On Way to Acquiring Reliability Growth in Software Systems

Javaid Iqbal
Assistant Professor,
P.G Department of Computer Science
University of Kashmir- North Campus, India

Dr. S.M.K.Quadri
Head,
P.G Department of Computer Science
University of Kashmir- Hazratbal Campus, India

Tariq Rasool
Lecturer,
P.G Department of Computer science
Islamic University of Science and Technology, India

ABSTRACT
Reliability of a software system has been one of the driving forces for the various software engineering processes and methodologies that led to their evolution and sophistication. The concerns for reliability of a software system surface very early on during the development phases of the software system. When it comes to acquisition of reliability, we should not immediately get model-oriented; instead every minutia of software development life cycle should be given its due.

This paper outlines the areas where reliability needs to pick up. This paper underlines the fact that it is only through carefully controlled and carefully applied software engineering process that software reliability growth can be achieved. It emphasizes on the acquisition of reliability of software systems as soon as the conception phase. We trace the reliability concerns from the early stage to the fully functional stage. The terms "hard reliability" and "soft reliability" are used.

General Terms
Software Reliability, Reliability Acquisition, Software Reliability Growth.

Keywords
Reliability, Reliability Acquisition, Software system Reliability Growth, Hard Reliability, Soft Reliability.

1. INTRODUCTION
Owing to the unexpectedly spiraling increase in the size and complexity of software systems during the past few decades, software reliability has become even more increasingly important for such massive systems [2]. Early on, when there was not so much of focus and concern for reliability of software systems, the software were "un-structured" and the so-called "spaghetti" type. However, as the software systems became ubiquitous, they grew in size, and complexity and this led to the development of "structured" and "modular" software systems where the inherent complexity is better coped with. D.L. Parnas and et al discuss in detail the modularization of complex systems in [10]. Moreover, the launch of software systems in highly time-critical and performance-critical applications got the reliability considerations into limelight in the domain of software reliability engineering. The process of acquisition of the reliability growth in any type of software system starts as early as the design/ architecture phase of the development of the software system. During design phase, the designer puts a finger on the "doing" of the software system and thus designs a holistic structure of software, for the intended functions to be carried out as per the blueprint. Starting thinking in terms of the structure of the software system itself implicitly connotes moving towards the reliability of the system. It is here that the first treatment to the inherently existent complexity can be made by an effective structural design. Every software system has to have some threshold reliability level in terms of its design and architecture, and implementation, for it to be called an operational software system which can be deployed for use. However, depending on the target functionalities and service specifications of the software system, the desired levels of reliability may be "hard" for mission-critical or safety-critical software systems and the levels of reliability may be "soft" for other types of software systems where a potential hazard may not be too much catastrophic. In this direction, much has been done to improve upon reliability and much more needs to be done.

Off late, the increasing uses of Commercial Off-The-Shelf Components (COTS), reusable components, and other types of components have added new dimensions to the domain of reliability analysis of software systems. Software engineering started with the traditional function-centered water-fall process model, went on to more sophisticated data-centered object-oriented abstractions and one step further to structure-centered component-based technology which again offers abstractions. While data collection is very important in the development and testing of traditional software reliability models. However, in the case of COTS or Open Source Code, this data may not be readily available, and the reliability of the software module is questionable. Thus the domain of Component-Based software Engineering (CBSE) is an area where reliability analysis has gained focus. Moreover, Simulation-based approaches to software reliability analysis have gone down very well with the software engineering community. Javaid Iqbal and Dr. S.M.K. Quadri discuss in detail the relevance of Simulation for the better acquisition of Software Reliability in [2].

2. SOFTWARE RELIABILITY AND THE NEED FOR ITS ACQUISITION AND GROWTH
Software Reliability is defined as the probability that software will provide failure-free operation in a fixed environment for a fixed interval of time [8]. The failure-free operation in the context of software is interpreted as adherence to its requirements [12]. It is a function of the software faults and its operational profile [9], i.e., the inputs to and the use of the software. Software does not have moving parts and does not

physically wear out as hardware, but is does outlive its usefulness and becomes obsolete [6]. Software reliability marks the hard-sought performance criterion in an evolving software process. Software reliability has been shaping the thought-process in the face of massively-complex software systems engineering. In such software systems, reliability becomes more and more volatile. Different factors contribute to the growth in unreliability of the system. In general, the more the complexity of a software system, the more is the level of unreliability. Thus, highly-complex/least-simple software systems are most unreliable whereas the least-complex/ most-simple software systems are the most reliable. Thus, an exigent need for the timely management of the software system arises in terms of the management of the size, and complexity. With spiraling increase in the complexity of software systems, performance considerations became more significant. Structural specifications need to be very clear. However, even though some level of unreliability does exist for a software system, the dynamic aspect of software quality needs to grow even as the unreliability turns its head on in massively complex systems. This necessitates a growth in reliability of software system as its size and complexity accumulate. Reliability Growth helps measure and predict the improvement of reliability through the testing process. Reliability growth also represents the reliability or failure rate of a system as a function of time or the number of test cases. The practice of building massively-complex software structures would be undoing the "build process", if there is no corresponding growth in reliability of the software system built.

3. MEASURING RELIABILITY

Reliability measures only the probability of failure. There are different metrics which facilitate the measurement of the quality characteristic reliability.Reliability can be measured in terms of fault rates either in the form of the frequency of faults, or in the form of a time-distribution of faults. Another way is to measure the Mean Time between Failures (MTBF), where the average time between software system faults is calculated. Mean Time to Repair (MTTR) measures the average time between a fault occurrence, and the restoration of the system to normal operation and thus measures the maintainability of the system [12].

MTBF = MTTF + MTTR

Where MTTF is the mean time to failure and is a measure of how long software is expected to operate properly before a failure occurs.

4. THE ROAD TO RELIABILITY

When it comes to acquisition of reliability, we should not immediately get model-oriented; instead every minutia of software development life cycle should be given its due**.** Though, it may appear that reliability considerations are born only after the birth of software system. In fact, the road to reliability starts as soon as the development of the software system starts. It may be noted that the software that is properly engineered takes the road through a well-structured process from requirements specification and design, via detailed specifications, to actual implementation.Thus, reliability is an "inclusive" attribute throughout the lifecycle of the software system. Against the backdrop of the introductory section of this paper, it is obvious that the reliability of a software system depends on the design methodology adopted, the nature of the software system vis-à-vis the hardness/softness of operation and the individual reliability of components and their cumulative reliability result as well. The design considerations may include topology, dependency degrees etc. Unlike hardware, software takes a different fault identification rate with the rate peaking at integration and testing. Rigorous removal of faults takes place during testing and continues on a sluggish side during the operational settings of the software, as the software heads towards a better reliability level. Operational characteristics may indicate the levels of reliability achieved. Moreover, the environmental settings also govern the reliability of the software system in or other way. To acquire a certain level of reliability, an accurate software system design needs to be in place and the behavioral specifications need to be understood.

A modular design helps contain the complexity of the software by way of increasing the "cohesion" of the modules while reducing the "coupling" between the modules. Today's software engineering standards call for software to be organized in accordance with a principle known variously as "Information Hiding," "Object-Oriented Programming", "Separation of Concerns," "Encapsulation," " Data Abstraction," etc. This principle is designed to increase the "cohesion" of the modules while reducing the "coupling" between modules [11].

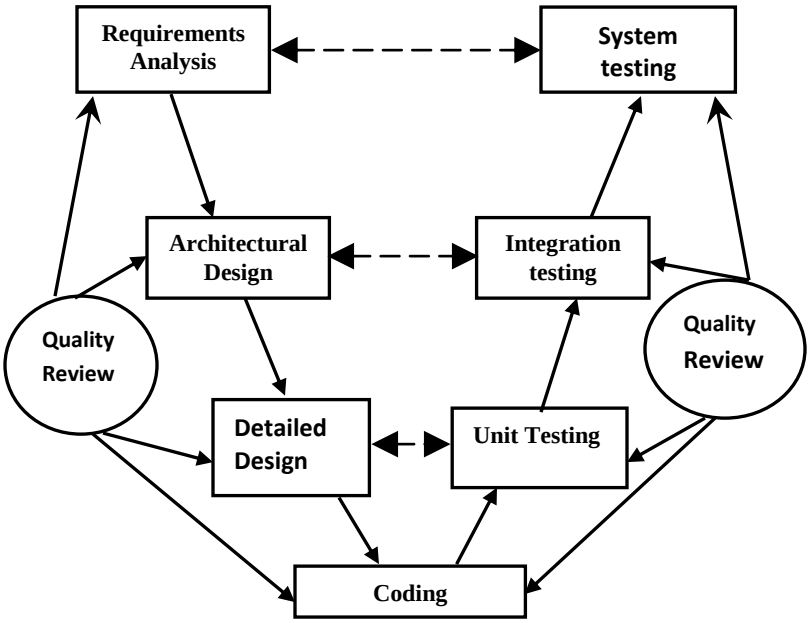

Figure 1: V-model of Software engineering

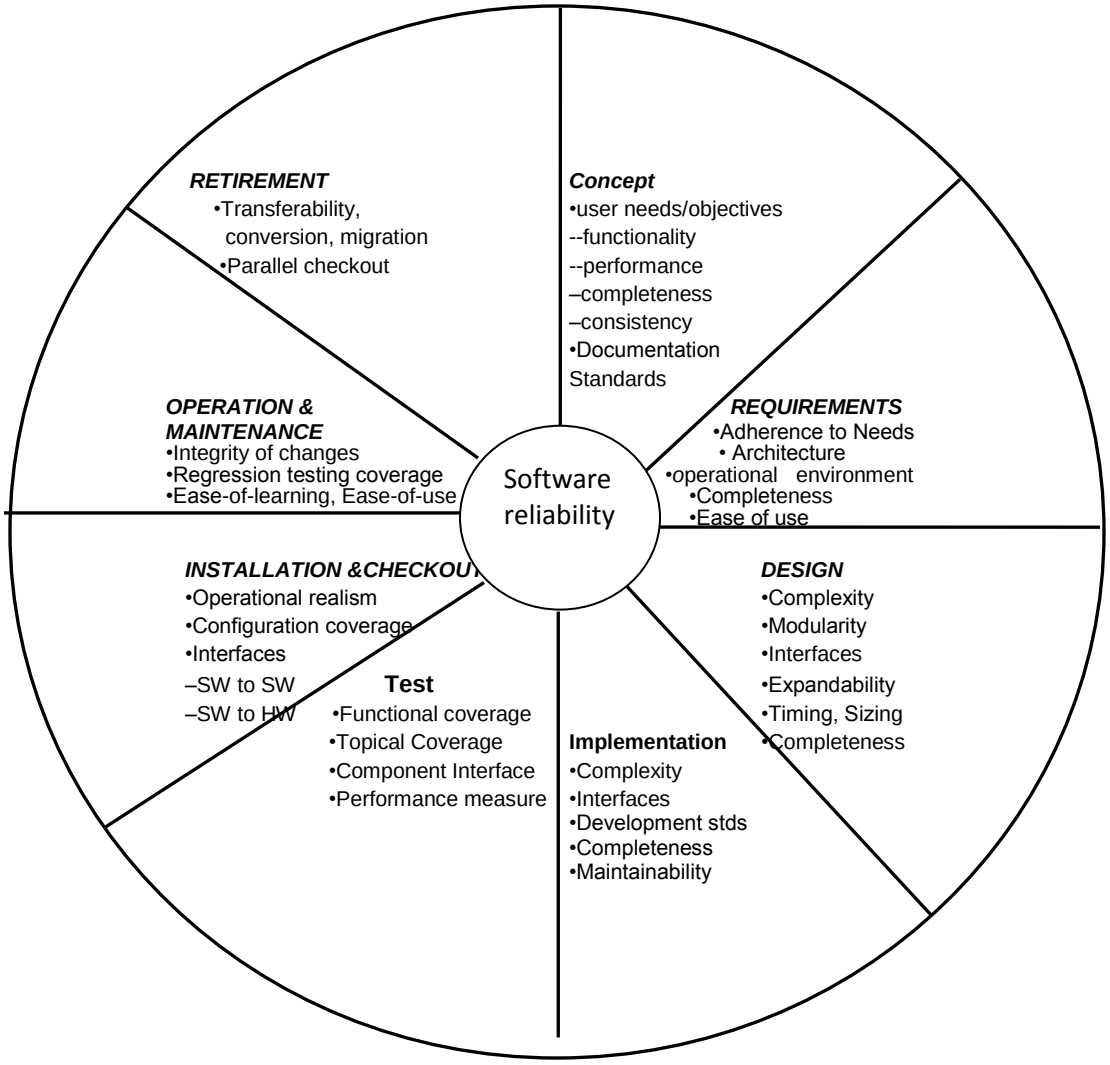

Figure 2: Quality Factors Impacting Reliability [4], [7]

5. WHAT HELPS ACQUISITION OF RELIABILITY

As per IEEE[3], [5]: "Software engineering(SE) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software." Another definition is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines [1]. The goal of SE is to produce quality software in a timely, cost effective manner. Against the backdrop of these definitions, we make the following statement for successful acquisition of reliability. The successful acquisition of reliability is assessed in terms of the objective of the system to be designed, the underlying assumptions made, the requirements design, the requirements specifications, the architectural design specification, the specification of the design methodology adopted, and the use of other benchmark and fitting software engineering methodologies. It should be clear that the road to the acquisition of reliability starts with the process of software product development. Though reliability measures are typically obtained from testing data, in case of COTS, the better way of obtaining reliability measures could be from the information supplied by its supplier. The increased modularization along with component engineering (development and testing) and then the integration testing help development and maintenance of highly reliable software economically, within stipulated time-schedule. IEEE Std 982.2-1988 includes the diagram in Figure 2, which indicates the relationship of reliability to the different life cycle phases [4], [7]. Figure 1 shows the V-model of software engineering.

As is evident from the diagrams in Figure 1and Figure 2, the quality attributes permeate the software life cycle from its conception, via various stages of gestation period, till its retirement. This also signifies the relevance of reliability considerations during the early phases of the software

development life cycle (SDLC). However, the pertinent attributes to a particular phase must be properly identified and assessed as per the time-line of the development process. It may be noted that many errors introduced/un-prevented at the requirement specification level may not show up until the final software product is ready and hence faulty. The role of these quality attributes becomes more pronounced in case of "hard reliability software systems". A coherent focus for acquisition of hard reliability must especially go to requirements, design, implementation and test phases of the SDLC. However, the key to success would be how comprehensively requirements are analyzed and, of course, tested. The quality assurance activities ensure that during SDLC, the statement "the right thing at the right time in the right way" [12] is observed strictly, with the fundamental focus on the planning activity and the compliance to them.

6. CONCLUSION

Acquisition of reliability and its growth in a software system must not be thought of a process attached exclusively to the debugging phase. In fact, the reliability considerations permeate the entire software development process. Any decision in any phase of the life cycle may have a bearing on the reliability of the software. In fact, the broader domain of quality aspect which includes the reliability element also, is a companion to all the phases of software life cycle. They have to go hand-in-hand for the assurance of quality. As a cautionary note, it would be a naivety to incorporate reliability consideration after the birth of software system or after some particular stage of its gestation period.

7. REFERENCES

[1] Bauer, F.L., Software Engineering, Information Processing, 71, 1972.

[2] Iqbal, J., and Quadri, S.M.K.,"Software Reliability Simulation: Process, Approaches and Methodology". Global Journal of Computer Science and Technology, volume 11, issue 8, May 2011.

[3] IEEE Standards Collection: Software Engineering, IEEE Standard 610.12-1990,IEEE, 1993.

[4] IEEE Standard 982.2-1987 Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software.

[5] IEEE STD 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE Computer Society, 1990.

[6] Kitchenham, Barbara, Pfleeger, Shari Lawrence, "Software Quality: The Elusive Target". IEEE Software 13, 1 (January 1996) 12-21.

[7] Rosenberg, L., Hammer, T., and Shaw, J. "Software Metrics and Reliability". 9th International Symposium on Software Reliability Engineering, Germany. November 1998.

[8] Musa, J.D., Iannino, A., and Okumoto, K., Software Reliability—Measurement, Prediction, Application. New York: McGraw Hill, 1987.

[9] Musa, J.D. "Operational Profiles in Software-Reliability Engineering," IEEE Software, vol. 10, no. 2, pp. 14-32, Mar. 1993.

[10] Parnas, D. L., Clements, P.C and Weiss, D.M.(1985) "Modular Structure of complex Systems". IEEE Transactions on Software Engineering, 11, pp 259-266.

[11] Parnas, D. L., Schouwen, J., and Kwan, S., "Evaluation of Safety-Critical software". Communications of ACM, June 1990.

[12] Pressman, Roger S., "Software Engineering: A Practitioner's Approach". The McGraw-Hill Companies, Inc., 1997.