Software Quality Estimation through Analytic Hierarchy Process Approach

B. B. Jayasingh, Professor, IT Dept., CVR College of Engineering, Ibrahimpatan, RR Dist-501510.
Email: bbjayasingh9@rediffmail.com
B. Rama Mohan, Associate Professor, CSE Dept., JNTUH College of Engineering, Hyderabad – 500085, (AP).
Email: b.ramamohan@jntuh.ac.in

Abstract--**Software quality assurance plays an important role to justify the software to reach the right level of quality. Its objective is to estimate the software quality and the errors in software modules before release to market place. In order to estimate the right level of quality one must apply some comprehensive techniques. To determine software quality, we present the Analytic Hierarchy Process (AHP) is the suitable approach for assessing the quality of software, with judgments by a group of experts rating. Since developing perfect or highly compatible software is not easy the AHP approach puts a threshold beyond which the quality of the new development is more than acceptable. We studied various techniques approached by different authors of software quality assurance to enable the stakeholder for choosing right kind of techniques suitable to their project.**

Index Terms—**Analytic Hierarchy Process, In Vivo Testing, Source Code Metrics, Software Quality, Divide And Conquer, Prioritization Problem.**

I. INTRODUCTION

Software runs on the machine and machine finds the errors with human interaction while development process. The development phase of SDLC (Software Development Life Cycle) includes more formal and detailed technical reviews that ensure to detect errors early. It is the time now for face-to-face communication instead of formal reviews, so the teams to decide and own the quality of each product. The product quality will be higher through the agile [1] development process that many organizations belief.

One specific form of technical debt [2] that has been studied for some time is design debt, also referred to as architecture debt. Design debt occurs whenever the software design no longer fits its intended purpose. A software project can run into design debt for various reasons. For example, adding a series of features that the initial architecture was not intended to support can cause debt and decrease the maintainability of software. Or, drifting away from a proposed architecture can bring short-term payoffs but might have consequences for the portability and interoperability of software. Reducing or eliminating design debt means in most cases that the design should be tailored and adapted towards changing requirements immediately and continuously.

We study various techniques approached by different authors of software quality assurance to enable the stakeholder for choosing right kind of techniques suitable to their project. The most important goals of the software industry is to develop high-quality and reliable software for their customers. We also consider the in vivo testing, in which tests are continuously executed in the deployment environment. However, the technical quality of source code (how well written it is) is an important determinant for software maintainability. Our survey focuses the low level source code metrics also effect to the high level quality characteristics. One must reconsider the divide and conquer principle applied consistently throughout the development (requirements documentation, design, review, coding, inspection, and testing) and maintenance of the product. However, predicting the exact number of faults is too risky, especially in the beginning of a software project when too little information is available. We conclude the analytical hierarchy process is the best suitable approach for software quality assurance.

In section II we discuss about the in vivo testing that run in the deployment environment which is hidden to the user. In section III we discuss the technical quality of source code (how well written it is) and how it affects to software maintainability. In section III we discuss the source code metrics that affect to the high level quality characteristics. In section IV we discuss the mathematical model for software quality assurance called divide and conquer where the complex job of building a software product must be reduced to a set of much simpler jobs. In section V we discuss the analytic hierarchy process consists of 6 criteria and 27 subcriteria where the prioritization problem solved to estimate the quality of software.

DOI: 10.32377/cvrjst0416

II. IN VIVO TESTING APPROACH

Testing at deployment environment needs attention of the developer where it runs continuously without the knowledge of user [3]. It is an improved version of unit or integration tests and focusing on aspects of the program that should hold true regardless of the state of the system. These tests keep the current state of the program while execution without affecting or altering the state that visible to users. This testing can be used to detect concurrency, security, or robustness issues.

In vivo testing [3] is a methodology by which tests are executed continuously without disturbing the state of the system that is hidden to the user. A prototype is developed in JAVA programming language called Invite (IN VIvo TEsting framework), focused on distributed execution of tests in earlier version. The current version includes a more detailed description in which it reveals the defects in real world applications.

A. In Vivo Testing Framework

The in vivo testing framework describes the steps for a software vendor regarding to use the Invite framework. The development of any new test code and the configuration of the framework must be done prior to distribute an in vivo-testable system.

1. Create test code
   The test methods must reside in the same class as the code they are testing (or in a superclass).
2. Instrument classes
   The vendor must select the methods from one or more Java classes in the application that under test for instrumentation.
3. Configure frameworks
   Each method runs with probability ρ, the vendor must configure Invite with values representing Before deployment.
4. Deploy applications
   The compiled code including the tests and the configured testing framework would ship as part of the software distribution. However, the customer would not even notice that the in vivo tests were running.

B. Testing Scenario

- The user is not aware of the presence of the testing, so his performance is not affected.
- These tests perform the following:
  - Checks the values of the individual variables
  - How the variables are related
  - Condition is held or not? – after some execution of the software

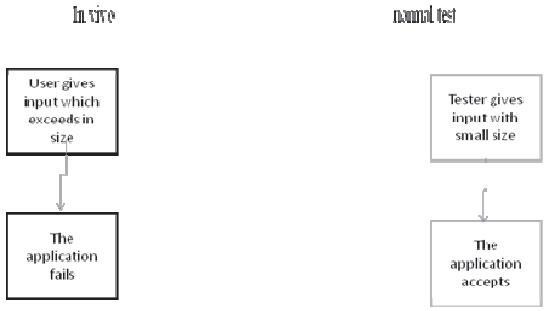

Figure 1 Testing Scenario

C. Concurrency Problem

- Function1 is used to destroy the processes that are running.
- Function 2 is used to create a new process.
- If function1 and function2 are called at the same time then function2 cannot create new process.
- This kind of defect can only be detected during run time, so In vivo testing approach is useful.

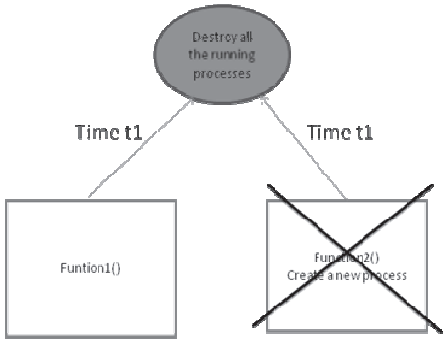

Figure 2 Concurrency problem

III. CODE QUALITY BENCHMARKING

The quality of source code also affect to software maintainability [4]. Whenever a change is required it must ensure how easy it is to perform the change, to implement the change, to avoid unexpected effects of that change and to validate the change.

Software Code Metrics

Software Improvement Group (SIG) chose 6 source code properties as key metrics for the quality assessments, namely:

1. Volume the larger the size the more to maintain since there is more information.
2. Redundancy duplicated code has to be maintained in all places where it requires.

3. Unit size the lowest-level piece of code and should be small to understand easily.
4. Complexity testing simple systems is easier than complex ones.
5. Unit interface size units with many interfaces to other units can be a symptom of bad encapsulation.
6. Coupling tightly coupled components are more resistant to change.

A. *Measurements Aggregation*

The sub characteristics are made quantifiable with the above source code metrics [5]. However, the metrics values are aggregated to a grand total of the whole system. Latter summarize to the cyclomatic complexity based on a set of thresholds. The aggregated measurements are used to determine a rating for each source code property, based on the application of another set of thresholds. These are further combined to calculate ratings for the sub characteristics and the general maintainability score for a given system [6].

B. *Standardized Evaluation Procedure*

The procedure consists of several steps, defined by SIG [7] quality model starting with the take-in of the source code by secure transmission to the evaluation laboratory and ending with the delivery of an evaluation report.

1. Intake: The source code is uploaded to a secure and standard location. For future identification of the original source a checksum is calculated.
2. Scope: it defines an unambiguous description of which software artifacts are to be covered by the evaluation. The description includes the identification (name, version, etc.) of the software system, a characterization (programming languages and the number of files analyzed) as well as a description of specific files excluded from the scope of the evaluation and why.
3. Measure: Apply an appropriate algorithm to determine the software artifacts automatically. The values of source code units are then aggregated to the level of properties of the system as a whole.
4. Rate: The values obtained in the measure step are combined and compared against target values to determine quality sub ratings and the final rating for the system.

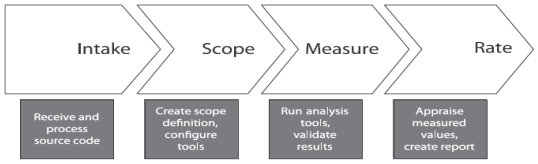

Figure 3 Evaluation framework

IV. SOURCE CODE METRICS AND MAINTAINABILITY

The ISO/IEC 9126 standard defines six high level product quality characteristics that are widely accepted both by industrial experts and academic researchers. These characteristics are: *Functionality*, *Reliability*, *Usability*, *Efficiency*, *Maintainability* and *Portability*. The characteristics are affected by low level quality properties [8], that can be *internal* (measured by looking inside the product, e.g. by analyzing the source code) or *external* (measured by executing the product, e.g. by performing testing). This work focuses on the relationship between the low level source code metrics and the high level quality characteristics defined by the standard.

Many researches propose maintainability models based on source code metrics. Bakota et al. [9] suggest a probabilistic approach for computing *maintainability* for a system. Heitlager et al. [10] also introduce a maintainability model. They transform metric value averages to the [9] discrete scale and perform an aggregation to get a measure for *maintainability*. Bansiya and Davis [11] developed a hierarchical model (QMOOD) for assessment of high level design quality attributes and validated it on two large commercial framework systems.

This work performed a manual evaluation of 570 class methods from five different aspects of quality. Now it is developed as a web-based framework to collect, store, and organize the evaluation results.

A. *Evaluated Systems*

An evaluated system JEdit is discussed, a well-known text editor designed for programmers. JEdit is a powerful tool written in Java includes syntax highlight, built-in macros, plug-in support, etc. The system selects 320 out of 700 methods to evaluate. The other evaluated system selects 250 out of 20,000 methods to evaluate. The evaluation was performed by 35 experts, who varied in age and programming experience.

DOI: 10.32377/cvrjst0416

B. The Evaluation Framework

The system consists of four modules:

1. AnalyzeManager - the module computes low level source code metrics and other analysis results.
2. Uploader - the module uploads the source code artifacts into a database.
3. AdminPages - the module manages the web interface and control the analysis process.
4. EvalPages - the module allow the users to evaluate the methods.

The questions are organized into the following five categories:

1. Analyzability - how easy the code to diagnose or to make a change.
2. Changeability - how easy the code to make a change in the system (includes designing, coding and documenting changes).
3. Stability - how easy the code to avoid unexpected effects after a change.
4. Testability - how easy the code to validate the software after a change.
5. Comprehension - how easy the code to comprehend (understanding its algorithm).

The author have shown the evaluator panel in fig. 4 for our better understanding.

Evaluator Panel

```
classExample{
public static void main(string args[])
{
int num=Integer.parseInt(args[0]);
int temp=num,res=0;
while(temp>0){
res=res+temp;temp--;}
```

Question: How easy it is to diagnoise the system for deficiencies or to identify where to make changes?

- o Good
- o Average
- o Poor

Figure. 4 Evaluator Screen

V. MATHEMATICAL MODELS

Present day software is more complicated where a small error can have bigger impact. With the complex task, the solution is the divide and Conquer strategy [12]. In this strategy the complex job can be reduced to a set of much simpler. This "Divide and Conquer" principle must be applied consistently throughout the development (requirements documentation, design, review, coding, inspection, and testing) and maintenance of the product

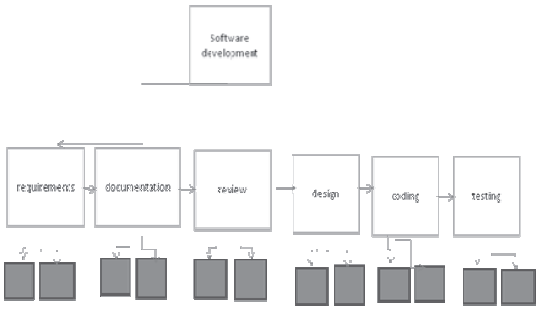

Figure 5 Divide and Conquer Model

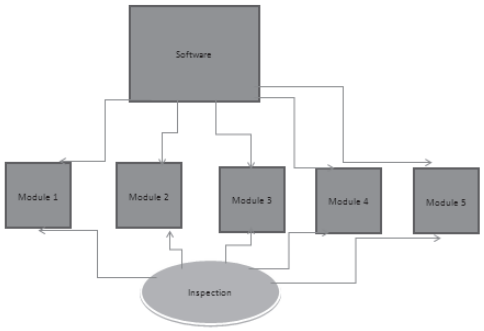

Figure 6 Inspection Using Divide and Conquer

It discusses one of three ways to measure software quality:

1. Reliability

It measures the failure rate of the software that depends on the way the software is used.

2. Correctness

It measures the correctness of a program because correctness is one property that high-quality software must have.

3. Trustworthiness

It is a question of whether users should trust a product. Some undesirable types of behavior called hazards have to be eliminated by the design and the process called hazard analysis. The more we test the more we trust in the product.

VI. ANALYTIC HIERARCHY PROCESS

The analytic hierarchy process (AHP) is the popular approach for assessing the quality of software, with judgments by a group of experts at different levels. It will explain the purpose and features of the system, what the system will do functional requirements and the constraints under which it must work. This document is

intended for the developer of software source code to understand their code quality. To determine software quality, quality metric models have been studied by many researchers. The AHP selects six criteria with 27 sub criteria in ISO/IEC 9126-1 (2001), which is the revision of 1991 version (ISO/IEC 9126, 1991) [13]. The relative rating provided by the expert contains nine scale value i.e. Equally important, Weakly important, Moderately important, Moderately plus, Strongly important, Strongly plus, Very strongly, Very very strongly and Extremely important. However the ratings provided by the experts are all fuzzy numbers. The proposed approach can help a group of various experts including developers, testers and purchasers, to measure the level of the software quality of the in-house development or the third party development.

A. Prioritization Problem

Consider a 3*3 group pair wise matrix for 3 criteria by 2 expert judgments as follows:

$$\begin{bmatrix} (1,1,1) & \begin{matrix}(3,4,5)\\(2,3,4)\end{matrix} & \begin{matrix}(5,6,7)\\(6,7,8)\end{matrix} \\ \begin{matrix}\left(\frac{1}{5},\frac{1}{4},\frac{1}{3}\right)\\\left(\frac{1}{4},\frac{1}{3},\frac{1}{2}\right)\end{matrix} & (1,1,1) & \begin{matrix}(3,4,5)\\(4,5,6)\end{matrix} \\ \begin{matrix}\left(\frac{1}{7},\frac{1}{6},\frac{1}{5}\right)\\\left(\frac{1}{6},\frac{1}{5},\frac{1}{4}\right)\end{matrix} & \begin{matrix}\left(\frac{1}{5},\frac{1}{4},\frac{1}{3}\right)\\\left(\frac{1}{6},\frac{1}{5},\frac{1}{4}\right)\end{matrix} & (1,1,1) \end{bmatrix}$$

Phase 1: expert1 matrix

Step 1: consider 3 criteria given by only 1st expert from the above matrix.

$$\begin{bmatrix} (1,1,1) & (3,4,5) & (5,6,7) \\ \left(\frac{1}{5},\frac{1}{4},\frac{1}{3}\right) & (1,1,1) & (3,4,5) \\ \left(\frac{1}{7},\frac{1}{6},\frac{1}{5}\right) & \left(\frac{1}{5},\frac{1}{4},\frac{1}{3}\right) & (1,1,1) \end{bmatrix}$$

Step 2: consider only the lower values from the above matrix (step 1).

$$\begin{bmatrix} 1 & 3 & 5 \\ \frac{1}{5} & 1 & 3 \\ \frac{1}{7} & \frac{1}{5} & 1 \end{bmatrix}$$

Step 3: consider the higher values of the above matrix(step 2) and calculate the Least Common Multiple (LCM).

$$[1 \quad 3 \quad 5]$$

LCM=15

So the factors are 1*15, 3*5, 5*3, consider the upper factors and add it i.e. 15+5+3=23.

Now calculate the reciprocal values

15/23= 0.652

5/23=0.217

3/23=0.130

Now the weighted matrix for lower values is

$$\begin{bmatrix} w_1^L \\ w_2^L \\ w_3^L \end{bmatrix} = \begin{bmatrix} 0.652 \\ 0.217 \\ 0.130 \end{bmatrix}$$

Step 4: calculate for medium values

Consider only the medium values from the above matrix (step 1).

$$\begin{bmatrix} 1 & 4 & 6 \\ \frac{1}{4} & 1 & 4 \\ \frac{1}{6} & \frac{1}{4} & 1 \end{bmatrix}$$

Step 5: consider the higher values of the above matrix(step 4) and calculate the Least Common Multiple (LCM).

$$[1 \quad 4 \quad 6]$$

LCM=12

So the factors are 1*12, 4*3, 6*2, consider the upper factors and add it i.e. 12+3+2=17.

Now calculate the reciprocal values

12/17= 0.705

3/17=0.176

2/17=0.117

Now the weighted matrix for medium values is

$$\begin{bmatrix} w_1^M \\ w_2^M \\ w_3^M \end{bmatrix} = \begin{bmatrix} 0.705 \\ 0.176 \\ 0.117 \end{bmatrix}$$

Step 6: calculate for upper values

Consider only the upper values from the above matrix (step 1).

$$\begin{bmatrix} 1 & 5 & 7 \\ \frac{1}{5} & 1 & 5 \\ \frac{1}{7} & \frac{1}{5} & 1 \end{bmatrix}$$

Step 7: consider the higher values of the above matrix (step 4) and calculate the Least Common Multiple (LCM).

$$[1 \quad 5 \quad 7]$$

LCM=35

So the factors are 1*35,5*7, 7*5, consider the upper factors and add it i.e. 35+7+5=47.

Now calculate the reciprocal values

35/47=0.744

7/47=0.148

5/47=0.106

Now the weighted matrix for upper values is

$$\begin{bmatrix} w_1^U \\ w_2^U \\ w_3^U \end{bmatrix} = \begin{bmatrix} 0.744 \\ 0.148 \\ 0.106 \end{bmatrix}$$

So the final weighted matrix for expert 1 is

$$\begin{bmatrix} w_1^L & w_1^M & w_1^U \\ w_2^L & w_2^M & w_2^U \\ w_3^L & w_3^M & w_3^U \end{bmatrix} = \begin{bmatrix} 0.652 & 0.705 & 0.744 \\ 0.217 & 0.176 & 0.148 \\ 0.130 & 0.117 & 0.106 \end{bmatrix}$$

Step 8: now divide the values with highest of upper values i.e. 0.744

DOI: 10.32377/cvrjst0416

$$\begin{bmatrix} w_1^L & w_1^M & w_1^U \\ w_2^L & w_2^M & w_2^U \\ w_3^L & w_3^M & w_3^U \end{bmatrix} = \begin{bmatrix} 0.876 & 0.947 & 1 \\ 0.291 & 0.236 & 0.198 \\ 0.174 & 0.157 & 0.142 \end{bmatrix}$$

Now multiply the column values of the two matrixes for the expert1

0.652*0.876 + 0.217*0.291 + 0.130*0.174= 0.657

0.705*0.947 +0.176*0.236 + 0.117*0.157= 0.726

0.744*1 + 0.148*0.198 + 0.106*0.142= 0.788

So the values for expert1 is

$$[0.657 \quad 0.726 \quad 0.788]$$

Phase 2: expert2 matrix

Step 1: consider 3 criteria given by only 2nd expert from the above matrix.

$$\begin{bmatrix} (1,1,1) & (2,3,4) & (6,7,8) \\ \left(\frac{1}{4},\frac{1}{3},\frac{1}{2}\right) & (1,1,1) & (4,5,6) \\ \left(\frac{1}{8},\frac{1}{7},\frac{1}{6}\right) & \left(\frac{1}{6},\frac{1}{5},\frac{1}{4}\right) & (1,1,1) \end{bmatrix}$$

Step 2: consider only the lower values from the above matrix (step 1).

$$\begin{bmatrix} 1 & 2 & 6 \\ \frac{1}{4} & 1 & 4 \\ \frac{1}{8} & \frac{1}{6} & 1 \end{bmatrix}$$

Step 3: consider the higher values of the above matrix(step 2) and calculate the Least Common Multiple (LCM).

$$[1 \quad 2 \quad 6]$$

LCM=12

So the factors are 1*12, 2*6, 6*2, consider the upper factors and add it i.e. 12+6+2=20.

Now calculate the reciprocal values

12/20= 0.6

6/20=0.3

2/20=0.1

Now the weighted matrix for lower values is

$$\begin{bmatrix} w_1^L \\ w_2^L \\ w_3^L \end{bmatrix} = \begin{bmatrix} 0.6 \\ 0.3 \\ 0.1 \end{bmatrix}$$

Step 4: calculate for medium values

Consider only the medium values from the above matrix (step 1).

$$\begin{bmatrix} 1 & 3 & 7 \\ \frac{1}{3} & 1 & 5 \\ \frac{1}{7} & \frac{1}{5} & 1 \end{bmatrix}$$

Step 5: consider the higher values of the above matrix (step 4) and calculate the Least Common Multiple (LCM).

$$[1 \quad 3 \quad 7]$$

LCM=21

So the factors are 1*21, 3*7, 7*3, consider the upper factors and add it i.e. 21+7+3=31.

Now calculate the reciprocal values

21/31= 0.677

7/31=0.225

3/31=0.096

Now the weighted matrix for medium values is

$$\begin{bmatrix} w_1^M \\ w_2^M \\ w_3^M \end{bmatrix} = \begin{bmatrix} 0.677 \\ 0.225 \\ 0.096 \end{bmatrix}$$

Step 6: calculate for upper values

Consider only the upper values from the above matrix (step 1).

$$\begin{bmatrix} 1 & 4 & 8 \\ \frac{1}{4} & 1 & 6 \\ \frac{1}{8} & \frac{1}{6} & 1 \end{bmatrix}$$

Step 7: consider the higher values of the above matrix (step 4) and calculate the Least Common Multiple (LCM).

$$[1 \quad 4 \quad 8]$$

LCM=8

So the factors are 1*8, 4*2, 8*1, consider the upper factors and add it i.e. 8+2+1=11.

Now calculate the reciprocal values

8/11=0.727

2/11=0.181

1/11=0.09

Now the weighted matrix for upper values is

$$\begin{bmatrix} w_1^U \\ w_2^U \\ w_3^U \end{bmatrix} = \begin{bmatrix} 0.727 \\ 0.181 \\ 0.09 \end{bmatrix}$$

So the final weighted matrix for expert 2 is

$$\begin{bmatrix} w_1^L & w_1^M & w_1^U \\ w_2^L & w_2^M & w_2^U \\ w_3^L & w_3^M & w_3^U \end{bmatrix} = \begin{bmatrix} 0.6 & 0.677 & 0.727 \\ 0.3 & 0.225 & 0.181 \\ 0.1 & 0.096 & 0.09 \end{bmatrix}$$

Step 8: now divide the values with highest of upper values i.e. 0.727

$$\begin{bmatrix} w_1^L & w_1^M & w_1^U \\ w_2^L & w_2^M & w_2^U \\ w_3^L & w_3^M & w_3^U \end{bmatrix} = \begin{bmatrix} 0.825 & 0.931 & 1 \\ 0.412 & 0.309 & 0.248 \\ 0.137 & 0.132 & 0.123 \end{bmatrix}$$

Now multiply the column values of the two matrixes for the expert2

0.6*0.825 + 0.3*0.412 + 0.1*0.137= 0.632

0.677*0.931 +0.225*0.309 + 0.096*0.132= 0.712

0.727*1 + 0.181*0.248 + 0.09*0.123= 0.782

So the values for expert2 is

$$[0.632 \quad 0.712 \quad 0.782]$$

Phase 3: Synthesis

The matrix values for expert1 is

$$[0.657 \quad 0.726 \quad 0.788]$$

And the matrix values for expert2 is

$$[0.632 \quad 0.712 \quad 0.782]$$

Now calculate the average of these two matrices

$$[0.644 \quad 0.719 \quad 0.785]$$

The final fuzzy value (0.644, 0.719, 0.785) is derived as the level of the quality. These results the quality of the new development is more than acceptable. Thus the SQA can give permission for a product of this quality to be used.

VII. CONCLUSION

We study various techniques approached by different authors of software quality assurance to enable the stakeholder for choosing right kind of techniques suitable to their project. We also consider the in vivo testing, in which tests are continuously executed in the deployment environment. However, the technical quality of source code (how well written it is) is an important determinant for software maintainability. Our survey focuses the low level source code metrics also effect to the high level quality characteristics. One must reconsider the divide and conquer principle applied consistently throughout the development (requirements documentation, design, review, coding, inspection, and testing) and maintenance of the product. We conclude the analytical hierarchy process is the best suitable approach for software quality assurance.

REFERENCES

[1] P. McBreen, "Quality Assurance and Testing in Agile Projects", McBreen.Consulting, 2003.

[2] N. Zazworka et al., Investigating the Impact of Design Debt on Software Quality, ACM 978-1-4503-0586-0/11/05, pp. 17-23.

[3] C. Murphy et al., Quality Assurance of Software Applications Using the In Vivo Testing Approach, International Conference on Software Testing Verification and Validation, IEEE DOI 10.1109/ICST.2009.18, 2009, pp. 111-120.

[4] R. Baggen et al., Standardized code quality benchmarking for improving software maintainability, Software Qual J, Springer Pub., DOI 10.1007/s11219-011-9144-9, 2012, pp. 287–307.

[5] I. Heitlager et al., A practical model for measuring maintainability, In proceedings of 6th international conference on the quality of information and communications technology (QUATIC 2007), IEEE Computer Society, pp. 30–39.

[6] J. Correia et al., A survey-based study of the mapping of system properties to iso/iec 9126 maintainability characteristics, In 25th IEEE international conference on software maintenance (ICSM 2009), September 20–26, 2009, pp. 61–70.

[7] J. P Correia, J. Visser, Certification of technical quality of software products, In an International workshop on foundations and techniques bringing together free/libre open source software and formal methods, FLOSS-FM 2008, pp. 35–51.

[8] P´eter Heged˝us, Tibor Bakota, L´aszl´o Ill´es, Gergely Lad´anyi, Rudolf Ferenc, and Tibor Gyim´othy, Source Code Metrics and Maintainability:A Case Study, ASEA/DRBC/EL 2011, Springer-Verlag Berlin Heidelberg 2011, CCIS 257, pp. 272–284.

[9] Bakota, T., Heged˝us, P., K¨ortv´elyesi, P., Rudolf, F., Gyim´othy, T., A Probabilistic Software Quality Model, In Proceedings of the 27th IEEE International Conference on Software Maintenance, ICSM 2011, IEEE Computer Society, Williamsburg 2011, pp. 368–377.

[10] I. Heitlager, T. Kuipers, J. Visser, A Practical Model for Measuring Maintainability, In Proceedings of the 6th International Conference on Quality of Information and Communications Technology, 2007, pp. 30–39.

[11] Bansiya, J., Davis, C., A Hierarchical Model for Object-Oriented Design Quality Assessment, IEEE Transactions on Software Engineering 28, 2002, pp. 4–17.

[12] David Lorge PARNAS, The use of mathematics in software quality assurance, Front. Comput. Sci., Higher Education Press and Springer-Verlag Berlin Heidelberg, DOI10.1007/s11704-012-2904-2, 2012, pp. 3–16.

[13] K. K. F. Yuen, C.W. Lau Henry, A fuzzy group analytical hierarchy process approach for software quality assurance management, Fuzzy logarithmic least squares method, Expert Systems with Applications 38, ELSEVIER Publication, 2011, pp. 10292–10302.

DOI: 10.32377/cvrjst0416