Evaluation of Understandability of Object-Oriented Design

Devpriya Soni

Department of Computer Science and Engineering
Jaypee Institute of Information Technology, Noida, sec- 128, India
devpriyasoni@gmail.com

Abstract. Quality of **s**oftware design directly affects the understandability of the software developed. As the size and complexity of the software increases it drastically affects quality attributes, especially understandability. The direct measurement of quality is difficult because there is no single model that can be applied in all situations. Models proposed by various researchers are not comprehensive. Quantitative measurement of an operational system's understandability is desirable both as an instantaneous measure and as a predictor of understandability over time. This work proposes the method of measuring understandability using Logical Scoring of Preferences (LSP) method. I have also evaluated one design through this model.

Keywords: Software Quality, Quantitative Measurement, LSP.

1 Introduction

The demand of the quality software is increasing at rapid pace due to the society's increasing dependence on software. Measuring quality in the early stage of software development is the key to develop high-quality software. Wrong interpretations can lead to misunderstandings and to faulty development results. It is difficult to manage and improve the process without understanding and the ability to properly express the process in use Therefore, the readability and understandability of the software has a lot of influence on the factors that directly or indirectly affect software quality. Complex design may lead to poor testability, which in turn leads to ineffective testing that may result to severe penalties and consequences. It is well understood fact that flaws of design structure have a strong negative impact on quality attributes. But, structuring a high-quality design continues to be an inadequately defined process [1]. Therefore, software design should be built in such a way so as to make them easily understandable, testable, alterable, and preferably stable. This work focuses on the understandability assessment during the design phase to produce quality software.

Our methodology for the quantitative evaluation of software's understandability in the design phase is based on the core evaluation models and procedures are grounded in the LSP model and continuous preference logic as mathematical background [2]. Kumar and Soni [4] have proposed a hierarchical model of quality attributes. This is used to evaluate quality of human resource system design which was proposed by Kumar and Gandhi [3].

K. Singh, A.K. Awasthi, and R. Mishra (Eds.): QSHINE 2013, LNICST 115, pp. 951–961, 2013.

2 Previously Proposed Quality Models for Object- Oriented Software Products

One of the earliest software product quality model was suggested by McCall et.al.[5]. They defined software product qualities as a hierarchy of factors, criteria and metrics. The McCall's quality factors are correctness, reliability, efficiency, integrity, usability and maintainability. Boehm [6] described a set of quality characteristics. International bodies ISO/IEC came up with ISO9126 model for ensuring quality in software products. The ISO9126 [7, 17] model defines six quality attributes namely functionality, reliability, efficiency, usability, maintainability and portability. They are further subdivided into 26 sub-attributes (criteria) and nearly 100 sub-criteria or metrics. All these models were developed for structured methodology of software product development.

Even though there are many object-oriented analysis and design methodologies, languages, database management systems and tools, relatively less work has been done in the area of object-oriented design quality assurance [7, 8]. However, many metrics were developed to measure size and complexity of an object-oriented software system. One of the most popular set of metrics (commonly know as CK Metrics suite) was proposed by Chidamber and Kemerer [9]. The same suite was later refined and presented with empirical validation by Chidamber and Kemerer [10]. Basili et.al. [11, 12] also performed the empirical validation of CK metrics suite.

A framework for building product based quality models has been developed by Dromey [13,14]. The framework is a methodology for development of quality models in a bottom-up fashion, providing an approach that will ensure that the lower-level details are well specified and computable [12]. Bansiya et.al. [15] extended this methodology to develop the hierarchical Quality-Model for Object-Oriented Design (QMOOD) assessment. In the Quality Model for Object-Oriented Design (QMOOD), Bansiya et.al [15] identified the initial set of design quality attributes as: functionality, effectiveness (efficiency), understandability (maintainability), extendibility (portability), reusability and flexibility.

Further, Keller and Cockburn [16] organized a workshop, in which one group agreed upon following list of perspectives, with each having substantial influence on the quality of design artifacts: maintainability, documentation, extensibility, cost, reliability, ease of use, internationalization, usability, market goals, performance, team structure.

The second group discussed design properties that are of interest for project participants (developers) and gave following attributes: clarity, simplicity, scalability, modifiability, extendibility, reusability, effectiveness, reliability, robustness, security, and cost.

The metrics proposed by Bansiya et.al. [15] are quite general in nature and they have not provided the methodology to measure these metrics. Keller and Cockburn [16] have observed that there was no consensus on the quality attributes. However, they prescribed attributes and metrics that are very broad in nature and are not in conformance with ISO/IEC 9126 standards.

The author in her previous work [4] has given a generic model which assesses quality of design in early stage of software product development life cycle. This hierarchical model is based on five factors, their sub-factors and metrics and shown in Figure 2.1. These five-factors for design quality assessment are: functionality (modifiability), effectiveness (efficiency), understandability (usability), reusability, and maintainability (flexibility).

3 Steps for the Evaluation of Design Quality

Steps required for the evaluation of design quality are:

1. ***Consider a hierarchical model for quality characteristics and attributes (i.e. A1 …. An):*** here, evaluators should define and specify the quality characteristics and attributes, grouping them into a model. For each quantifiable attribute Ai, we can associate a variable Xi, which can take a real value: the measured value.

2. ***Defining criterion function for each attribute and applying attribute measurement:*** In this process, the evaluators should define the basis for elementary evaluation criteria and perform the measurement sub-process. An elementary evaluation criterion specifies how to measure quantifiable attributes. The result is an elementary preference, which can be interpreted as the degree or percentage of satisfied requirement. For each variable Xi , i = 1, ...,n it is necessary to establish an acceptable range of values and define a function, called the elementary criterion. This function is a mapping of the measured value in the empirical domain [18] into the new numerical domain. Then the final outcome is mapped in a preference called the elementary quality preference, EQi. We can assume the elementary quality preference EQi as the percentage of requirement satisfied by the value of Xi . In this sense, EQi = 0% denotes a totally unsatisfactory situation, while EQi = 100% represents a fully satisfactory situation [2]. Ultimately, for each quantifiable attribute, the measurement activity should be carried out.

3. ***Evaluating elementary preferences:*** In this task, the evaluators should prepare and enact the evaluation process to obtain an indicator of partial preference for design. For n attributes, the mapping produces n elementary quality preferences.

4. ***Analyzing and assessing partial quality preferences*****:** In this final step, the evaluators analyze and assess the elementary, partial and total quantitative results regarding the established goals.

3.1 Establishing Elementary Criteria for Understandability

The significance of understandability is very obvious that can be perceived as ‘If we can't learn something, we won't understand it. If we can't understand something, we can't use it - at least not well enough to avoid creating a money pit. We can't maintain a system that we don't understand - at least not easily. And we can't make changes to our system if we can't understand how the system as a whole will work once the changes are made’ [22]. Understandability of software documents is thus important as ‘the better we know what the thing is supposed to do, the better we can test for it’.

A good software design with manageable complexity usually provides proper data abstraction; it reduces coupling while increasing cohesion that make them easily understandable. As advocated by researchers and practitioners that understandability aspect of software is highly desirable and significant for developing quality

1. Functionality
    - 1.1 Design Size
        - 1.1.1 Number of Classes (NOC)
    - 1.2 Hierarchies
        - 1.2.1 Number of Hierarchies (NOH)
    - 1.3 Cohesion
        - 1.3.1 Cohesion Among Methods of Class (CAM)
    - 1.4 Polymorphism
        - 1.4.1 Number of Polymorphic Methods (NOP)
    - 1.5 Messaging
        - 1.5.1 Class Interface Size (CIS)
2. Effectiveness
    - 2.1 Abstraction
        - 2.1.1 Number of Ancestors (NOA)
        - 2.1.2 Number of Hierarchies (NOH)
        - 2.1.3 Maximum Depth of Inheritance (MDIT)
    - 2.2 Encapsulation
        - 2.2.1 Data Access Ratio (DAR)
    - 2.3 Composition
        - 2.3.1 Number of aggregation relationships (NAR)
        - 2.3.2 Number of aggregation hierarchies (NAH)
    - 2.4 Inheritance
        - 2.4.1 Functional Abstraction (FA)
    - 2.5 Polymorphism
        - 2.5.1 Number of Polymorphic Methods (NOP)
3. Understandability
    - 3.1 Encapsulation
        - 3.1.1 Data Access Ratio (DAR)
    - 3.2 Cohesion
        - 3.2.1 Cohesion Among Methods of Class (CAM)
    - 3.3 Inheritance
        - 3.3.1 Functional Abstraction (FA)
    - 3.4 Polymorphism
        - 3.4.1 Number of Polymorphic Methods (NOP)
        - 5.6.1 Number of aggregation relationships (NAR)
        - 5.3.2 Number of aggregation hierarchies (NAH)

4. Reusability
    - 4.1 Design Size
        - 4.1.1 *Number of Classes (NOC)*
    - 4.2 Coupling
        - 4.2.1 *Direct Class Coupling (DCC)*
    - 4.3 Cohesion
        - 4.3.1 *Cohesion Among Methods of Class (CAM)*
    - 4.4 Messaging
        - 4.4.1 *Class Interface Size (CIS)*
5. Maintainability
    - 5.1 Design Size
        - 5.1.1 *Number of Classes (NOC)*
    - 5.2 Hierarchies
        - 5.2.1 *Number of Hierarchies (NOH)*
    - 5.3 Abstraction
        - 5.3.1 *Number of Ancestors (NOA)*
    - 5.4 Encapsulation
        - 5.4.1 *Data Access Ratio (DAR)*
    - 5.5 Coupling
        - 5.5.1 *Direct Class Coupling (DCC)*
        - 5.5.2 *Number of Methods (NOM)*
    - 5.6 Composition
        - 5.6.1 *Number of aggregation relationships (NAR)*
        - 5.3.2 *Number of aggregation hierarchies (NAH)*
    - 5.7 Polymorphism
        - 5.7.1 *Number of Polymorphic Methods (NOP)*
    - 5.8 Documentation
        - 5.8.1 *Extent of Documentation (EOD)*

Fig. 2.1. Hierarchical design quality assessment model

software. Through the findings of literature survey there are various aspects of software that either directly or indirectly influences quality of software design including understandability factor [19], [20].

Therefore, out of the five factors of the hierarchical model [4] I have focused on the understandability aspect in this work. Understandability is further decomposed into four sub factors namely: encapsulation, cohesion, inheritance and polymorphism. However, I have measured only three sub-factors in this work and they are: encapsulation, cohesion and polymorphism.

For each attribute Ai we can associate a variable Xi which can take a real value by means of the elementary criterion function. The final result represents a mapping of the function value into the elementary quality preference, EQi. The value of EQi is a real value that 'fortunately' belongs to the unit interval. As stated by Dujmovic et al. in [2]:

"*the elementary preference is interpreted as a continuous logic variable. The value 0 denotes that Xi does not satisfy the requirements and the value 1 denotes a perfect satisfaction of requirements. The values between 0 and 1 denote a partial satisfaction of requirements. Consequently, all preferences are frequently interpreted as a percentage of satisfied requirements, and defined in the range [0, 100%]*".

Further, the preference can be categorized in three rating levels namely: satisfactory (from 60 to 100%), marginal (from 40 to 60%), and unsatisfactory (from 0 to 40%). For instance, a marginal score for an attribute could indicate that a correction action to improve the attribute quality should be taken into account by the manager or developer. Figure 3.1, shows two elementary criteria for attributes of understandability. There are two major categories to classify elementary criteria, that is, absolute and relative criteria. Moreover, regarding the absolute elementary criteria, these are further decomposed in continuous and discrete variables.

The preference scale for the *Data Access Ratio (DAR)* metric is a multi-level discrete absolute criterion defined as a subset, where 0 implies ratio is less then 5%; 80% or more implies satisfactory (100%) ratio.

The resulting value of this discrete multivariable absolute criterion could be between 0 (completely unsatisfactory) and Xmax (completely satisfactory). If the measured value of X is above Xmax, the corresponding elementary preference X will be equal to Xmax. Similar criteria were followed for other metrics as well.

3.2 Computing Partial Preference for Maintainability

In this process, the evaluators should define and prepare the evaluation process to obtain a quality indicator for each competitive system. Applying a stepwise aggregation mechanism, the elementary quality preferences can be accordingly structured to allow the computing of partial preferences. Thereby global preferences can be obtained through repeating the aggregation process at the end. The global quality preference represents the global degree of satisfaction of all involved requirements. Here I am computing partial preferences for understandability. In this study, we use a logical scoring of preferences model called LSP model. A broad treatment of LSP relationships and continuous Logic Preference (CLP) operators could be found in [2, 21], as well as the mathematical background.

The strength of LSP resides in the power to model different logical relationships to reflect the stakeholders' needs, namely:

- Simultaneity, when is perceived that two or more input preferences must be present simultaneously
- Replaceability, when is perceived that two or more attributes can be replaced (there exist alternatives, i.e., a low quality of an input preference can always be compensated by a high quality of some other input).
- Neutrality, when is perceived that two or more input preferences can be grouped independently (neither conjunctive nor disjunctive relationship)
- Symmetric relationships, when is perceived that two or more input preferences affect evaluation in the same logical way (tough may be with different weights)
- Asymmetric relationships, when mandatory attributes are combined with desirable or optional ones; and when sufficient attributes are combined with desirable or optional ones.

Figure 3.2, depicts the aggregation structure for understandability characteristic. The stepwise aggregation process follows the hierarchical structure of the hierarchical model from bottom to top. The major CLP operators are the arithmetic means (A) that models the neutrality relationship; the pure conjunction (C), and quasi-conjunction operators that model the simultaneity one; and the pure disjunction(D), and quasi-disjunction operators that model the replaceability one. With regard to levels of simultaneity, we may utilize the week (C-), medium (CA), and strong (C+) quasi-conjunction functions. In this sense, operators of quasi-conjunction are *flexible and* logic connectives. Also, we can tune these operators to intermediate values. For instance, C-- is positioned between A and C- operators; and C-+ is between CA and C operators, and so on. The above operators (except A) mean that, given a low quality of an input preference can never be well compensated by a high quality of some other input to output a high quality preference. For example at the end of the aggregation process we have the sub-characteristic coded 3.1 (called Encapsulation in the hierarchical Model, with a relative importance or weight of 0.3), and 3.2 sub-characteristic (Cohesion, 0.4 weighted), and 3.4 sub-characteristic (polymorphism, 0.3 weighted).

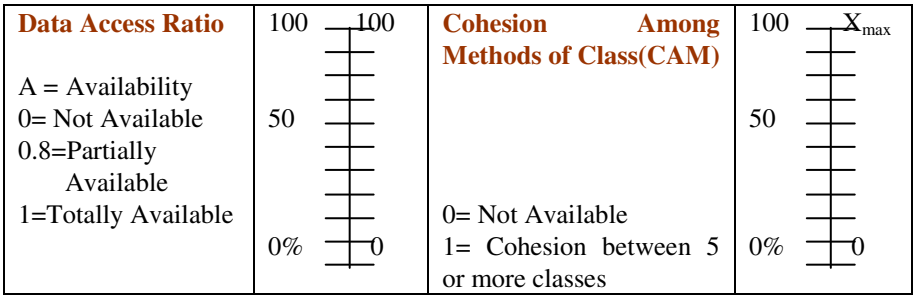

Fig. 3.1. Sample elementary criteria defined as preference scales taken from the hierarchical model

All these sub-characteristic preferences are input to the C-- logical function, which produce the partial global preference coded as 3, (called Understandability).

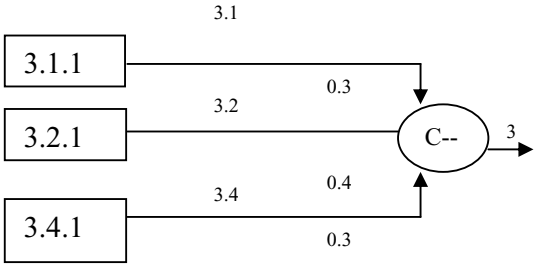

Fig. 3.2. Structure of Partial Logic Aggregation for Understandability

Similarly, we can also utilize the quasi-disjunction operators in a range of strong (D+), medium (DA), and week (D-) *or* polarization, and also their intermediate values. For instance, D-- is positioned between A and D- operators; and D-+ is between DA and D- operators; and D+- is between D+ and DA operators; and finally, D++ is between D+ and D operators. D operator represents the pure disjunction.

4 Assessing Understandability of the Design Selected

Figure 4.1 shows the design of human resource management information system, which is developed to take care of the important function of the Human Resource Development. The system keeps record of the employees both regular and ad-hoc along with their qualification details, the designation at the time of joining the organization, the present designation and number of promotions any employee has been given since he joined the organization. It keeps the detailed record of employee family members, medical facilities along with his telephone number, job responsibilities of each and every employee and the reporting officer/person of each employee is also maintained and several other information as shown in Fig 4.1.

In the evaluation process, I decided the elementary criterion for each metric, as shown in fig 3.1. I then confronted partial preferences as shown the section 3.2 and fig 3.2.

The partial outcomes for each subfactor and the total outcome for understandability is shown in Table 1.

This shows that the design of the human resource information system is falling into a satisfactory level because it has 85.79% of the quality preference.

Table 1. Detailed result of partial quality preferences after computing the aggregated criteria function of the design

Characteristics and Sub-characteristics	Values
3. Understandability	
3.1 Encapsulation	
3.1.1 Data Access Ratio (DAR)	.8
3.2 Cohesion	
3.2.1 Cohesion Among Methods of Class (CAM)	.8
3.4 Polymorphism	
3.4.1 Number of Polymorphic Methods (NOP)	1
Partial Quality Preference	**85.79**

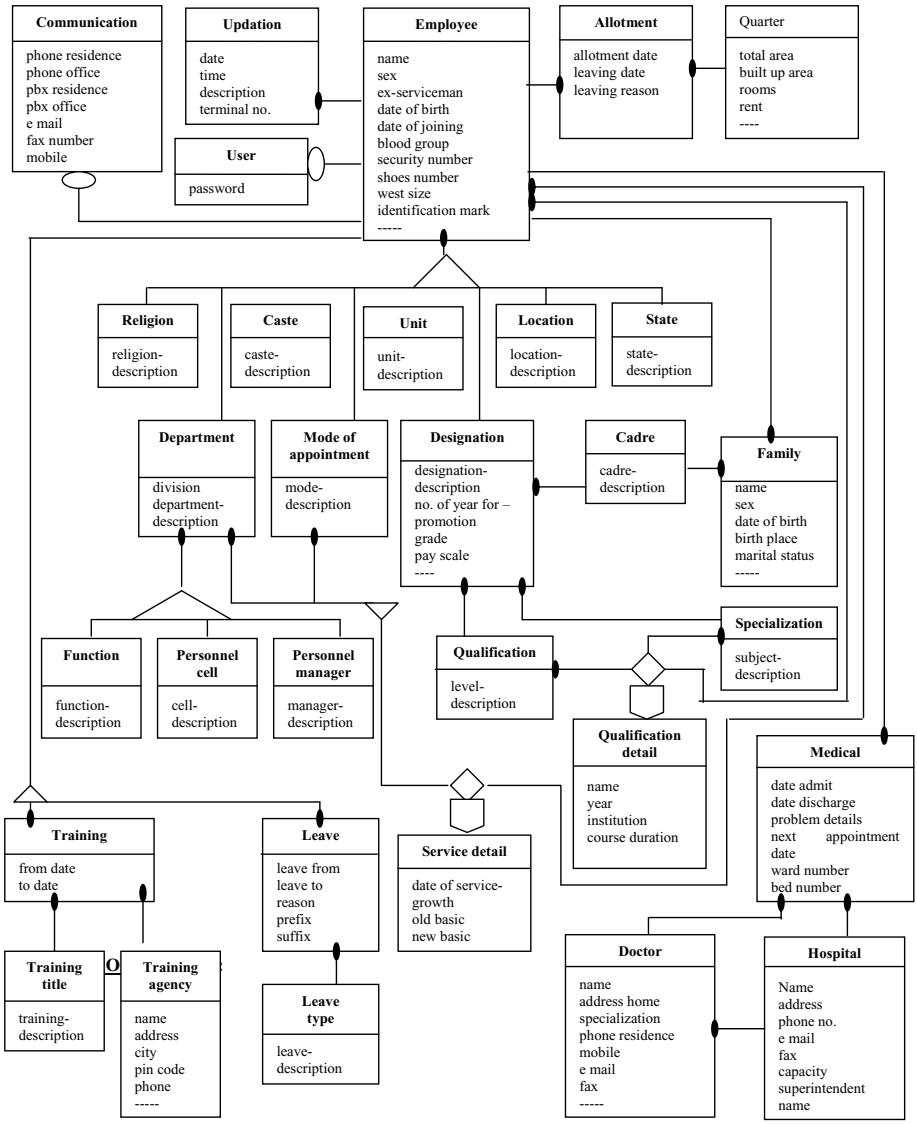

Fig. 4.1. Class Diagram for Human Resource Information System

5 Conclusion

In this work we have proposed a methodology, for the quantitative evaluation of software's understandability in the design phase. The core evaluation model and procedures are grounded in the LSP model and continuous preference logic. The attributes and metrics of understandability are measured from the hierarchical model proposed by Kumar and Gandhi [3]. The weights assigned for preferences are

arbitrary and can be changed according to the requirement. I have found that the understandability of design [3] came out to be 85.79 which means that the system will be easy to understand.

The method is suitable for comparing alternative designs of a system for understandability aspect. This will help choose a design that is most suited for understanding especially when the software has been deployed.

References

1. Valdaliso, C., Eljabiri, O., Deek, F.P.: Factors Influencing Design Quality and Assurance in Software Development: An EmpiricalStudy. In: Electronic Proceedings of the First International Workshop on Model-based Requirements Engineering (MBRE 2001), San Diego, California, pp. 78–82 (2001)
2. Dujmovic, J.J.: A Method for Evaluation and Selection of Complex Hardware and Software Systems. In: Proceedings of the 22nd International Conference for the Resource Management and Performance Evaluation of Enterprise CS, CMG 1996, vol. 1, pp. 368–378 (1996)
3. Kumar, M., Gandhi, S.K.: Object-Oriented Modeling Design Approach to General Human Resource System. Journal of MACT 2, 34–35 (2003-2004)
4. Kumar, M., Soni, D.: Observations on Object-Oriented Design Assessment and Evolving New Model. In: Proc of the National Conference on Software Engineering, pp. 161–164 (2007)
5. McCall, J.A., Richards, R.K., Walters, G.F.: Factors in Software Quality, National Tech. Information Service, Springfield, Va, vols. 1,2, and 3 (1977), AD/A-049-014/015/055
6. Boehm, B.W.: Characteristics of Software Quality. TRW Inc. (1978)
7. Software Product Evaluation - Quality Characteristics and Guidelines for Their Use, ISO/IEC Standard ISO-9126 (1991), `http://www.cse.dcu.ie/essiscope/sm2/9126ref.html`
8. Olague, H.M., Etzkorn, L.H., Messimer, S.L., Delugach, H.S.: An Empirical Validation of Object-Oriented Class Complexity Metrics and their Ability to Predict Error-prone Classes in Highly Iterative, or Agile Software: a Case Study. Journal of Software Maintenance 20(3), 171–197 (2008)
9. Chidamber, S.R., Kemerer, C.F.: Towards a Metric Suit for Object-Oriented Design. In: Proc. of Sixth Object-Oriented Programming Systems, Languages and Applications, pp. 197–211 (1991)
10. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite For Object-Oriented Design. IEEE Trans. Software Eng. 20(6), 476–493 (1994)
11. Basili, V., Briand, L., Melo, W.: A Validation of Object-Oriented Design Metrics as Quality Indicators. IEEE Transactions of Software Engineering 22(10), 751–761 (1996)
12. Elish, M.O., Elish, K.O.: Application of TreeNet in Predicting Object-Oriented Software Maintainability: a Comparative Study. In: Proc. of European Conference on Software Maintenance and Reengineering (CSMR 2009), March 24-27, pp. 69–78 (2009)
13. Dormey, G.R.: A Model for Software Product Quality. IEEE Trans. Software Eng. 21(2), 146–162 (1995)
14. Dormey, G.R.: Cornering the Chimera. IEEE Software 13(1), 33–43 (1996)
15. Bansiya, J., Davis, C.G.: A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering 28(1), 4–17 (2002)

16. Keller, R.K., Cockburn, A.: Object-Oriented Design Quality. In: OOPSLA, Workshop#12, Atlanta, Georgia, pp. 63–67 (1997)
17. Antonellis, P., Antoniou, D., Kanellopoulos, Y., Makris, C., Theodoridis, E., Tjortjis, C., Tsirakis, N.: A Data Mining Methodology for Evaluating Maintainability According to ISO/IEC-9126 Software Engineering Product Quality Standard. In: Proc. 11th IEEE Conference on Software Maintenance and Reengineering (CSMR 2007), March 21-23, pp. 35–42 (2007)
18. Fenton, N.E., Pfleeger, S.L.: Software Metrics: a Rigorous and Practical Approach, 2nd edn. PWS Publishing Company (1997)
19. Gao, J., Ming-Chih, S.: A Component Testability Model for Verification and Measurement. In: Proceedings of the 29th Annual International Computer Software and Applications Conference, pp. 211–218. IEEE Computer Society (2005)
20. Jimenez, G., Taj, S., Weaver, J.: Design for Testability. In: The Proceedings of the 9th Annual NCIIA Conference (2005)
21. Olsina, L.S.: Web-site Quality Evaluation Method: a case Study on Museums. In: ICSE 1999 – 2nd Workshop on Software Engineering over the Internet (1999)
22. Jacob, B., Niklas, L., Waldermarsson, P.: Relative Indicators for Success in software development. In: Department of Communication Systems. Lund University (2001)