2011

A Dichotomous Stakeholder-Centric Software Evaluation Framework

Radu Vlas
Georgia State University, rvlas@cis.gsu.edu

A DICHOTOMOUS STAKEHOLDER-CENTRIC SOFTWARE EVALUATION FRAMEWORK

Radu Vlas
Georgia State University
rvlas@cis.gsu.edu

ABSTRACT

This research-in-progress presents a problem that eluded practitioners and researchers for a long period of time. While pressured to release new software to market very rapidly, software development companies find themselves often choosing between quality and timeliness. Since from an economical point of view the quality of a software product is determined by consumers' perceptions, this research proposes a dichotomous stakeholder-centric framework designed to support software quality assessment processes.

Keywords

Software quality, quality assessment, user perspective, developer perspective

INTRODUCTION

Software engineering domain represents a relatively young area of research when compared to social or natural sciences. It comes as a logical consequence that a generally accepted perspective on the specifics of some areas of software engineering has not yet been reached. Software product quality represents one such area in which there are continuous research efforts to crystallize a convergent perspective. Over time, researchers have attempted to define software quality through the lens of numerous views: product-based view, manufacturing-based view, user-based view, or various other views (Wong 2002). It is important to note that only few software quality perspectives adopted by researchers include stakeholders' perspectives on quality into the overall concept of software quality. A comprehensive model of stakeholders' perspectives on software quality has not been developed. This research posits that one's emotional formation, technology-related knowledge, and past experiences determine one's software quality assessment strategy. This study proposes the investigation of this complex of factors generating a dichotomy of stakeholders' perspectives and proposes a methodology for software quality assessment.

The importance of software quality has been highlighted by Broy in 2004: "software quality is a crucial issue in a society that vitally depends on software systems" (Broy, Deißenböck and Pizka 2004). In spite of this fact, software products being released to market show a large number of weaknesses (Kuhn, Wallace and Gallo 2004; Arora, Caulkins and Telang 2005; Newsham, Palmer, Stamos and Burns 2007). Research associates this fact with an increasing complexity of software systems (Offutt 2002). Another explanation indicates that the tendency of releasing a less functional software product earlier (first to market) (Bayus 1997) is justified by the limited amount of resources required later for fixing it (Arora et al. 2005). Although it has been shown that loss of software quality has a negative impact on the market and on its users (Choudhary 2007), Banker and Slaughter validate the economical viability of a "first to market" approach (Banker and Slaughter 1997). The question this raises is; *are there better development methodologies that software development organizations can employ in order to improve the perceived quality of their products and services while using the "first to market" strategy?*

THEORETICAL BACKGROUND

As described by a wide range of researchers, quality is a measure of conformance. Software quality has been defined as either conformance to specifications, or conformance to users' needs and expectations. Researchers traditionally defined software quality through characteristics of the software development processes (Nord and Tomayko 2006), through characteristics of the software product itself (Plosch, Gruber, Hentschel and Korner 2007), or through attributes emphasized by the users of the software product (Feigenbaum 1983; Ishikawa 1985). These last three perspectives form the basis for a large number of generally accepted software quality philosophies and quantitative models (Berander, Damm and Eriksson 2005). In spite of this, there is still a need for an articulated view delineating the role users play in assessing software quality. The perceptions of quality and individual assessment criteria the users have were never formalized into a quality assessment methodology.

Presently, practitioners assess quality based on the two standards the International Organization for Standardization published. ISO/IEC 9126 ((ISO) 2001) has a four part structure that addresses the following areas: process quality (quality of the development process), internal quality (quality of the source code), external quality (quality delivered by the software product, quality of execution), and quality in use (to which extent the user needs are met in the user's working environment). ISO 25000 includes requirements specification into the quality evaluation process (Zubrow 2004).

New perspectives on software as a product and on software development as a process bring to attention new factors that complement the ISO standards. According to Armour, software is more a storage medium than a product, while software development process is more a knowledge generation process than a product manufacturing process (Armour 2000b, a). Acknowledging Armour's perspective, one has to accept that the general definition of software product quality provided by the ISO standards should be complemented with a human-centric view in which personal traits play a significant role.

The theory of emotions (Frijda 1986) can inform the development of the software quality assessment methodology. The emotion process model (Frijda 1996, 2007) centers on the individual and describes the instantaneous emergence and transformation of emotion components. Individual behaviors are exhibited as a response to a set of individual emotions. Individual emotions are shaped by one's personal reaction to the results of appraising a current condition or event.

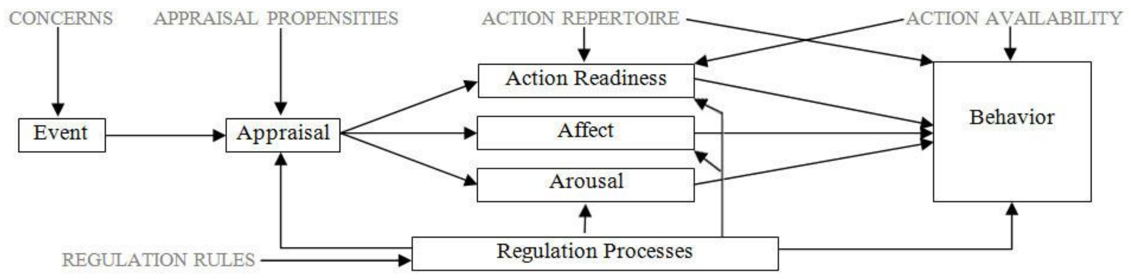

Figure 1: The Emotion Process and Its Condition (Frijda 1996)

In the context of software quality assessment, *current condition* is determined by an individuals' exposure to a software artifact. *Appraising* refers to "the information processes that link perception of an event to emotional meaning" (Frijda and Mesquita 1998). During the appraisal process, one will consider the potential use of the software artifact as a means of addressing personal concerns. The intensity of emotions elicited during the appraisal process is correlated with the strength and the importance of the individual *concerns* addressed by the condition being appraised. *Affect*, *arousal*, and *action readiness* refer to an individual's emotional processes generating the preconditions for a behavior. *Regulation processes* refer to the attenuation or enhancement of emotion because of anticipated effects. The two types of regulation processes, internal (from inner subject), and external (from the environment) have a significant influence on one's quality assessment (Frijda 1986; Frijda and Mesquita 1998). Internal regulation processes can be personal preferences, individual norms and standards, or pre-determined reactions formed as a consequence of previous experiences or acquired knowledge. One's past experiences might have a unidirectional influence on one's emotion processes. If the person has knowledge in a relevant area then the strength and direction of person's emotions will be influenced by the match between the artifact's expected and perceived behavior. External regulation processes include societal, organizational, or group norms and standards. An individual might be externally constrained to perceive an artifact as not being beneficial or efficient. Some people believe that Microsoft products are unreliable. A person sharing this belief can assign estimates of quality before confirming such assessment. *Behavior* includes intentional behavior that is motivated by action tendency. People with different experiences, knowledge, and understanding of software systems develop different sets of emotions and assess the quality of software differently.

The expectation confirmation theory (Lin, Wu and Tsai 2005) has also relevance for the development of a software quality assessment methodology. This theory states that expectations and perceived performance, mediated through disconfirmation, lead to satisfaction. One's expectations are determined by one's past experiences with similar artifacts, knowledge in the area, and shared beliefs. One's evaluation of perceived performance is largely guided by one's expectations. When there is a mismatch between the expected and exhibited level of performance, disconfirmation is high and has a negative impact on one's satisfaction with the artifact. Conversely, when there is a match, one's satisfaction is enhanced and leads to the intention of using the software artifact. One's background in the area of software use and development can have a significant impact on one's expectations, which in turn leads to one's satisfaction with a software artifact. Ultimately, one's experience and knowledge determines both the decision to use an artifact or not, and the quality one associates with an artifact.

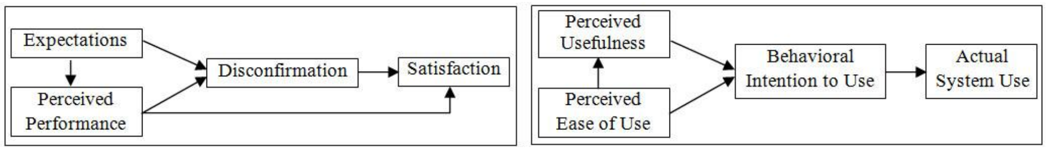

Figure 2. a) The Expectation Confirmation Theory (source:(Lin et al. 2005)); b) TAM (source: (Davis 1989))

Another relevant theory is the technology acceptance model (TAM) (Davis 1989). The technology acceptance model posits that a person's intention to use a software artifact is determined by the perceived usefulness and the perceived ease of use

(Venkatesh 2003). An individual's perceptions of usefulness and ease of use are significantly determined by that individual's past experiences and knowledge in areas relevant to the software system. A data mining expert might decide not to use a software tool based on two factors: first, the data mining expert possesses the knowledge required to manually perform the tasks at hand; second, the data mining expert recalls a negative experience from the past dealing with a similar software tool. Our data mining expert might decide against the use of a software artifact largely due to past experiences and available knowledge.

A DICHOTOMOUS STAKEHOLDER-CENTRIC SOFTWARE EVALUATION FRAMEWORK

A dichotomous classification of software stakeholder perspectives

Peoples' reactions to events and situations they face are based on personal traits. A person's past experiences can also influence that person's emotions associated with the appraisal of a current context. That person can have his or her decisions regarding aspects of technology influenced by the emotions generated as a direct result of the past experiences. Therefore, past experiences can become a determinant factor in choosing a specific course of action in a given situation. Another factor with a significant influence on one's decision process is the knowledge relevant to the decision process that one possesses. If a web developer has to choose what web browser to install on a small organization's network of computers, then he or she will use the knowledge acquired over time for deciding which web browser would provide most benefits while exposing organization's computers to less amount of risk. This is not the case of a user with limited web development knowledge. This user will build his or her opinion largely on interface and functionality related aspects of the website, with interface-related details being predominant. Therefore, people with advanced technical knowledge and experiences tend to use more evaluation criteria when assessing the quality of a product or service. While the quality assessment approaches adopted by people with advanced technical knowledge and experiences and those lacking this background are different, they also overlap in few respects. Previous research focused on perceptions of quality instead of assessment approaches, and thus failed to find significant differences (Jung, Kim and Chung 2004). All software users, regardless of background, agree that ease of use is a desired property of software. In spite of this overlapping, the two perspectives build on distinct approaches and constitute the basis of our dichotomous classification of software stakeholders into technical and non-technical software stakeholders.

The dichotomy of approaches to quality proposed in this study can be further detailed and justified if we group common quality factors into 7 sets of criteria: (1) general measures of quality, (2) complementary services and components, (3) individual preferences, (4) subjective measures, (5) front-end specific measures, (6) back-end specific measures, and (7) features. These sets are described in the next paragraphs. First 2 sets include general quality factors commonly shared by all software users. Next 2 sets include factors which are more specific to non-technical users. Last 3 sets include factors only a technical user would completely understand. They are commonly used by technical users in their assessment of quality.

The common elements of the two approaches are captured in the general measures of quality and in the complementary services and components. The later includes characteristics of documentation availability and completeness, maintenance support, and customer service support. While the two approaches might differ in the way they analyze documentation, the difference is normally not significant. General measures of quality include measures that are generally accepted by all software stakeholders as indicators of quality. Such measures are performance (e.g. speed), efficiency (e.g. level of computer resources utilization), accuracy (e.g. degree of answer correctness), functionality (e.g. customization, solution finding algorithm, user-system interaction dynamics), reliability (e.g. error ratio in a given context and over a determined period of time), robustness (e.g. ability to recover from unexpected situations or improper functionality parameters), complexity (e.g. user perceived level of complicatedness), and other similar generally accepted measures of quality.

Subjective measures of quality, which are part of the non-technical software stakeholder approach, might include attachment (e.g. affection towards specific characteristics of a software artifact), or familiarity (e.g. knowledgeability of a software system's functionality and use due to previous experiences with similar artifacts). Last of the four sets of criteria a regular software stakeholder uses for assessing quality is represented by individual preferences. They are split up into functionality-related and presentation-related preferences. Functionality-related individual preferences might refer to one's liking of drop-down menus as opposed to pop-up options, or selection lists, or wizard-type selection when facing multiple options. Presentation preferences are centered on various aesthetic elements (e.g. graphical elements used to highlight frames in a multi-frame interface, or similar non-functional graphical elements), choice of colors or their combination in a color scheme, or structure and organization of the graphical user interface (e.g. grouping of similar elements of a GUI in the same area of the interface, or consistent look and feel). Software systems can include a user customization feature which might address some of the functionality-related and presentation-related preferences.

Technical software stakeholder perspective includes the two sets of quality measures shared with the regular software stakeholder perspective and three additional sets of measures. First of the additional sets refers to the features the software system provides. Here one can assess the customization level of the software system (e.g. the extent to which the software system is able to offer customization options), completeness (e.g. the extent to which the software system properly addresses the complete set of user needs and expectations), flexibility (e.g. ability to adapt to various types of functionality requirements and sets of inputs), modularity (e.g. the organization of the software system into a structure of inter-dependent and communicating modules), and other software features similar in nature. Last two additional sets relate to front-end and back-end elements of a software system. Front-end measures of quality are related to the user interface. A person with advanced knowledge related to standards of GUI design can use Miller's law as a way of confirming or disconfirming a software system's quality. This is not the case for a regular user lacking such knowledge. Back-end measures of quality address characteristics relating to source code (e.g. indentation, comments, clarity, structure and organization, choice of programming language, size), architecture (e.g. organization of functional units of code, choice of database support), database design and data communication (e.g. structure and organization of data in a database, data communication architecture and protocols), software development life-cycle (e.g. development processes and methodologies employed).

A dichotomous software evaluation framework

In ISO 25000, the "product quality measurement reference model" is presented in order to explain the steps along a product's lifecycle at which specific quality measures can be performed (software development process, internal quality, external quality, quality in use). Requirements specification is implicitly considered to be part of the process quality step in the ISO 25000 quality model. Given that software products meet the specifics of knowledge mediums and since the software development activity matches the unique characteristics of a knowledge creation process, as indicated by Armour, I propose the explicit representation of requirements specification in the measurement reference model.

The ISO 25000 standard labels quality measures that are perceived by a software system's users as "quality in use." One's perception of quality is not limited strictly to characteristic directly derived from the software system itself. Often, one considers the availability of maintenance options or support as an indication of quality, despite these not being integral part of the software system. As a consequence of this, I propose to specifically mention the significance of complementary services and components by adding them as an additional component to the product quality model. The two enhancements proposed are included in the product quality model presented in Figure 3. The ability of those people having an advanced technical background to look "deeper" into the characteristics of a software system enables them to exhibit a clearly distinct approach to software quality assessment. This approach cannot be duplicated by the common users of software. People with advanced technical backgrounds commonly include all six areas of the product quality model into their assessment of quality. In contrast to the advanced user of software, common users will include only three areas in their software quality evaluation: complementary services and components, quality in use, and (less often) external quality. Different areas have different weights on the individual overall assessment of quality. Therefore, the influence of these six areas on an overall quality score should be weighted in order to more accurately represent people's real perceptions of quality.

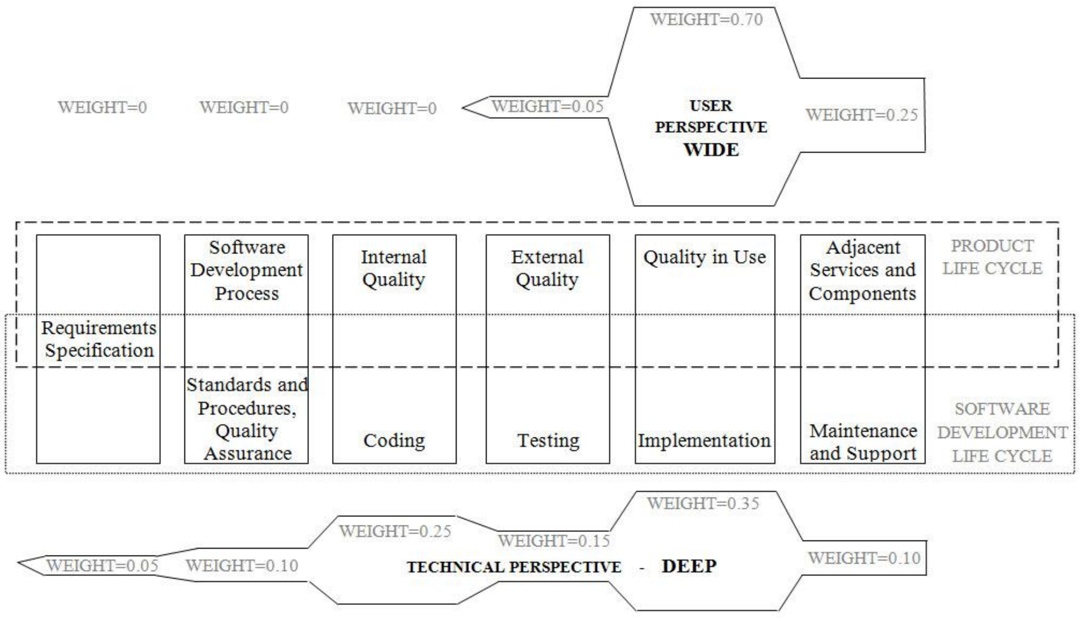

Figure 3. A Dichotomous Software Evaluation Framework (adapted from ((ISO) 2001; Zubrow 2004))

In this research-in-progress paper I propose a framework for computing "stakeholder quality scores" corresponding to technical and to non-technical perspectives on quality. In this framework I propose a number of weights corresponding to the importance of each quality area in the perception of quality for each perspective. Subsequent studies should attempt to refine the proposed weights based on the results of empirical studies involving large populations of technical and non-technical users of software systems. Technical users of software distribute the assessment of quality over all six areas of the software development lifecycle. In contrast to the technical users, the non-technical users of software distribute their assessment of quality over only three areas of the product lifecycle. As a consequence of this, the non-technical perspective on quality can be labeled as "wide" while the technical perspective on quality can be labeled as "deep." The dichotomous software evaluation framework is presented in Figure 3.

EVALUATION

The evaluation of the proposed framework has to be rigorously demonstrated through well-executed evaluation methods (Hevner, March and Park 2004). The dependent variables are represented in this research-in-progress by the software quality assessment scores. A comprehensive empirical study can compute the quality scores corresponding to the two perspectives described in the dichotomous classification of software stakeholders. It can also propose a method of combining the two scores into one overall quality score. A number of the independent variables in this study will need to be operationalized according to the established literature. Other independent variables, such as source code or development life-cycle will require the development of an appropriate measuring process.

An experiment can be conducted for the evaluation and validation of the proposed framework. The subjects will be selected based on their responses to a selection survey that will gather information on their technical background and knowledge. Additional selection of subjects will be performed at the beginning by conducting a workshop designed to test subjects' technical knowledge and abilities. Next, the subjects will be asked to assess the quality of a number of software artifacts. The main experiment will require the subjects to use the provided software artifacts in order to perform a series of small tasks. Next, the subjects will be asked to assess the quality of the provided software artifacts again. A questionnaire will be administered to determine the criteria used by subjects for assessing software quality, and their relative importance.

Hevner proposed in 2004 seven guidelines for design science research. This study is evaluated against those seven guidelines as follows. Design as an artifact – The proposed dichotomous stakeholder-centric software evaluation framework includes a software quality assessment methodology. They correspond to Hevner's definition of an artifact (Hevner et al. 2004). Problem relevance – Modern societies are increasingly dependent on computer systems. In this context, the quality of software systems becomes increasingly important to the efficient and reliable functioning of institutions, organizations, and individuals (Broy et al. 2004). Design evaluation – The evaluation of the proposed artifact will be conducted against the established metrics for software quality. The results will be compared against other results on software quality from the academic literature in the field of software quality. Research contribution – The dichotomous stakeholder-centric software evaluation framework is the main contribution to research. It represents a "solution to heretofore unsolved problem" (Hevner et al. 2004). The framework provides researchers with a better understanding of consumers' perspectives on software quality. Software development organizations could design development methodologies that focus on the areas perceived by consumers as essential to defining product quality. Research rigor – The design of the proposed framework follows principles established in the academic literature. The evaluation of the framework is based on established metrics and accepted operationalizations of the variables included. Design as a search process – Vaishnavi and Kuechler proposed a general design cycle which emphasized the importance of an iterative circumscription process (Kuhn et al. 2004). This research follows these guidelines. Communication of research – The results of this study will be communicated with researchers in the fields of information systems and computer science through conferences and journals.

CONCLUSION

The study offers a dichotomous classification of software stakeholders that provides a new way of approaching the topic of software quality assessment. The results of this research have the potential of changing the way software development organizations design and manage software projects. The contributions to both research and practice are significant and provide opportunities for new and interesting research projects.

REFERENCES

1. (ISO), I. O. f. S. (2001). ISO/IEC 9126-1: Software engineering-product quality-part 1: Quality model. Geneva, Switzerland, International Organization for Standardization.

2. Armour, P. G. (2000a). "The Case for a New Business Model - Is software a product or a medium?" Communications of the ACM **43**(8): 19-22.

3. Armour, P. G. (2000b). "The Five Orders of Ignorance." Communications of the ACM **43**(10): 17-20.

4. Arora, A., et al. (2005). "Sell First, Fix Later: Impact of Patching on Software Quality." Management Science.

5. Banker, R. and S. Slaughter (1997). "A Field Study of Scale Economies in Software Maintenance." Management Science **43**(12): 1709-1725.

6. Bayus, B. (1997). "Speed-to-market and new product performance trade-offs." Journal of Product Innovation Management **14**: 485-497.

7. Berander, P., et al. (2005). Software quality attributes and trade-offs, Blekinge Institute of Technology.

8. Broy, M., et al. (2004). A Holistic Approach to Software Quality at Work. World Congress for Software Quality.

9. Choudhary, V. (2007). "Software as a service: Implications for investment in software development." Journal of Management Information Systems **24**(2): 141-165.

10. Davis, F. D. (1989). "Perceived usefulness, perceived ease of use, and user acceptance of information technology." MIS Quarterly **13**(3): 319-339.

11. Feigenbaum, A. V. (1983). Total quality control, McGraw-Hill.

12. Frijda, N. H. (1986). The Emotions. New York, Cambridge University Press.

13. Frijda, N. H. (1996). Passions: Emotion and Socially Consequential Behavior. Emotion: Interdisciplinary Perspectives. R. D. Kavanaugh, B. Zimmerberg and S. Fein. (Eds.), Mahwah: Lawrence Erlbaum**:** 1-27.

14. Frijda, N. H. (2007). The Laws of Emotion, Mahwah: Lawrence Erlbaum Associates.

15. Frijda, N. H. and B. Mesquita (1998). The Analysis of Emotions: Dimensions of Variation. What Develops in Emotional Development? M. Mascolo and S. Griffin. (Eds.) New York, Plenum Press**:** 273-295.

16. Hevner, A., et al. (2004). "Design Science in Information Systems Research." MIS Quarterly **28**(1): 75-105.

17. Ishikawa, K. (1985). What is total quality control? : the Japanese way, Prentice-Hall.

18. Jung, H., et al. (2004). "Measuring software product quality: A survey of iso/iec 9126." IEEE Software **21**(5): 88-92.

19. Kuhn, R., et al. (2004). "Software Fault Interactions and Implications for Software Testing." IEEE Transactions on Software Engineering **30**(6).

20. Lin, C. S., et al. (2005). "Integrating perceived playfulness into expectation-confirmation model for web portal context." Information & Management **42**(5): 683-693.

21. Newsham, T., et al. (2007). Forensics software: Weaknesses in critical evidence collection. Proceedings of the 2007 Black Hat Conference.

22. Nord, R. L. and J. E. Tomayko (2006). "Software Architecture-Centric Methods and Agile Development." IEEE Software **23**(2): 47-54.

23. Offutt, J. (2002). "Quality Attributes of Web Software Applications." IEEE Software **19**(2): 25-32.

24. Plosch, R., et al. (2007). The EMISQ Method - Expert Based Evaluation of Internal Software Quality. Proceedings of 3rd IEEE Systems and Software Week, Baltimore, IEEE Computer Society Press.

25. Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D (2003). "User acceptance of information technology: Toward a unified view." MIS Quarterly **27**(3): 425-478.

26. Wong, B. (2002). The Appropriateness of Gutman's Means-End Chain Model in Software Evaluation. Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02).

27. Zubrow, D. (2004). Measuring Software Product Quality: the ISO 25000 Series and CMMI, Carnegie Mellon University - Software Engineering Institute.