Chapter 3
A Review of Software Quality Methodologies

Saqib Saeed
University of Siegen, Germany

Farrukh Masood Khawaja
Ericsson Telekommunikation GmbH, Germany

Zaigham Mahmood
University of Derby, UK

ABSTRACT

Pervasive systems and increased reliance on embedded systems require that the underlying software is properly tested and has in-built high quality. The approaches often adopted to realize software systems have inherent weaknesses that have resulted in less robust software applications. The requirement of reliable software suggests that quality needs to be instilled at all stages of a software development paradigms, especially at the testing stages of the development cycle ensuring that quality attributes and parameters are taken into account when designing and developing software. In this respect, numerous tools, techniques, and methodologies have also been proposed. In this chapter, the authors present and review different methodologies employed to improve the software quality during the software development lifecycle.

INTRODUCTION

The abstract nature of software products make them very different from conventional products that we can touch and see being built. As a result, the probability of human induced errors in software systems becomes high. Software systems have revolutionized every field of life. It is hard to believe that software systems are not being applied in any organizational setting. Extensive use of software applications and products in everyday life requires dependable and efficient software applications to run the day to day activities smoothly. This is equally true in case of business organization and commercial environments. As a result, similar to other products, the quality is emerging as an important attribute of such software systems. Crosby (1979) defines quality as "conformance to requirements" and Pressman (2010) extends this definition to include "adherence to standards". Pressman (2010) defines

DOI: 10.4018/978-1-4666-4301-7.ch003

software quality as "conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software." This definition not only focuses on functional requirements but also includes performance requirements, adherence to development standards and presence of generic professional quality attributes to determine the quality level of software artifact. However, the definition of software quality is a contested concept in the software engineering literature (cf. Kitchenham & Pfleeger, 1996; Petrasch, 1999). The suggestion is that, in defining software quality, user requirements and elicitation of such requirements must form an important aspect of quality. An important reason behind this is that, in contrast to the conventional products, elicitation of software requirements is regarded as the responsibility of the development team instead of the customers. As a result, the requirements are categorized in different types e.g. user requirements, functional requirement, non functional requirements, usability requirements, performance requirements. The additional difficulty is that these requirements may well conflict with each other; they may even have different priority levels among stakeholders (developer, user, financer etc.) depending on stakeholders' perspectives or the application domain. It is also quite possible that it may be difficult to achieve some of these requirements.

IEEE (1990) defines software quality as "the degree to which a system, component, or process meets customer or user needs or expectations." This definition only focuses on the conventional marketing principle that customer is right and quality is only measured based on the satisfaction level of customer needs. On the other hand, other quality parameters which are invisible to users, but extremely important, get neglected. It is not easy to quantify quality. It is, therefore, often assessed in terms of its characteristics. Ghezzi et al., (2003) describe following eleven characteristics of software quality:

- Correctness
- Reliability
- Robustness
- Performance
- Usability
- Verifiability
- Maintainability
- Reusability
- Portability
- Understandability
- Interoperability

Furthermore quality is also determined by the process employed to design the product, product characteristics and the project attributes in terms of life cycle stages. Every application area also has its own specialized quality requirements which need to be considered while designing quality objectives of a software project. This highlights the fact that there is no strict definition of quality as each software development project and product has its own quality goals. In this chapter, we highlight how the quality can be improved in each software lifecycle phase and also suggest methodologies that can be put in place to improve the software development projects.

QUALITY IN SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

The importance of enhanced quality suggests that quality is not something that is added to a product after it has been realized (such as garnishing a dish that has already been cooked). It requires a quality culture where the raw materials and the processes involved have built-in quality. In such a culture, enhancement of quality becomes everyone's responsibility. Thus, it becomes an ongoing activity performed throughout the software development lifecycle, through all the various software development phases; requirements engineering, system design, development and testing. As previously mentioned, software quality can also be correctly