2012 International Conference on Software and Computer Applications (ICSCA 2012)
IPCSIT vol. 41 (2012) © (2012) IACSIT Press, Singapore

Quantifying Software Reliability Attribute through the Adoption of Weighting Approach to Functional Requirements

Mona Abd Elghany [1], Nermine Khalifa [2] and Marwa Abd Elghany [3]

[1]Assistant Professor in Productivity and Quality Management, College of Management & Technology, Arab Academy for Science & Technology

mabdelghany2000@gmail.com

[2] Assistant Professor in Business Information Systems, College of Management & Technology, Arab Academy for Science & Technology

nermine_khalifa@aast.edu, nerminek@gmail.com

3Lecturer in Business Information Systems, College of Management & Technology, Arab Academy for Science & Technology

marwam@aast.edu

Abstract - High quality software can hardly to be assured. The Software project reports refer to failure rate of 71% due to exceeding budget or did not fulfill business requirements. Therefore, Quality attributes could not be achieved without certain requirements specified by project managers that should be exhibited within the system thus. Different stakeholder groups are involved in software development and use their conceptual quality requirements to refine the broad concept of quality into a number of well-defined and measurable attributes related to the software product itself. Though, despite the growing awareness of the importance of achieving similar configuring alignment between business stakeholders and software developers, little attempt has been made to empirically examine the requirements for software quality. Software Reliability is an important to attribute of software quality, Software Reliability is hard to achieve, because the complexity of software tends to be high. Providing a failure-free software operation for a certain time within pre-defined environmental setting is key determination of software reliability. This paper is focusing on reliability attributes and its measures. The paper uses weighting factors of software quality attributes in an attempt to quantify the reliability attribute. This paper conducted an empirical study in order to rate the importance of reliability features related to software reliability attribute quality. The paper applies a weighting technique in order to indicate the potential of each reliability features. The results suggested that alike approach to the other quality attributes, like maintainability for example, might align the software developers with the business stakeholders' desired requirements. Direct measurement of quality attributes should be stimulated and in fact such quantified measurement can be utilized to establish consistency using the existing approach. However, the approach needs to be made more accessible to promote its use in order to decide whether consistent independent estimates of the true values of software quality attributes can be assigned for quality attributes developed.

Keywords: functional requirements, software quality attributes, software measurement, reliability features

1. Introduction

The main objective of this paper is to give a clear illustration about a designed questionnaire that will be utilized to be displayed to the software project manager to specify the level of reliability that is sought to be applied in the desired software system. It is hoped that this work will act as a useful example to practitioners in the near future for other software quality attributes like maintainability for example, but with different questioning criteria.

* Corresponding author. Mona Abd Elghany Tel.: +2/035565429
E-mail address: mabdelghany2000@gmail.com

In this paper, the second section is dedicated to discuss the software requirement specifications and exemplifying the adopted approach of rating the software reliability features through assigning different weights in percentage to these functional requirements, interpreted into reliability quality features, according to its fundamentality in the desired software product. Subsequently in the third section, focusing on software reliability requirements and presenting the reliability questionnaire resulted from the conducted interviews with different IT specialists and projects managers, also with the demonstration of the data gathering technique and the description of the questionnaire equation.

2. Literature Review

Assuring the quality of software cannot be easily maintained in the process of software development. A survey of software project in the period between 2002 and 2004 indicates either a huge failure or complete collapse of these projects. This survey refers such failure either to the overdue of project s' timing or exceeding the estimated budget. In the same manner, a number of projects were abandon before their delivery [15]. Therefore, software quality had emerged and highlighted in different phases of software development. Reference [8] discussed the Software quality. She refers to thirteen sub-attributes that can define the concept of high quality software. Most of literature deals with quality attributes as standard attributes applicable for all types of software regardless its fitness in various organizational setting and the perception of stakeholder for the quality term. Therefore, fitness of software to organizational needs had been discussed by many researchers to reflect the effect of IT expertise and cultural issues on software appropriateness. The concept of ''IT-Business alignment" has been recently engaged to define the meaning of quality term and resolve the conflict of interest and preferences between the stakeholders. The emergence of aligning the software to organizational setting is mandatory in order to justify its investments. Reference [10] focused on the variance of quality perceptions between IT-expert and non-expert while [8] investigated same issue and included the managerial levels to indicate their perception. The researchers indicate that if there a gap in between to define "Quality software", the proposed system would barley fulfill the business requirements. Therefore, sharing an identical meaning of quality term is essential in order to reach affordable and applicable piece of software.

Accordingly, some researches prefer to classify the non-functional into consumer-oriented and technical-oriented requirements [2]. Many researchers pointed that consumer-oriented attributes are relative one and cannot be assessed by absolute value. Customer acceptance coupled with the time consumed and cost burden are key issue to be considered while weighting software quality requirement [7]. In order to reach an acceptable level of software requirement, the desired values of attributes have to be defined in-advance. Fitness of software in certain organizational setting may differ and indicate a different scale of applicability for the same software used by different users. Different users are dealing with the same software from different perspectives so the term of software quality differs consequently ([1]; [6]). Therefore, decision makers and system developers are required to set an acceptable level of software quantity accordance to the cost maintained.

Many researchers such as [14] indicate that it is hard to obtain software that can totally satisfy the quality requirements. Sometimes, direct more fund or human resource to improve quality of software might lead to negatives consequences. As a result, different quality measures may fit different business environment and needs so managers need to define the potential of quality measures for their business. Different priorities can be defined to software quality attributes according to its fitness in the organizational setting and its value from stakeholders' perspectives [8].Software quality attributes are considered to be subjective and have a variable scale in conformity to stakeholders' perception [5]. ISO defines several quality attributes that are considered as controllable attributes, these attributes are relative and variable from different perspectives. For example, [13] points out that maintainability of software may differ according to the time consumed in maintenance and the applicability of doing the required changes.

Reference [4] and [12] proposed hieratical models for quality measures, therefore it was agreed that there is no need for estimating a measure subjectively. Preliminary level of quality model will be assessed so low-level attributes will be assigned to numerical value. Since, there is no formal relationship model that can

indicate how to relate quality attributes with low level measures. Estimating time and effort needed for each task would provide an objective measure for each attribute [11]. Some researchers propose using a comparison technique in order to validate subjective measurements. This validation technique will be used as an alternative of referring back to a conceptual model or piece of theory.

Reliability is one of the key attributes in software quality. Reference [3] defines the reliability of a system to "a measure of the ability of a system to keep operating over time". They classify the reliability as sub-attribute of dependability where computer system can reasonably provide the service expected to.

Others define reliability as "probability of failure-free software operation for a specified period of time in a specified environment." Referring to ISO-9126 Software Quality Characteristics, reliability is defined as "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". Reference [3] assumed that the dependability of system includes reliability, availability, safety, and security. The figure (1) demonstrates a number of concerns, factor and method that are considered too critical for dependability view.

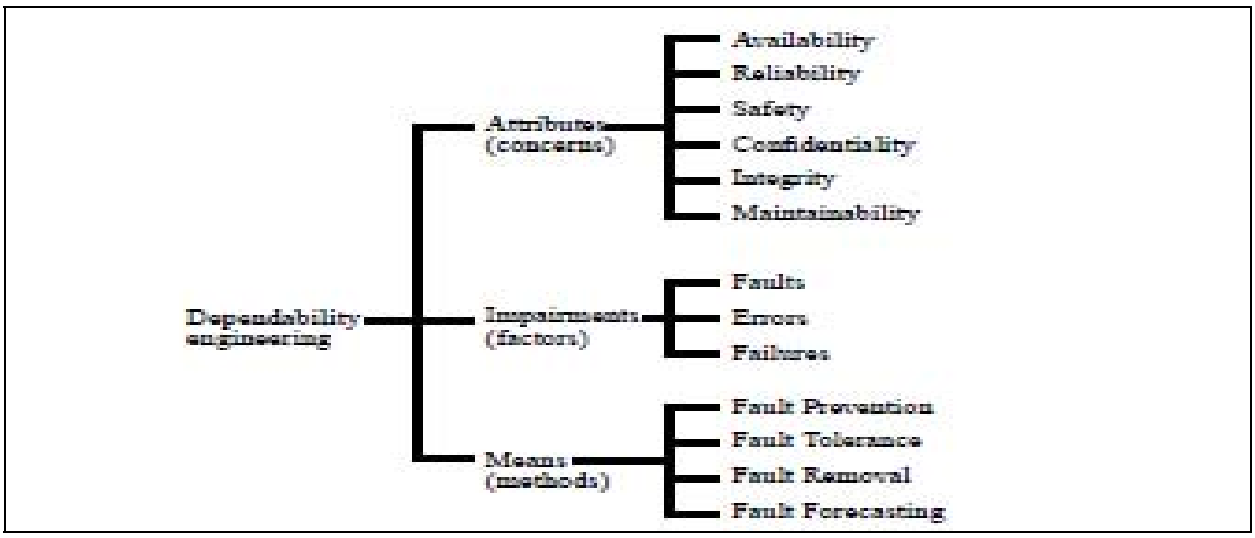

Fig. (1): Dependability Tree; Source: [3]

Reference [3] focused on measuring different factor that might affect dependability of software. They define different taxonomies for failures, errors and faults. A failure occurs when there is deviation in the behavior of system. There is different failure modes included in failure taxonomy: domain failures, perception by the users, and consequences on the environment. Domain failures might be value or timing orientated. A value failure means that system ends up with inappropriate outcome while timing one is related to the timeline needed for a task. Perception failure can be classified into consistent and inconsistent according to user perspectives. Users may share the same perception of failure or may this perception may differ from a user to another. Consequence failures include wide range of failures from benign or minor one to a total catastrophic one. Figure (2) indicates the taxonomy proposed by authors for failure classification.

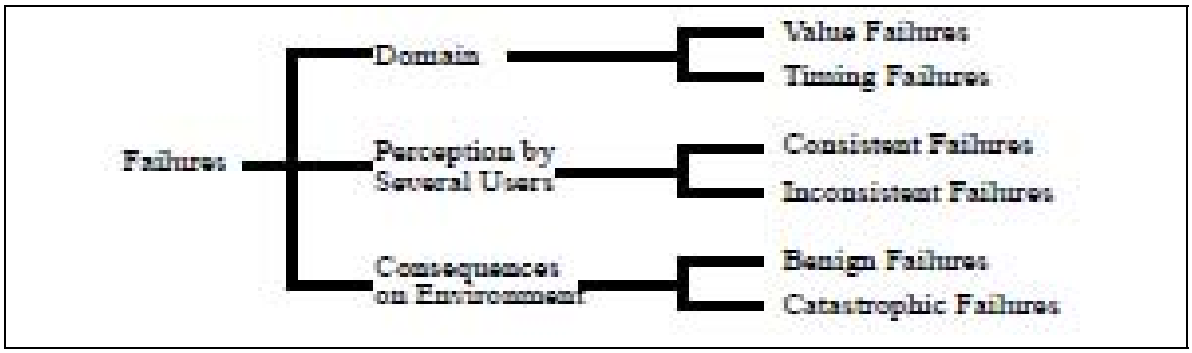

Fig. (2): Failure Classification; Source: [3]

The authors differentiate between error and failures and indicate that the error is state that is liable to lead to a type failure. Error may occur in idleness function of system, or might be acceptable by system users.

3. Software Requirements Specifications

The requirements for a system are the descriptions of what the system should do, the services that it provides and the constraints on its operation. The software requirements specification (SRS) is an official statement of what the system developers should implement. It should include a detailed specification of the system requirements. It helps to understand what the system is supposed to do and how its use can support users and satisfy stakeholders. It is critical to the success of a development project. Accordingly it should be based on a model in which the result is an unambiguous and complete specification document. The generic IEEE standard 830-1998 [9] describes recommended approaches for the SRS. It describes the content and qualities of a good SRS. It can be adapted to the considered project.

3.1. Reliability Questionnaire

When discussing reliability requirements, multiple definitions of reliability requirements are difficult to understand satisfaction criteria, and lack a clear track for deriving reliability requirements from business goals.

From the above it can be said that reliability requirements represent constraints on functional requirements that are needed to satisfy applicable system's reliability goals and they express these goals in operational terms, precise enough to be given to the developer.

The authors report here a survey to examine software reliability requirements at the early stage of the software development life cycle. Interviews handled with a small sample size of fewer than 40 project managers (including requirements' engineers, system designers, software developers, system operational support, software maintenance specialists, testers, etc.) whom currently and previously were engaged in managing software development projects from the Information and Documentation Center and also staff members of the Computing & Technology Faculty, in the Arab Academy for Science and Technology (AAST) due to their expertise in programming and analysis, within the collaborating institution. Potential responses to each interview question were recorded to ensure that each question sought sufficient and appropriate data. The resultant list of questions was reviewed and modified to respond to comments. Although the survey can reach a larger number of potential respondents, it was limited in its ability to permit the authors to follow up and clarify the questions and their responses. Then, the survey was carried out and a questionnaire was distributed to the same sample. The checklist used in the distributed questionnaire is for eliciting and prioritizing reliability requirements when developing application projects. This is obtained by judging the degree of relevance of each statement to software reliability requirements from the participants' real experience to determine the weighting factor of the question number i = $Wf(q_i)$ = (answer value to question i /summation of all answer values to all questions) =

$$\sum^{m} [ans_i / \sum^{n} ans_i] / m \quad (1)$$

Where m is the number of distributed questionnaires to participants; $0 = <ans_i = < 5$ (Likert Scale) and n is the number of questions which is equal to 24 in case of reliability attribute. Then using this weighting factor of the question number i to calculate the measure of the software quality attribute in percentage as follows:

$$\sum_{i=1}^{n} [wf(q_i) * wf(ans)_{qi}] \quad (2)$$

$Wf(ans)_{qi}$ is the weighting factor of the answer to question number i that would vary according to the responding answers of the project managers depending on the project application type the manager desires to develop; its value would be 0 if he does not need this feature to be included in the required software otherwise it would be equal to 1 if he seeks to apply this feature.

Moreover, it is presumed that the weighting of any question was depending on the experience of the participant (who is an employer in the Information & Documentation Center assuming that he developed several types of projects such as financial, human resources, web based application, and so on…) thus he is giving an accurate weight for every reliability feature required; those weights would be constants in the program.

The reliability requirements that would be outlined in the following sections apply to all information systems as tailored from literature review, and the structured interviews conducted by the authors as well. The questionnaire (see table 1) was divided into five parts as illustrated consecutively below.

Table 1: Produced Questionnaire from Conducted Interviews; Source: (The Authors)

Question Element		**Importance Degree**				
First: Questions relating to monitoring failure type using the event logging mechanism	**NA**	**1**	**2**	**3**	**4**	**5**
1. Identifies error to handle input						
2. Detects error to produce output like hangs						
3. Encounters error to produce correct output like crash failures						
4. Registers application closure like normal exits, and user enforced exits						
5. Records statement-related failures embedded in the program source line of codes						
6. Notifies the users in case of resource unavailability or failure related to an action handling (e.g. file does not exist, network not available, file writing fails, etc.)						
7. States the used platform, the hardware configuration and the software version in which the failure occurred						
Second: Questions relating to measuring performance ratio of software application	**NA**	**1**	**2**	**3**	**4**	**5**
8. Gives the session timeline						
9. Counts the number of user sessions						
10. Records the user inactive time per session						
11. Scores the actual user usage						
Third: Questions relating to availability through runtime diagnosis and shutdown event tracker to trace downtimes and restores	**NA**	**1**	**2**	**3**	**4**	**5**
12. Reports time taken to remove or repair a fault for returning back the system to its ready-state						
13. Allocates the recovery times for the annotation of the number of faults						
14. Normalizes the number of faults by the code size estimates to be expressed in terms of fault densities						
15. Assesses configuration changes that require a restart to make file registry						
16. Identifies fault prevention that could result from preventive maintenance						
17. Recognizes fault prevention resulting from supply response						
18. Pinpoints fault prevention resulting from administrative delays						
Fourth: Questions relating to interoperability to integrate with other applications and management flexibility	**NA**	**1**	**2**	**3**	**4**	**5**
19. Detects error due to installation setting configuration						
20. Provides the timeline needed to reach a stable state						
21. Discloses the number of reboots taken to perform any action is employed						
Fifth: Questions relating to system ability of the operational environments	**NA**	**1**	**2**	**3**	**4**	**5**
22. Considers the Difficulty of Programming						
23. Indicates the Level of Programming Technologies						
24. Detects the Percentage of Reused Code						

3.2. Study Results

The pilot study yielded the following results after the data analysis of the distributed questionnaires as shown in table below.

Table 2: Pilot Study Results; Source: (The Authors)

Question Number Q_i	Question Weight $W_i f$	Description		Score
		Selected Feature	Answer	

Q_1	$W_1f = 2\%$	a) identifies error to handle input= (1) b) does not identify error to handle input = (0)	a)✓	(2*1)= 2%
Q_2	$W_2f = 8\%$	a) detects error to produce output like hangs = (1) b) does not detect error to produce output = (0)	a)✓	(8*1)= 8%
Q_3	$W_3f = 15\%$	a) encounters error to produce correct output like crash failures= (1) b) incapable of encountering error to produce correct output like crash failures= (0)	a)✓	(15*1)= 15%
Q_4	$W_4f = 6\%$	a) registers application closure like normal exits, and user enforced exits= (1) b) does not register= (0)	a)✓	(6*1) = 6%
Q_5	$W_5f = 5\%$	a) records statement-related failures embedded in the program source line of codes= (1) b) does not record= (0)	a)✓	(5*1) = 5%
Q_6	$W_6f = 3\%$	a) notifies the users in case of resource unavailability or failure related to an action handling = (1) b) does not notify= (0)	a)✓	(3*1) = 3%
Q_7	$W_{7f} = 1\%$	a) states the used platform, the hardware configuration and the software version in which the failure occurred= (1) b) does not state= (0)	b)✓	(1*0) = 0%
Q_8	$W_8f = 7\%$	a) gives the session timeline= (1) b) does not give= (0)	a)✓	(7*1) = 7%
Q_9	$W_9f = 3\%$	a) counts the number of user sessions= (1) b) does not count= (0)	a)✓	(3*1) = 3%
Q_{10}	$W_{10}f = 4\%$	a) records the user inactive time per session= (1) b) does not record= (0)	a)✓	(4*1) = 4%
Q_{11}	$W_{11}f = 6\%$	a) scores the actual user usage= (1) b) does not score=(0)	a)✓	(6*1) = 6%
Q_{12}	$W_{12}f = 5\%$	a) reports time taken to remove or repair a fault for returning back the system to its ready-state= (1) b) does not report= (0)	a)✓	(5*1) = 5%
Q_{13}	$W_{13}f = 4\%$	c) allocates the recovery times for the annotation of the number of faults= (1) d) does not allocate= (0)	b)✓	(4*1) = 4%
Q_{14}	$W_{14}f = 1\%$	a) normalizes the number of faults by the code size estimates to be expressed in terms of fault densities=(1) b) does not calculate fault density =(0)	b)✓	(1*0) = 0%
Q_{15}	$W_{15}f = 4\%$	a) assesses configuration changes that require a restart to make file registry= (1) b) does not assess= (0)	a)✓	(4*1) = 4%
Q_{16}	$W_{16}f = 2\%$	a) identifies fault prevention that could result from preventive maintenance=(1) b) does not identify =(0)	b)✓	(2*0) = 0%
Q_{17}	$W_{17}f = 3\%$	a) recognizes fault prevention resulting from supply response=(1) b) does not require such recognition= (0)	b)✓	(3*0) = 0%
Q_{18}	$W_{18}f = 2\%$	a) pinpoints fault prevention resulting from administrative delays= (1) b) does pinpoint = (0)	b)✓	(2*0) = 0%
Q_{19}	$W_{19}f = 3\%$	a) detects error due to installation setting configuration= (1) b) does not detect= (0)	a)✓	(3*1) = 3%
Q_{20}	$W_{20}f = 3\%$	a) provides the timeline needed to reach a stable state= (1) b) does not provide= (0)	a)✓	(3*1) = 3%

Q_{21}	$W_{20}f = 4\%$	a) discloses the number of reboots taken to perform any action= (1) b) does not reveal= (1)	a)✓	(4*1) = 4%
Q_{22}	$W_{22}f = 2\%$	a) considers the Difficulty of Programming= (1) b) does not consider= (0)	b)✓	(2*0) = 0%
Q_{23}	$W_{23}f = 2\%$	a) indicates the Level of Programming Technologies= (1) b) does not indicate= (0)	b)✓	(2*0) = 0%
Q_{24}	$W_{24}f = 3\%$	a) detects the percentage of Reused Code= (1) b) does not indicate= (0)	a)✓	(3*1)= 3%
Total	100%	Software Reliability Quality Attribute Percentage		87%

The preceding table illustrates how a software quality attribute such as reliability would be represented in percentage (i.e. to be put into a quantity); taking into consideration that the given values of each question weight are supposed to be constants obtained from the distributed questionnaires as described earlier in the proposed formula.

If the required project to be developed is assumed to have the following specifications resulting of the evaluation of similar checklists to the rest of the attributes: Usability = 95%, Maintainability = 85%, Reliability = 90%, Security = 60%.

The use of % is indicative that the functionality attributes of the software do not represent 100% of the user requirements from each factor. The assumption being that user needs *may, indeed do, change with time*. This use of % unit is important since it shows the relative 'saturation' in the attributes with respect to current user requirements thus reflecting the 'maturing' user needs.

4. Conclusion

This paper exposed the vague pathway for deriving reliability requirements from business goals, and the shortage of specification and guidance for the software developers. The presented approach expresses concrete reliability requirements that are: more explicit about who can do, what, and when, articulating the system's reliability goals in operational terms that are precise enough to be given to a developer, and providing a specification, or a prescription, or a behavior in terms of phenomena to achieve the desired effect. Also, this work portrays the functional requirements tested in a binary mode, either the software product supports the requirement or it does not, therefore either passes or fails the test. For a non-functional requirement, the software product does or does not meet the requirement is the mutual equivalent perception. Yet, the main contribution of this paper remains in giving a weighing-style quantification of reliability requirements. Similar questionnaires would convey the related software features to other software quality attributes in order to obtain the needed software system quality, the business is looking for.

5. References

[1]. Arthur, l. J. (1985). *Measuring programmer productivity and software quality*. New York: Wiley.

[2]. Avizienis, A., Laprie, J.-C., et al. (2001). Fundamental concepts of computer system dependability. *IARP/IEEE-RAS workshop on robot dependability*: Technological challenge of dependable robots in human environments, Seoul, Korea, May 21–22, 2001.

[3]. Barbacci, Mario, Klein, Mark H., Longstaff, Thomas A., Weinstock, Charles B. (1995). *Quality Attributes*. Technical Report of Software Engineering Institute Carnegie Mellon University Pittsburgh, Pennsylvania.

[4]. Boehm, B. (1978). *Characteristics of software quality*. New York: North Holland.

[5]. Coniam, S. W., & Diamond, A. W. (1995). *Practical Pain Management*—a guide for practitioners. Oxford, UK: OUP.

[6]. Dromey, R. G. (1995). A model for software product quality. *IEEE Transactions on Software Engineering*, 21(2), 146–163.

[7]. Gentleman, W. M. (1996). Software quality world-wide: What are the practices in a changing environment: *Proceeding of the sixth international conference on software quality (6ICSQ)*, Ottawa, Canada.

[8]. Haigh, Maria (2010). Software quality: non-functional software requirements and IT-business alignment. *Software*

Quality Journal (18) 361–385, DOI 10.1007/s11219-010-9098-3
[9]. IEEE Std. 830-1998. 20 October 1998. IEEE Recommended Practice for Software Requirements Specifications, *IEEE Computer Society*. Software Engineering Standards Committee. No. SH94654.
[10]. Jeffery, M., & Leliveld, I. (2004). Best practices in IT portfolio management. *MIT Sloan Management Review*, 45(3), 41–49.
[11]. Kitchenham, B. A. and Walker, J. G. (1989). A quantitative approach to monitoring software development. *Software Engineering Journal*. 4 (1), 2-13.
[12]. McCall, J. A. et al. (1977). Concepts and definitions of software quality factors in software quality. *NTIS* (Vol.1). Springfield, VA: NTIS.
[13]. Rosenberg, J., (1997). Problems and prospects in quantifying software maintainability. *Journal of Empirical Software Engineering*. 2(2): 173-177.
[14]. Shumskas, A. F. (1992). *Software risk mitigation total quality management for software* (pp. 190–220).New York: G. G. a. J. I. M. Schulmeyer, Van Nostrand Reinhold.
[15]. The Standish Group's CHAOS Report 2004.