2. The ISO 9126 quality model.

In Fig.1 [16], the FFMM architecture is illustrated. A test engineer (end user) produces a description of the measurement application, *User Script*, whose semantic and syntactic correctness is verified by the *Script Checker*. Then, from the *User Script*, the *Builder* assembles the *Measurement Program*, according to the architecture of the *Scheme* by picking up suitable modules from the *Software Module Library*. If some modules are not available in the library, a template is provided to the user (administrator user) in order to implement them according to a suitable predisposed structure. Once debugged and tested, the *Measurement Program* will be stored in the *Database* in order to be reused.

If software quality requirements are not clearly stated, they could be interpreted in different ways by different people. This could result in software that is inconsistent with user expectations and of poor quality. As said before, international standards were therefore developed to address this issue and provide a definition of software quality, along with guidance for its evaluation [9]-[14]. This paper aims at the assessment of the quality level achieved by the release 3.0 of FFMM according to the guidelines of these standards.

The software quality model provided by the standard ISO 9126 [9] defines six quality characteristics (Fig. 2):

- Functionality: the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
- Reliability: the capability of the software product to maintain a specified level of performance when used under specified conditions.
- Usability: the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.
- Efficiency: the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
- Maintainability: the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.

Table 1. Quality characteristics, metrics, and target values.

Maintainability		**Portability**	
Class Depending Child (CDC)	FALSE	Class Depending Child (CDC)	FALSE
Class Depth (DEPTH)	≤ 7	Class Depth (DEPTH)	≤ 7
Essential Complexity (Ev(G))	≤ 4	Coupling between Objects (CBO)	≤ 2
Multiple Inheritance (FAN IN)	≤ 1	Multiple Inheritance (FAN IN)	≤ 1
Access to Protect or Public Data (PUB_ACCESS)	= 0	Lack of Cohesion of Methods (LOCM/LCOM)	≥ 75%
Access to Public Data Definition (PUB_DATA)	= 0	Response for Class (RFC)	≤ (WMC*DEPTH)+1
Response for Class (RFC)	≤ (WMC*DEPTH)+1	Weighted Methods for Class (WMC)	≤ 14
Cyclomatic Complexity (v(G))	≤ 10		
Weighted Methods for Class (WMC)	≤ 14		
Module Design Complexity (Iv(G))	≤ 7		
Design Complexity ($S_0 = \Sigma(Iv(G))$)	*smaller the better*		

- Portability: the capability of the software product to be transferred from one environment to another.

The standard defines an additional quality characteristic:

- Quality in use: the capability of the software product to enable specified users to achieve specified goals with effectiveness, productivity, safety and satisfaction in specified contexts of use.

The quality characteristics have defined sub-characteristics and the standard allows for user defined sub-subcharacteristics in a hierarchical structure. The ISO framework is completely hierarchical, each subcharacteristic is related to only one characteristic. The defined quality characteristics cover all quality aspects of interest for most software products and as such can be used as a checklist for ensuring a complete coverage of quality.

The quality model defines three different views of quality:

- Software quality in use
- External software quality
- Internal software quality

The software quality in use view is related to application of the software in its operational environment, for carrying out specific tasks by specific users. External software quality provides a black box view of the software and addresses properties related to the execution of the software on computer hardware and applying an operating system. Internal software quality provides a white box view of software and addresses properties of the software product that typically are available during the development. Internal software quality is mainly related to static properties of the software. Internal software quality has an impact on external software quality, which again has an impact on quality in use.

In the following, the ISO reference model is employed for the assessment of the internal quality of FFMM source code. At this stage, the release 3.0 is not yet widely used by users others than the developers. Moreover, innovative user interfaces are under development but not yet employed, contributing to make premature the evaluation of the quality in use. The proposed metrics are product-oriented (such as size, maintainability, portability), rather than process-oriented (time, costs, productivity), and are meant to be employed for quality assessment and improvement. Metrics specifically developed for object-oriented systems evaluation are also considered, while modularity and performance of the AOP fault detector included in the framework are discussed in a specific paper [20].

3. EXPERIMENTAL RESULTS

In this section, the results of the FFMM 3.0 software quality characterization are discussed. The analysis was carried out by means of the tool UnderstandC++ [21]. Heuristic thresholds were employed, as proposed in literature [22]-[24], in order to define the metrics target values. An example of metrics, quality characteristic they affect, and target values is reported in Tab. 1 [22], [25]. In particular, the complexity metrics (such as Essential Complexity and Cyclomatic Complexity) measure the logic

Table 2. FFMM 3.0 size metrics summary.

FFMM 3.0 size metrics summary	
Blank Lines	4'115
Classes	96
Code Lines (LOC)	16'253
Comment Lines	6'977
Comment to Code Ratio	0.43
Declarative Statements	4'779
Executable Statements	8'642
Files	131
Functions	1'082
Inactive Lines	172
Lines	28'119

Table 3. FFMM 3.0 complexity metrics.

	Average	Max	Target	% OK (program units)
Essential Complexity (Ev(G))	1.2	19	≤ 4	99
Cyclomatic Complexity (v(G))	2.4	40	≤ 10	97

complexity of the software modules and hence the effort required for testing and maintain them. The object-oriented metrics (such as LCOM, FAN IN, CBO, RFC, WMC, DEPTH) measures the extent to which features typical of object-oriented systems are exploited (e.g. inheritance) or achieved (e.g. lack of coupling and cohesion).

Tab. 2 reports a short summary of size metrics computed on FFMM. Besides merely dimensional metrics such as the Lines of Code (LOC), the Comment to Code Ratio measures the percentage of comment lines with respect to the lines of code. A value between 20% and 35% is considered acceptable. Lower values are undesired, since they may significantly affect the maintainability. Anyway, also higher values (as in this case) are considered anomalous since they are likely to be due to commented code rather than to useful comments.

As far as the complexity metrics (Essential Complexity and Cyclomatic Complexity) are concerned, the results show that in FFMM there is space for improvement of the considered quality characteristics (Tab. 3). In particular, although the average complexities respect the heuristic upper bounds, the maximum values exceed them in a significant way. This implies that the complexity is concentrated in few points that need to be simplified in order to decrease the effort required for software testing and maintenance.

Analogous remarks can be made from the analysis of the object-oriented metrics (Tab. 4). All the metrics considered have acceptable average values but most of them (LCOM, FAN IN, CBO, RFC, WMC with the exception of DEPTH and CDC) show maximum values significantly exceeding the heuristic thresholds. For example objects showing a high degree of coupling (CBO) are difficult to maintain and reuse, classes with high cohesion (low LOCM) can probably

Table 4. FFMM 3.0 object-oriented metrics.

	Average	Max	Target	% OK (classes)
Class Depending Child (CDC)	FALSE	FALSE	FALSE	100
Class Depth (DEPTH)	1	4	≤ 7	100
Multiple Inheritance (FAN IN)	0.7	2	≤ 1	85
Response for Class (RFC)	20	138	≤ (WMC* DEPTH)+1	45
Coupling between Objects (CBO)	3.9	21	≤ 2	50
Lack of Cohesion of Methods (LOCM/LCOM)	0.42	1	≥ 0.75	40
Weighted Methods for Class (WMC)	13	103	≤ 14	74

be splitted in subclasses, high WMC and RFC imply a big effort for software development, learnability and maintenance, and so on. Again, an acceptable average quality level is partially compromised by some parts of the software that need improvement interventions. Conversely, the values of the FAN IN metric exceeding the threshold are the result of a conscious design choice, since all the devices implemented in FFMM inherit from two abstract classes. These two classes are completely independent from each other, therefore the multiple inheritance is not expected to cause any undesired side effects.

4. CONCLUSIONS

This paper presents the results of the software quality characterization of the release 3.0 of the Flexible Framework for Magnetic Measurements. The characterization was carried out with reference to the quality model ISO 9126 developed by the International Standard Organization. Both complexity and object-oriented metrics were evaluated. Although the results highlighted an acceptable average quality level, improvements are required in order to decrease the maximum complexity and to exploit more profitably the concepts of object-oriented programming. Furthermore, only the internal quality of FFMM source code was taken into account. As a consequence, the quality assessment relates more to the developer point of view than to that of the user. The characterization will therefore be completed by encompassing an evaluation of the external quality (including performance, vital in real time measurement applications) and the quality in use.

ACKNOWLEDGEMENTS

This work is supported by CERN trough the agreement LHC/AT/K1464 with the University of Sannio, whose support authors gratefully acknowledge.

REFERENCES

[1] D. Garvin, “What does “Product Quality” really mean?”, Sloan Management Review, Fall 1984, pp.25-45.

[2] B. Kitchenham, S. L. Pfleeger, “Software quality: the elusive target”, IEEE Software, Issue 1, Vol. 13, pp. 12-21, 1996.

[3] I. Tervonen, P. Kerola, “Towards deeper co-understanding of software quality”, Information and Software Technology 39 (1998) 995-1003.

[4] ISO 8402 *Quality Management and Quality Assurance – Vocabulary*, International Organization for Standardization, Geneva, 2nd Edition, 1994.

[5] B. W. Boehm, J.R. Brown, M. Lipow, “Quantitative Evaluation of Software Quality”, Proc. of the Second Intern. Conf. on Software Engineering, pp. 592-605, 1976.

[6] N. E. Fenton, *Software Metrics – A Rigorous Approach*, Chapman & Hall, 1991.

[7] T. Gilb, *Principals of Sofware Engineering Management*, Addison-Wesley, Reading, Mass., 1987.
[8] J. A. McCall, P.K. Richards, G.F. Walters, *Factors in Software quality*, Vol. 1, 2, and 3, AD/A049-014/015/055, Nat'l Tech. Information Service, Springfield, Va., 1977.
[9] International Standard ISO/IEC 9126-1, "Software Engineering – Product Quality – Part 1: Quality Model", International Organization for Standardization, International Electrotechnical Commission, 2001.
[10] International Standard ISO/IEC 9126-2, "Software Engineering – Product Quality – Part 2: External Metrics", International Organization for Standardization, International Electrotechnical Commission, 2003.
[11] International Standard ISO/IEC 9126-3, "Software Engineering – Product Quality – Part 3: Internal Metrics", International Organization for Standardization, International Electrotechnical Commission, 2003.
[12] International Standard ISO/IEC 9126-4, "Software Engineering – Product Quality – Part 4: Quality in Use Metrics", International Organization for Standardization, International Electrotechnical Commission, 2004.
[13] International Standard ISO/IEC 15939, "Software Engineering – Software Measurement Process", International Organization for Standardization, International Electrotechnical Commission, 2002.
[14] International Standard ISO/IEC 14598, "Software Engineering – Product Evaluation", International Organization for Standardization, International Electrotechnical Commission, 2000.
[15] R. Van Solingen, E. Berghout, *The Goal/Question/Metric Method*. McGraw-Hill Education, 1999.
[16] P. Arpaia, L. Bottura, M. Buzio, D. Della Ratta, L. Deniau, V. Inglese, G. Spiezia, S. Tiso, L. Walckiers, "A software framework for magnetic measurement at CERN", *Proc. of IEEE Instrumentation and Measurement Technology Conference*, Warsaw, Poland, May 1-3 2007.IMTC 07.
[17] Marc Eaddy, Thomas Zimmermann, Kaitlin D. Sherwood, Vibhav Garg, Gail C. Murphy, Nachiappan Nagappan, and Alfred V. Aho, "Do Crosscutting Concerns Cause Defects?," *IEEE Trans. on Software Engineering*, May 16, 2008.
[18] J. E. Beck, J. M. Reagin, T. E. Sweeney, R. L. Anderson, T. D. Garner, "Applying a component-based software architecture to robotic workcell applications", *IEEE Trans. on Robotics and Automation, Vol. 16, N. 3 pp. 207-217, Jun. 2000.*
[19] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J.M. Loingtier, J. Irwin, "Aspect-Oriented Programming", *Proc. of the 11th European Conference on Object-Oriented Programming (ECOOP*), Vol. 1241, pp. 220-242, Springer-Verlag, 1997.
[20] P. Arpaia, M. L. Bernardi, G. Di Lucca, V. Inglese, G. Spiezia, "An Aspect Oriented Programming-based approach for fault detection implementation in automatic measurements systems", in press on *Computer Standards & Interfaces*.
[21] http://www.scitools.com/products/understand/
[22] http://www.dia.uniroma3.it/~torlone/sistelab/annipassati/sbavaglia.pdf
[23] V. A. French, "Establishing Software Metric Thresholds", Int. Workshop on Software Measurement (WSM 99), Lac Supérieur, Canada, Sept. 1999.
[24] M. Lanza, R. Marinescu, S. Ducasse, *Object-oriented metrics in practice*, Springer, Berlin, 2006.
[25] http://www.scitools.com/documents/metrics.php