JOURNAL OF APPLIED QUANTITATIVE METHODS

ON QUALITY AND MEASURES IN SOFTWARE ENGINEERING

Ion I. BUCUR[1]
PhD, University Lecturer
Politechnica University of Bucharest, Romania

E-mail: ion.bebe.bucur@gmail.com

Abstract: *Complexity measures are mainly used to estimate vital information about reliability and maintainability of software systems from regular analysis of the source code. Such measures also provide constant feedback during a software project to assist the control of the development procedure. There exist several models to classify a software product's quality. These models often include different measures and on their basis it is established a degree to which the product satisfy each quality attribute. Each model can have a different set of attributes at the highest level of classification, and also, the attributes can be defined differently at all levels. Actually, more and more activities are based on computer programs and they become highly dependent on their quality. In principle, everyone agrees that quality is important, but few agree on what quality is. In this paper, we will present the most important models and standards for measuring software quality. Afterworlds', we will give some metrics for software complexity and we will explain its relationship with the quality.*

Key words: *quality, source code, ISO, metrics, cyclomatic complexity, complexity, testing, reengineering*

1. Introduction

The problem of quality's evaluation is quite an old one; it was approached long time ago and led to the publication of the first quality standards by ISO (International Standards Organization) since the late 80's. The goal of these standards was to eliminate products amateurism by certifying some of their values or qualities. Nowadays, the quality control methods are more and more implemented in all companies in order to provide products and services in conformity with the clients' demands. Organizations all over the world are more and more concerned with the raising of software products quality, which may lead to success in many directions, from programmed microwaves to watches and toys. The quality, when present, is transparent, but easily recognizable when missing.

Software complexity is one branch of software metrics dedicated to direct measurement of software quality attributes, being distinct to indirect software measures such as reported system failures, project milestone status, etc.

Complexity measures are mainly used to estimate vital information about reliability and maintainability of software systems from regular analysis of the source code. Such measures also provide constant feedback during a software project to assist the control of

the development procedure. During testing and maintenance, complexity measures are providing detailed information about software modules to help pinpoint areas of potential instability.

There exist several models to classify a software product's quality. These models often include different measures and on their basis it is established a degree to which the product satisfy each quality attribute. Each model can have a different set of attributes at the highest level of classification, and also, the attributes can be defined differently at all levels.

2. General quality standards

Different people may have different views on what software quality is. For some, it is a largely aesthetic and practical issue, dealing with the question of how efficiently and elegantly, a computer program performs a task and source code looks. For others, quality is defined as strict conformance to requirements and absence of bugs. In both cases, there are sets of practices that are either required, or highly useful in this pursuit.

Thus, the standards have been introduced in the attempt to assure some universal reference systems. They are used on a large scale since they assure a background for the organizations to define a quality model for a software product. This way, however, each organization has the possibility to specify with precision its own quality model. This can be done by establishing some reference values for the attributes quality.

In accordance with ISO 8402, the quality is in fact presented by a set of characteristics, which can be divided as follows:

- Economic characteristics: expressed by means of costs, resources economies, as well as productivity and growth performance.
- Social and psycho-sensorial characteristics: manifested by rendering profitable the creative elements, by eliminating the routine and the stereotypy as well as by operators assisted training;
- Technical characteristics: presented in the specialized literature and very well systematized by ISO 9126 - a standard that exclusively deals with the software systems evaluation.

Among the quality characteristics, there are a lot of subordination relations, the interdependence, hierarchy, unit, decomposition, and the complexity of these relations leads to the quality characteristics assembly to make up a system. The quality characteristics are aggregates of the quality attributes, which correspond to actual properties that the programming systems must have.

Further on, in this article, one will analyze the technical characteristics due to the fact that these are most important for software systems evaluation.

3. ISO 9126 - International standard for evaluating software products

In 1991, the ISO published its first international consensus on the terminology for the quality characteristics for software product evaluation (ISO 9126 / 1991). From 2001 to 2004, the ISO published an expanded version, containing both the ISO quality models and inventories of proposed measures for these models. The standard is divided into four parts which addresses, respectively, the following subjects: quality model, external metrics, internal metrics, and quality in use metrics:

- Quality models - ISO 9126-1.

- External metrics - ISO TR 9126-2.
- Internal metrics - ISO TR 9126-3.
- Quality in use metrics - ISO TR 9126-4.

Internal metrics are those that do not rely on software execution (static measures) while external metrics are applicable to running software. Ideally, the internal quality determines the external quality and this one determines the results of quality in use.

The quality model established in the first part of the standard, ISO 9126-1, classifies software quality in a structured set of factors as follows:

- Functionality - A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs. This characteristic has the following attributes: Suitability, Accuracy, Interoperability, Compliance, Security;
- Reliability - A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time. This characteristic has the following attributes: Maturity, Recoverability, Fault Tolerance;
- Usability - A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users. This characteristic has the following attributes: Learnability, Understandability, Operability;
- Efficiency - A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions. This characteristic has the following attributes: Time Behavior, Resource Behavior;
- Maintainability - A set of attributes that bear on the effort needed to make specified modifications. This characteristic has the following attributes: Stability, Analyzability, Changeability, Testability;
- Portability - A set of attributes that bear on the ability of software to be transferred from one environment to another. This characteristic has the following attributes: Installability, Conformance, Replaceability, and Adaptability.

The sub-characteristic *Conformance* is not listed above and applies to all characteristics. Examples are conformance to legislation concerning *Usability* or *Reliability*.

Each quality sub-characteristic (as Adaptability) is further divided into attributes. An attribute is an entity which can be verified or measured in the software product. Attributes are not defined in the standard, as they vary between different software products.

ISO 9126 distinguishes between a defect and nonconformity, a **defect** belongs to the application space being the nonfulfilment of intended usage requirements, whereas **nonconformity** is defined upon the application specification space and is defined as being the nonfulfilment of specified requirements.

4. Software complexity measures

Software measurement method is a rule designed for assigning a number of identifier to software in order to characterize it. It is essential to distinguish between the characteristics which one would wishes to measure and the way by which this characteristic is evaluated and appreciated. Code complexity, as an example, is a characteristic used to describe a piece of a code. There are many different measures to evaluate this characteristic. One could find in literature measures as the number of lines code lines, testability, easiness

in fixing error, code understandability and many other ways used to evaluate code complexity.

The term software complexity means mostly the degree to which a system or component has a design or implementation that is difficult to understand and verify (testable). Still, there is no total consensus concerning this definition, another interpretation sees the complexity as a measure of the resources expended by a system while interacting with a piece of software to perform a given task. If the interacting system is a programmer, then complexity is defined by the difficulty of performing tasks such as coding, debugging, testing or modifying the software.

All these definitions associate the software complexity with the difficulty or performing a task on the software. An implicit assumption is that software complexity correlates well with the work effort (man-hours) required developing or maintaining the software.

Among the most well-known attempts to measure the complexity are: Software Science, which deals with the difficulty of understanding the code, Cyclomatic Number, which deals with the code's complexity structure, and Information Flow, which deals with the relation between modules. During the last years, six metrics have been proposed to measure some baselines in terms of Object Oriented Design, like Number of Class, Number of Children, Depth of Inheritance Tree etc.

5. Complexity, Reengineering and Testing

There is in common usage hundreds of software complexity measures, ranging from the simplest, such as source lines of code, to the complex, such as number of variable definition/usage associations. It is essential to use a low complexity subset of these measures for implementation. One of the most important criterions for metrics selection is uniformity of usage. One can read mostly in all papers that the key idea here is *open reengineering*. The reason that makes *open systems* so popular for commercial software applications stems in the fact that the user is guaranteed a certain level of interoperability - it means that the applications work together in a common framework, and software systems can be ported across different hardware platforms with minimal effort. Complexity measurement using metrics is a primary request, but open reengineering extends to other modeling techniques such as flow graphs, structure charts, and structure-based testing. Common complexity measures as the *Halstead Software Science* metrics are a significant step up in value. Halstead measures were introduced in 1977 and have been used and experimented with extensively since that time. They are one of the oldest measures of program complexity. By counting the number of total and unique operators and operands in the program, measures are derived for evaluating program size, programming effort, and estimated number of defects. Halstead metrics are, in fact, independent of source code format, so they are able to measure intrinsic attributes of the software systems. Halstead metrics are considered by several authors as being a little bit controversial, especially in terms of the psychological theory behind them, but they have been used productively on many projects. The main weakness, however, is that the derived mathematical formulas of the main Halstead metrics are considerably unconcerned from the measured code, so there isn't a strong prescriptive component.

One can identify code of an application as being potentially unpredictable, but the Halstead theory doesn't say much about how to test it, if it is testable, or how to improve it, if one proves to be necessary. Despite these limitations, Halstead Software Science metrics are very helpful and constructive for identifying computationally-intensive code with many dense formulas, which represent possible sources of inaccuracy or errors that other complexity procedures are likely to miss. However, their properties are well-known and, in they have been shown to be a very strong component of the *Maintainability Index Technique* measurement of maintainability method.

The McCabe *Cyclomatic Complexity Measure* is very flexible and extensively used for software systems complexity evaluation, mostly for existing ones. It measures the number of linearly-independent paths through a program module. The McCabe complexity is one of the more widely-accepted software metrics; it is intended to be independent of language and language format. The complexity number is generally considered to provide a stronger measure of a program's structural complexity than is provided by counting lines of code, previously used. It is widely proposed as the foundation of every software complexity tool. It may be considered a broad measure of soundness and confidence for a software system. This complexity measure is based purely on the code's decision structure. It makes this method to be uniformly applicable across projects and languages being completely insensitive to cosmetic changes in code. Many studies have reported its correlation with errors in software code, so it is used to predict reliability. More significantly, experimental studies have shown that the risk of errors is rising for functions having cyclomatic complexity over 15, so one could consider it as a validated threshold for reliability screening. If a function has a cyclomatic number of 15, there are at least 15 (but probably more) execution paths through it. More than 15 paths are hard to identify and test. Functions containing one selection statement with many branches make up an exception. Also, this assessment can be performed step by step during development and can even be estimated from a detailed design. Considering a specified software module, one can easily calculate cyclomatic complexity, in a manual way, by counting the decision constructs in the code. This approach allows building up continuous control during project development, so that unreliable code is prevented early at the unit development stage. A reasonable upper limit cyclomatic number of a file is 100. Using automated tools one can verify code compliance at any stage of the project development. McCabe's cyclomatic complexity measure gives precise testing rules. Most complex function being most error prone piece of code has to be first considered in order to receive required testing.

One of the most successful measurement concepts, used for quantitative productivity levels is function point metrics. Software measure based on *function points* techniques (FP) reflects the user's view of a system's functionality and gives size as functionality. One unit (the function point) represents the amount of information processing that a module offers the user. The unit is viewed separately from the way in which the information processing is carried out in principle. This concept was introduced in the mid-1970s when IBM commissioned engineer Allan J. Albrecht and his colleagues to explore software measurement and metrics. IBM was motivated for this assignment by the growing impact of software quality within the company tied with the difficulties and obvious limitations of the ubiquitous *line of the code* metrics, used before.

Functional point data has two targets. First one is an estimation variable used mainly to evaluate the size of each software module, while the second one is intended as a

baseline metrics collected from older projects developed by same team and used conjunctively with estimation variables helping to devise cost and effort projections. Function points are categorized into five groups: outputs, inquiries, inputs, files, and interfaces. Basically the approach proceeds to identify and count of unique function types:

- external inputs (file names, as example);
- external outputs (e.g. reports, messages);
- queries (interactive inputs needing a response);
- external files or interfaces (files shared with other software systems);
- internal files (invisible outside the system).

Function point metrics extended among many companies because they did provide substantial benefits to their users. The first benefit of function point metrics is that they are offering substantial ability to the software industry in order to carry out economical based studies for developed products [05, 09, 10, and 24]. These metrics have become the standard for studying topics associated with software, including but not limited to:

- Outsource contracts;
- Quality baseline and benchmarks ;
- Process improvement economics;
- Litigation analysis;
- Productivity baseline and benchmarks.

Function points are powerful metrics but successful usage of them is not a trivial task. Accurate counting of function points metrics require good training. Main feature of function point metrics is the fact that them are able to measure economic productivity or the defect volumes found in software requirements, design, and user documentation as well as measuring coding defects.

6. Conclusions

The quality of software is given by its capacity to be used effectively, efficiently and comfortably by any user, for a set of goals, in the specified conditions. The quality characteristics of a software product are described by a set of standardized properties described by the International Standards Organization (ISO). For example: the functionality, the reliability, usability and the others attributes of ISO 9126 on which the users are concerned.

The software complexity is highly connected with its quality for a simple reason. After the initial developing phase of a piece of code, one usually invests a lot in the maintenance and permanent updating of the respective software. In order to ensure the quality of a software program, one needs to have a good capacity to maintain and better organize the code sources. A program with an advanced complexity will always need big investments to permanently guarantee services in conformity with the client's demands.

References

1. Abran, A., Al-Qutaish, R.E., Desharnais, J. M., Habra, N., **An Information Model for Software Quality Measurement with ISO Standards,** In: **„SWEDC-REK, International Conference on Software Development"** , Reykjavik, Island , University of Iceland, 2005, p. 104-116

2. Abu Talib, M., Abran, A., Ormandjieva, O., **COSMIC-FFP & Functional Complexity (FC) Measures: A Study of their Scales, Units and Scale Types,** In Proceedings of **„The 15th International Workshop on Software Measurement - IWSM'2005"**, Montreal, Canada, Shaker-Verlag , 2005, p. 209-225
3. Abu Talib, M., Ormandjieva, O., Abran, A., Buglione, L., **Scenario-Based Black-Box Testing in COSMIC-FFP,** In: **„Software Measurement European Forum - SMEF 2005"**, Rome, Italy , 2005, p. 173-182
4. Al-Qutaish, R.E., Abran, A., **An Analysis of the Design and Definitions of Halstead's Metrics,** In proceedings of „**The 15th International Workshop on Software Measurement - IWSM'2005"**, Montreal, Canada , Shaker-Verlag , 2005 , p. 337-352
5. Anton, A.I., and Potts, C., **Functional Paleontology: System Evolution as the User Sees It,** In: Proceedings of **„The 23rd International Conference on Software Engineering, ICSE01"**, Toronto, 12-19 May 2001, p. 421-430
6. Azuma, M., **SQuaRE: The next Generation of ISO/IEC 9126 and 14598, International Standards Series on Software Product Quality, in Proceedings of the European Software Control and Metrics Conference (ESCOM),** 2-4 April 2001, London, UK, p. 337-346
7. Boehm, B.W., Abts, C., Brown A.W., Chulani, S., Clark , B., Horowitz, E., Madachy, R., Reifer, D., and Steece, B., **Software Cost Estimation with COCOMO II,** Prentice Hall PTR, 2000
8. Bruegge B., Dutoit, A.H., **Object-Oriented Software Engineering - Using UML, Patterns, and Java,** Pearson Education Inc., Pearson Prentice Hall, 2004
9. Garmus, D., Herron, D., **Function Point Analysis - Measurement Practices for Successful Software Projects,** Addison-Wesley, 6th Printing, December 2004
10. Halstead, M.H., **Elements of Software Science, Operating, and Programming Systems Series**, Volume 7, New York, NY : Elsevier, 1977
11. Homer, S., and Selman, A. L., **Computability and Complexity Theory,** Springer Verlag, New York , 2001
12. ISO, **ISO/IEC 9126-1, Software Engineering - Product Quality - Part 1: Quality model,** Geneva , International Organization for Standardization, 2001
13. ISO, **ISO/IEC FCD 25000, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE,** Geneva, International Organization for Standardization, 2004
14. ISO, **ISO/IEC FCD 25020, Software and System Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Measurement Reference Model and Guide,** Geneva , International Organization for Standardization, January 24, 2005
15. ISO, **ISO/IEC IS 9126, Software Product Evaluation - Quality Characteristics and Guidelines for Their Use,** Geneva , International Organization for Standardization, 1991
16. ISO, **ISO/IEC PDTR 25021, Software and System Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Measurement Primitives,** Geneva , International Organization for Standardization, 2004
17. ISO, **ISO/IEC TR 9126-2, Software Engineering - Product Quality - Part 2: External Metrics,** Geneva , International Organization for Standardization, 2003
18. ISO, **ISO/IEC TR 9126-3, Software Engineering - Product Quality - Part 3: Internal Metrics,** Geneva , International Organization for Standardization, 2003
19. ISO, **ISO/IEC TR 9126-4, Software Engineering - Product Quality -Part 4: Quality in Use Metrics,** Geneva , International Organization for Standardization, 2004
20. Lopez Martin, M.-A., Habra, N., Abran, A., **A Structured Analysis of the McCabe Cyclomatic Complexity Measure,** In: Proceedings of the 14th International Workshop on Software Measurement (IWSM2004) Berlin, Germany, Shaker Verlag, 2004
21. McCabe, T., **A Complexity Measure,** In: IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976, p. 308-320

22. McCabe, T.J., and Watson, A.H., and McCabe and Associates, Inc., **Software Complexity,** December 1994, http://www.stsc.hill.af.mil/crosstalk/1994/12/xt94d12b.asp
23. SC7, **ISO/IEC FCD 25000, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE,** ISO/IEC JTC1/SC7 WG6, January 1, 2004
24. Suryn, W., Abran A., and April A., **ISO/IEC SQuaRE : The Second Generation of Standards for Software Product Quality,** In: **„The 7th IASTED International Conference on Software Engineering and Applications",** California , USA , 2003
25. Tran-Cao, D., Abran A., and Levesque, G., **Functional Complexity Measurement,** In: Proceedings of the **„International Workshop on Software Measurement (IWSM'01)"**, Montreal, Quebec, Canada , August 28-29, 2001, p. 173-181
26. Tran-Cao, D., Levesque, and G., Meunier, J.-G., **Software Functional Complexity Measurement with the Task Complexity Approach,** In: Proceedings of the International Conference RIVF'04, Hanoi, Vietnam , February 2-5, 2004, p. 77-85
27. Tran-Cao, D., Levesque, G., Abran, A., **From measurement of software functional size to measurement of complexity,** In: ICSM 2002, Montreal, Canada , 2002, p. 11-22
28. VanDoren, E., **Cyclomatic Complexity,** July 2000
29. VanDoren, E., **Maintainability Index Technique for Measuring Program Maintainability,** March 2004, http://www.sei.cmu.edu/str/descriptions/mitmpm.html

---

[1] Ion I. Bucur has graduated the Faculty of Automatic Control and Computers in 1975, and the faculty of Mathematics in 1982; he holds a PhD diploma in Computer Science and Engineering from 1999.
Currently he is lecturer within the Department of Computer Science and Engineering at Faculty of Automatic Control and Computers from the University "Politehnica" of Bucharest.
He published several books and over 20 journal articles in the field of computer engineering. His work focuses both on hardware and software applications.
He is currently teaching architectures of information systems, logic design and design testing, and project management of IT&C projects. He is member of IEEE, ACM and SRAIT.
He provides expert level support for ongoing design and verification projects with emphasis on the performance aspects of simulation and verification tools.
Provides expert level training and introduction on advanced topics such as:

- HDL simulators,
- Software Code Testing Estimation (COCOMO - II, Function Points, etc) and Software Project Management,
- Advanced computer-programming techniques,
- Tuned and optimized complex application,
- Programming languages, irrespective compilers and related tools: C, C++, Visual Basic, PL/SQL, VHDL, Verilog and Verilog PLI,
- Database design, optimization and implementation,
- Dynamic programming techniques,
- ISO 9001-3 Quality Assurance documents, protocols, methods,

Trained to manage projects with multiple teams.
He has participated in the organizing committee of recent International Conferences on Control Systems and Computer Science, and First Symposium on Technical Physics and Physical Engineering (TPPE 2005) chairman of the 8th (F) Session "Physics, Informatics and Computer Engineering", the 3rd International Colloquium "Mathematics in Engineering and Numerical Physics", MENP3F, 7-9 October 2004 Bucharest being member of the international committee and chairman of the 6th Section "Modeling in Engineering".