Military Technical College
Kobry El-Kobbah
Cairo, Egypt

10th International Conference
On Aerospace Sciences&
Aviation Technology

SOFTWARE SOURCE CODE: A QUALITY ASSURANCE MEASUREMENT SYSTEM

Dr. Ismail A. Taha* Dr. Kamel A. Elhadad*

ABSTRACT

Today software systems play a critical role in various aspects of human life, from rockets to health care, and become part of everyday life. Many of these systems are relied upon as being essential for the completion of day-to-day activities. The increased reliance on computer applications, and organizations that produce software puts more and more strain on software developers to produce high quality systems. For these reasons many international standards, requirements, and constrains were established to assure quality of software. This paper introduces a new software Source Code Quality Assurance Measurement System named "SCQAM". In addition, it presents some of the most important software quality assurance fundamentals used during the different phases of software development life cycle. Particularly, the focus of this paper is bounded to the coding phase, where in this phase the cure of software system will be established. Therefore, the scope of this paper covers most of the related aspects of software quality assurance of the coding phase including: software metrics, software quality factors, and software quality models like McCall's model, Boehm's model, ISO 9126 model, and SATC NASA model. As a result of analyzing these models, the proposed "SCQAM" system was designed, developed, and tested. The proposed SCQAM can measure over 30-source code metrics, then group these metrics to compute nine distinct quality factors and indicators, then an overall quality indicator of the input source code is calculated. The experimental results show the superiority of the SCQAM system over Project Analyzer; another quality assurance measurement system, specifically in the area of source code quality measurement.

Key Words: **Source Code Quality Assurance Measurement Systems, Quality Assurance, Software Quality Assurance, and Software Engineering.**

* Egyptian Armed Forces.

1. INTRODUCTION

Producing high quality software became a condition for software companies and developers to stay in the market. This enforces them to think about quality improvement activities, and quality assurance systems. This is probably the reason why so many process improvement, experiments and measurement systems are initiated, but few of them are successful. The basic difficulties in this paradigm are the vast number of factors included in producing a high quality software product. These factors include understanding the relationships among basic elements of the software to be produced. For example, the procedures of producing the product, the resources involved in software production, the selection of relevant quality attributes in each case, the metrics to be applied for measuring the selected quality attributes, and the usage of the measurements' results in order to improve software quality are all interrelated factors [8,9]. There is variety of standards, models, best practices that should be enforced to make high quality software products [3,4,5,10,19,29]. In fact, all of them are connected to software quality assurance, but there is no unified view or model to tell software producers/developers, how to produce an efficient software high quality system and how to evaluate the quality of the produced source code [12].

The scope of this paper is to survey, understand and analyze the existing approaches to software quality assurance and figure out the relationships among the different approaches. Then and based on this analysis, a candidate solution (the proposed "SCQAM" system) that would help a software developer to deal with quality assurance for program source codes is introduced and tested.

2. SOFTWARE QUALITY ASSURANCE MODELS

2.1 Product Quality Assurance Models

The elements defining software product quality assurance and the relationships between these elements have been summarized for the first time in two software quality models developed in the USA. One of the models was developed in 1977 by a team of researchers, lead by Barry W. Boehm [6]. The development of the other model is connected to the work done in 1978 by James A. McCall [1,7]. The two quality models focus on the software final product, and identify key attributes of the product, called quality factors. The quality factors are high-level quality attributes, like *reliability*, *usability*, and *maintainability*. Both models assume that the quality attributes are still on a too high level to be meaningful or to be measurable. Therefore, further decomposition is needed. The decomposition of the quality attributes is then called quality criteria. In a third level of decomposition the quality criteria are associated with a set of directly measurable attributes called quality metrics. These models are briefly sketched in Fig. 1.

The Boehm and McCall models are typical of fixed quality models [21]: we assume that all important quality factors needed are a subset of those published in these two models. To control and measure each attribute, we accept the models associated criteria and metrics, and, most importantly, the proposed relationships among factors, criteria and metrics.

Later, a new model named ISO 9126 was introduced [1,16,17]. This model is a derivation of McCall's model. It defines software quality as *"The totality of features and characteristic of a software product that bare on its ability to satisfy stated or implied needs."* The standard claims that the quality is composed of 6 factors: *functionality*, *reliability*, *efficiency*, *usability*, *maintainability*, *portability*, and that one or more of them are enough to describe any component of software quality [1,14]. The deficiency of this model is that it does not provide proper definition of the lower-level details and metrics needed to attain a quantitative assessment of product quality. This lack of specifics in these models offers little guidance to software developers who need to build quality products. However, ISO 9126 is considered a software product evaluation standard. It identifies six Software Quality Characteristics [17]:

1. ***Functionality:*** which covers the functions that a software product provides to satisfy user needs.

2. ***Reliability:*** which relates to capability of software to maintain its level of performance.

3. ***Usability:*** which relates to the effort needed to use software.

4. ***Efficiency:*** which relates to the physical resources used when the software is executed.

5. ***Maintainability:*** which relates to the effort needed to the make changes to the software.

6. ***Portability:*** which relates to the ability of software to be transferred to a different environment.

ISO 9126 suggests sub-characteristics for each of the primary characteristics. They are useful as they clarify what is meant by the main characteristics.

2.2 Process Quality Assurance Models

This part presents elements of another possible way of approaching software quality assurance, called the process quality assurance approach.

A well-known framework for process assessment is the Capability Maturity Model of SEI [1] and Bootstrap [24].

The Software Capability Maturity Model (CMM) developed at the Software Engineering Institute (SEI) of Carnegie - Mellon University [13,33]. The Capability Maturity Model describes software process management maturity relative to five levels [22,23], as depicted in Fig. 2. The Bootstrap methodology is an extension of the CMM, developed by a European Community ESPRIT project, between September 1991 and February 1993 [31].

3. THE PROPOSED SYSTEM

The objective of the proposed software Source Code Quality Assurance Measurement System **SCQAM** is to measure the quality of application's source codes even if there is no other information available but the source code. The proposed system measures the quality of a given software source code through three consecutive layers of calculations:

1. The first layer: where the source code quality metrics are calculated.
2. The second layer: where some of the software quality factors are calculated as a weighted sum of the obtained source code quality metrics obtained in the first layer of the system.
3. The third layer: where an overall quality indicator for the given source code is calculated as a weighted sum of the obtained quality factors obtained in the second layer.

Fig. 3 depicts the three layers of quality assurance measurements of the proposed SCQAM and their interrelationships.

Typically, software quality is measured using a weighted sum of criteria measurements [21,25,30]. In the proposed SCQAM system, a set of standard formulas is used, in the metrics calculation layer, to calculate each quality metric C_i. In the metrics calculation layer, the proposed system measures 23 source code quality metrics as shown in Fig. 4. Then each metric is scaled/normalized ($0 \leq scale \geq 1$). This normalization is done to avoid any over-effect of some metrics over the others.

In the factors calculation layer, where 9 factors are measured [2], interrelated set of k-measured quality metrics are used to calculate the source code quality factors affected by these metrics using Equation 1. The effect of each quality metric C_i on the measured quality factor QF_j is represented by a weight value $W_i = <0,1>$. The values of $\{W_i: i:1..23\}$ are predetermined and assigned along with each source code quality metrics measured by the SCQAM system. The system provides its knowledgeable users with the capability of adapting the default values of $\{W_i: i:1..23\}$ to indicate the importance of the metrics with respect to the application nature of the source code in hand. For example, if the

quality assurance measurement system introduced in this paper allows Visual Basic developers to evaluate their source code quality before the implementation phase.

4. EXPERIMENTAL RESULTS

The proposed SCQAM system was implemented to test applications written in Visual Basic Programming language. This section presents the experimental results of implementing the SCQAM system to measure a set of Visual Basic source code applications. The SCQAM system was used to measure the quality metrics, factors, and overall quality indicator of two different versions of a Visual Basic program, written by two persons, to solve a simple application problem. The reason of making the application problem simple was to validate the correctness of the measured quality measures and indicator of the SCQAM. This simple application, which two persons were asked to program using VB has an identical user interface compliance to our requirements to the two programmers. This application purpose is to generate 10 integer numbers, and store them in an array. These 10 numbers should then be sorted and stored in another array. The programs should also extract the minimum and maximum numbers then calculate the average of these 10 numbers.
Table (1), Fig. 5 and 6 show all source code metrics for each procedure used in both examples, while Fig. 7, shows a set of SCQAM system snapshots. Other experiments to test the proposed system were conducted including measuring the source code of the proposed system itself [2].

5. CONCLUSION AND FUTURE WORK

This paper introduced a new software source code quality assurance measurement system named SCQAM. The SCQAM is based on some of the previously developed industry standards and models like Boehm, McCall, ISO 9126, CMM, and SATC NASA models for software quality assurance models. The introduced SCQAM has three layers of calculations, the metrics, the factors, and a unique overall software quality indicator calculation layers. It can measure up to 31 software quality metrics; only 23 of them are presented in this paper, and 9 software quality factors. Where each software quality factor is calculated as a weighted sum of the set of measured metrics; after normalizing them, that affects it. Similarly, the overall indicator is calculated as a weighted sum of the measured software quality factors after normalized them. These weights assigned to the software quality metrics and the factors are adaptive and can be set according to the application nature.
Based on the research done and briefly presented in this paper we can say "Introducing software quality assurance of the whole software development life cycle cannot be done at once [27,28]. It *takes time* and it has to be done *step by step* according to the phases of the life cycle of the developed system [26].
The implementation of the proposed system can easily expanded to measure the quality of other programming languages; like C++, not just the VB language.

measured source code solves tasks related to military application, then it will use a set of weights that is most probably different when it is used in a business application.

In addition, the proposed system provides its users with the capability of assigning a minimum and maximum target values for each metric and factor. These minimum and maximum vales are considers the boundaries of the acceptable limits for each metric and factor.

Similarly, in the overall quality indicator calculation layer, a weighted sum of the calculated software source code normalized factors are computed to provide an overall indicator of the quality of the software source code in hand using Equation 2.

$$QF_j = \frac{\sum_{i=1}^{k} C_i^* * W_{ij}}{\sum_{i=1}^{k} W_{ij}} \quad [1] \qquad Overall\ Quality\ Indicator = \frac{\sum_{j=1}^{m} QF_j^* * W_j}{\sum_{j=1}^{m} W_j} \quad [2]$$

In equation 1, QF_j is the $j^{th.}$ quality factor, W_{ij} is the effective weight of the normalized $i^{th.}$ source code metric (C_i^*) on the $j^{th.}$ quality factor QF_j, and k is the number of metrics affecting this QF_j. In equation 2, QF_j^* is the normalized $j^{th.}$ Quality factor QF_j, W_j is the effective weight of the normalized $j^{th.}$ Source code factor (QF_j^*) on the overall quality indicator of the software source code in hand, and m is the number of factors affecting this source code.

Since the maintainability factor is affected by other metrics than those available from the source code, then in the proposed system it was replaced by another term called "maintainability index (*MI*)" and calculated by Equation 3 [21].

$$MI = 171 - 3.42 * \ln(aveE) - 0.23 * CC - 16.2 * \ln(aveLOC) + 50 * \sin(\sqrt{2.4 * perCM}) \quad [3]$$

Where *aveE* is the average of "*Halstead effort*" per module, *CC* is the average of "*cyclomatic complexity*" per module, *aveLOC* is the average *"lines of code"* per module, and *perCM* is the average "*percentage of lines of comments*" per module. Generally, software quality measurements may be fundamental or derived, i.e., measured directly or derived by combining two or more measurements. Halstead Software Science measurements have been discredited on both empirical and theoretical grounds. However, it should be noted that the use of delivered source instructions and number of unique operands might be useful measurements. Cyclomatic complexity can be a useful measurement in the planning and assessment of testing. Outside this application area, its usefulness may be limited because of its close relationship with LOC [11,15]. Quality by itself is a vague concept and practical quality requirements have to be carefully defined. Most of the qualities that are apparent to the users of software can only be tested only when the system is completed [20,32,34].
Increasing in-line comments will increase readability. Comment lines and average value of cyclomatic complexity affect the maintainability factor. The software source code

SCQAM was able to measure a very important reliability indicator even before installing the software. Comparing SCQAM system with Project Analyzer [18]; another software quality measurement system, it was found that SCQAM preponderates Project Analyzer, by computing source code overall quality indicator and more quality factors.

REFERENCES

[1] ATLAS Quality Assurance Group, 2002, http://www.atlas.web.cern.ch.

[2] Ayman H. Odeh Taha, " Software quality assurance: Design and implementation of software source code quality assurance measurement system", M.Sc. thesis, Military Technical Collage, Cairo, Egypt, 2002.

[3] Bob Hughes, "Practical software measurement", McGraw-Hill Companies, 2000.

[4] Fenton N., "Software Metrics - A Rigorous Approach", Chapman & Hall, London, 1991.

[5] Fenton N. and M., "Software Metrics and Risk", European Software Measurement Conference, 1999.

[6] Grant Rule P., "The Importance of the size of software requirements", NASSCOM Conference 2001,p.18.

[7] Humphrey, Watts S., "Managing the software process", the SEI Series in Software Engineering, 1990.

[8] IEEE Standard for Application and Management of the Systems Engineering Process, 8 December 1998, available at http://webstore.ansi.org/ansidocstore.

[9] Internet: (http://hissa.nist.gov), Quality Characteristics and Metrics.

[10] Internet: (http:// www.cs.washington.edu), A Model-based Approach to Object-Oriented Software Metrics.

[11] Internet:(http://sern.ucalgary.cacoursescpsc451F00Complexity.html), Complexity - Software Metrics

[12] Internet: (http://cse.dcu.ie/essiscope/sm2/charact.html), Quality Characteristics.

[13] Internet: (http://www.telecom.lth.se), A case study on GUI enhancement through framework composition.

[14] Internet: (http://irb.cs.tu-berlin.de/~zuse/), Technical Metrics for Software.

[15] Internet: (http://louisa.levels.unisa.edu), Software Quality Metrics.

[16] Internet: (http://satc.gsfc.nasa.gov), A Software Quality Model & Metrics.

[17] Internet: (http://satc.gsfc.nasa.gov), Standards for non-OO languages.

[18] Internet: (http://www.aivosto.com/vb.html), Project Analyzer.

[19] Internet: (http://www.axeljosefson.com), Quality Assurance Guideline.

[20] Internet: (http://www.enel.ucalgary.ca), Measuring External Product Attributes.

[21] Internet: (http://www.indus.uah.edu/phd/chapter2/ChapterII.html), Quality Models and Object-Oriented Metrics.

[22] Internet: (http://www.stc-online.org), Software quality assurance a technical report, 2000.
[23] Internet: (http://www.whatistesting.com/sanalysisa.htm), Panos Ntourntoufis, "From Software Quality Control to Quality Assurance".
[24] Internet: (http://irmc.state.nc.us), Guidelines for Agency Software Metrics Planning.
[25] Internet:(http://sern.ucalgary.ca/~philip/), Software Quality Assurance Proposal.
[26] James F. Peters, Witold Pedrycz; "Software Engineering an engineering approach", John Wiley & Sons, Inc., 2000.
[27] Johanna Rothman, "Using Quality to Drive Project Lifecycles", 1999.
[28] Katalin Balla, "The complex quality world: developing quality management systems for software companies", Eindhoven University, 2001.
[29] Linda H. Rosenberg, "What is Software Quality Assurance? ", STC conference 2002.
[30] Linda H. Rosenberg, " The Science of Software Quality Assurance", ASQ Conference 2001.
[31] Majida Sharif, Sarah Salahuddin; "Capability Maturity Model-Implementation and Implications", FAST-NU Conference, July 2000.
[32] Ropponen J. and Lyytinen K., "Components of Software Development Risk", IEEE Trans. Software Eng. Vol.26, No 2, p.98-110, February 2000.
[33] Rushby Craig, "Software Quality Assurance in a CMM Level 5 Organization", the Journal of Defense Software Engineering, May 1999.
[34] Thomas Liedtke, Peter Paetzold, "Three Numbers to Measure Project Performance", International Conference on Applications of Software Measurement ASM 2001.

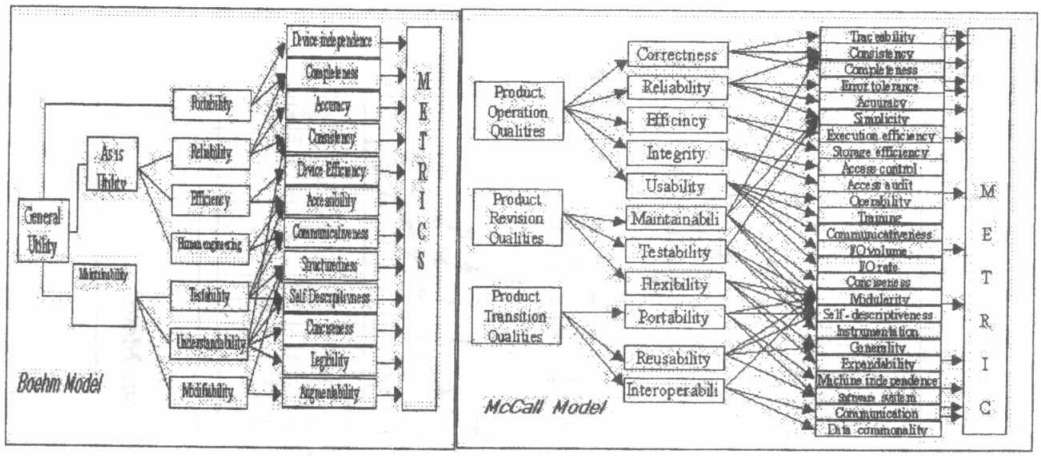


Figure (1): Boehm and McCall Software Quality Assurance Models

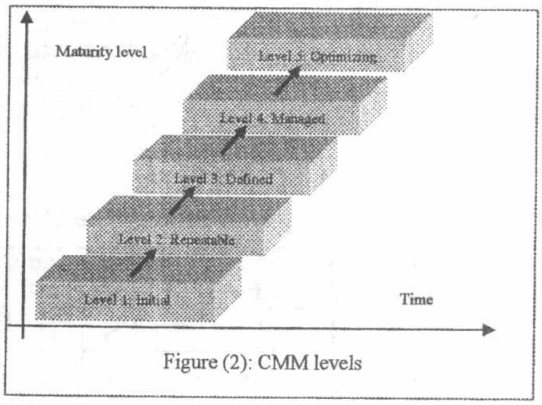


Figure (2): CMM levels

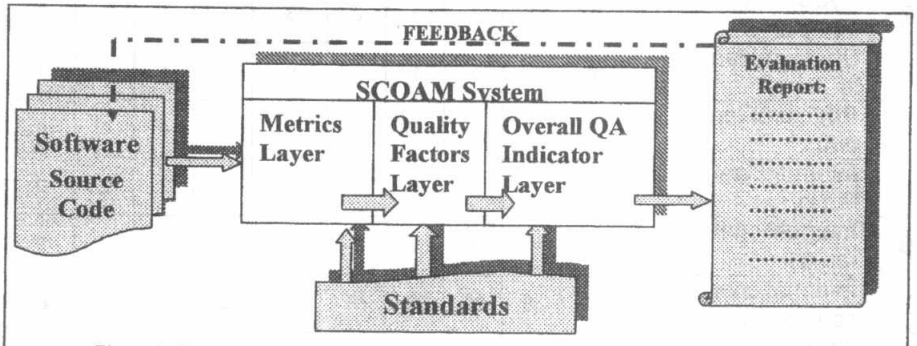


Figure 3: The Proposed SCQAM System Components and its three-calculation layers

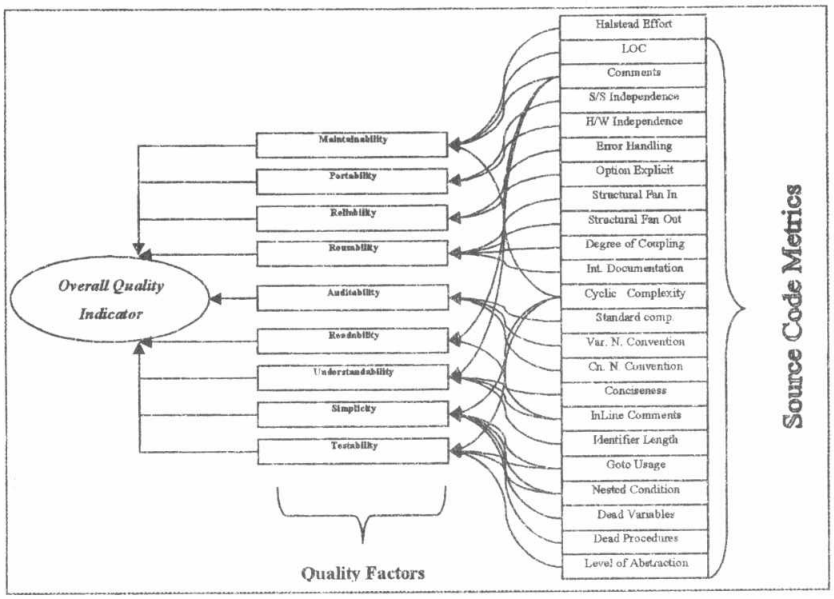


Figure 4: SCQAM System

Table 1: Metrics of all procedures of the VB Source code of the same Application

No	Metric	Measured Values of Program #1 Procedures						Measured Values of Program #2 Procedures					
		P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{21}	P_{22}	P_{23}	P_{24}	P_{25}	P_{26}
1	Dead Variables	No	No	No	No	No	No	No	No	No	No	No	No
2	Cyclomatic complexity	3	1	4	3	3	2	3	1	4	3	3	2
3	Structural fan in	0	0	1	1	1	1	0	0	1	1	1	1
4	Structural fan out	4	0	0	0	0	0	4	0	4	0	0	0
5	Informational fan in	5	0	3	2	2	2	6	0	3	2	2	2
6	Informational fan out	0	0	1	1	1	1	1	0	3	2	2	2
7	Informational complexity	0	0	36	20	20	18	108	0	117	44	40	36
8	Nested conditions	1	0	3	2	2	1	1	0	3	2	2	1

9	Nested loops	1	0	2	1	1	1	1	0	2	1	1	1
10	Total lines	16	9	14	10	10	9	37	22	14	20	20	19
11	LOC	15	0	12	10	10	9	18	12	12	11	10	9
12	Comments line	1	0	2	0	0	0	19	9	9	9	9	9
13	Space lines	0	0	0	0	0	0	0	1	1	1	1	1
14	Local variables	0	0	1	1	1	1	0	0	1	1	1	1
15	Operators	9	7	6	5	5	7	9	7	6	5	5	6
16	Unique operators	1	4	2	2	2	3	1	4	2	2	2	3
17	Operands	19	15	14	12	12	14	19	15	14	12	12	12
18	Unique operands	14	11	7	7	7	9	14	11	7	7	7	8
19	Procedure Vocabulary	15	15	9	9	9	12	15	15	9	9	9	8
20	Procedure Length	28	22	20	17	17	21	28	22	20	17	17	18
21	Procedure Volume	109.3	85.9	63.3	53.8	53.8	75.2	109.3	85.9	63.3	53.8	53.88	62.2
22	Level of abstraction	0.85	0.36	0.5	0.58	0.58	0.42	0.92	0.36	0.5	0.58	0.583	0.44
23	Effort	74	234	126	92	92	175	74	234	126	92	92	140
24	Time (sec)	4	13	7	5	5	9	4	13	7	5	5	7
25	Goto Usage	0	0	0	0	0	0	0	0	0	0	0	0
26	Error Handling usage	No	No	No	No	No	No	Yes	Yes	Yes	Yes	No	Yes
27	InLine Comments	4	0	1	0	0	0	3	0	1	1	0	2
28	Complexity/size	0.2	0.11	0.33	0.3	0.3	0.22	0.16	0.08	0.33	0.27	0.3	0.22
29	Internal Documentation	0.062	0	0.14	0	0	0	0.51	0.42	0.14	0.45	0.473	0.5
30	Estimated length	53.3	46.0	21.6	21.6	21.6	33.2	53.3	46.0	21.6	21.6	21.65	28.7
31	Impurity	1.90	2.09	1.08	1.27	1.27	1.58	2.33	2.09	1.08	1.27	1.27	1.59

where P_{ij} is the procedure number j in the i^{th}. Program.

The quality factors chart for program 1 has the following figure

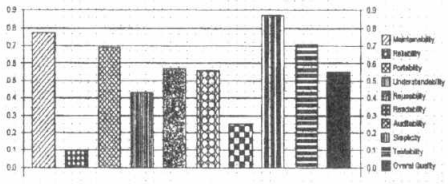


Figure 5: Quality factor chart for example1 with Overall quality indicator =0.550

The quality factors chart for program 2 has the following figure

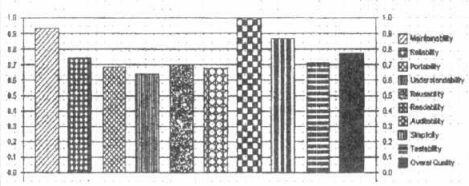
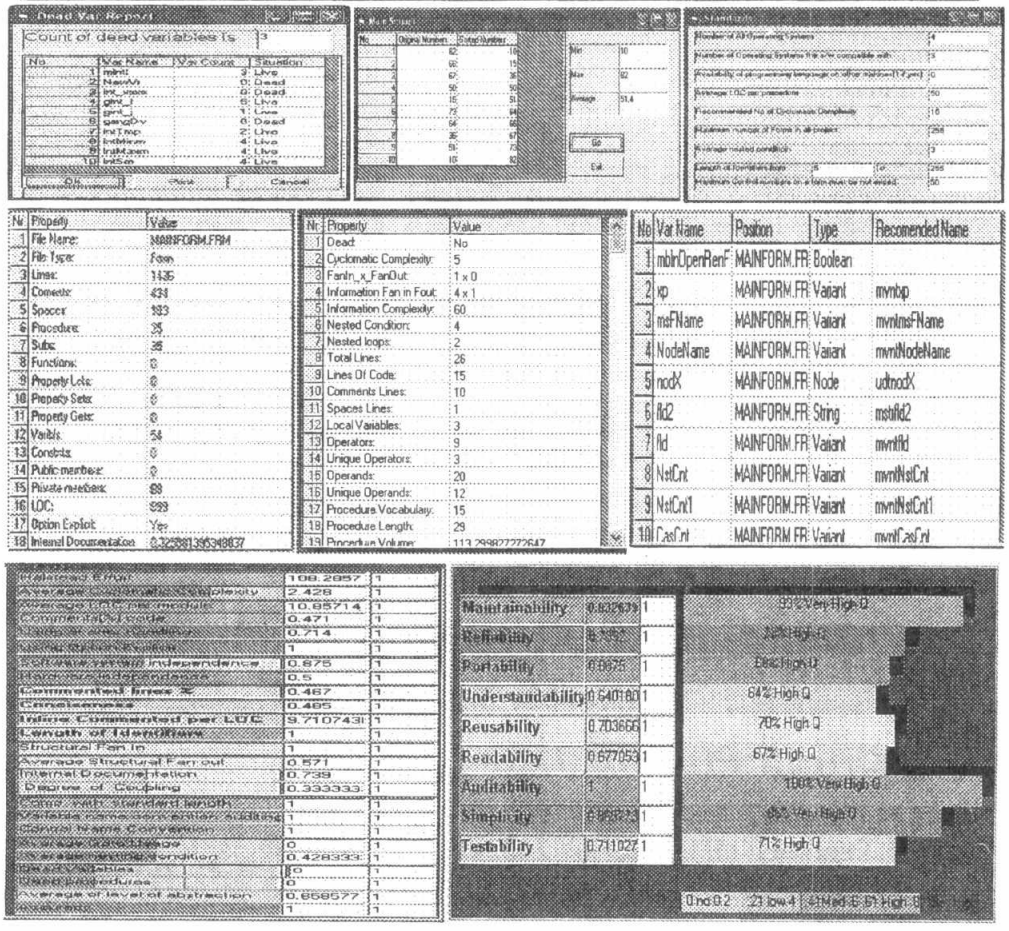


Figure 6: Quality factor chart for example2 with Overall quality indicator = 0.773



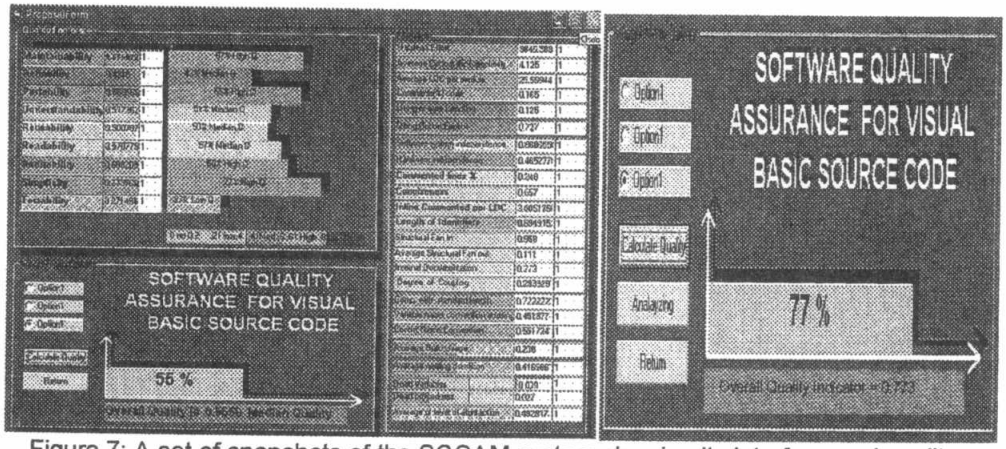


Figure 7: A set of snapshots of the SCQAM system showing its interface and quality assurance reports and their contents