Software Reliability and Quality Assurance

Reliability of a software product is an important concern for most users. Users not only want highly reliable products, but also want quantitative estimation of the reliability of a product before taking a buying decision. However, it is very difficult to accurately measure the reliability of any software product. One of the main problems in quantitative measurement of reliability of a software product is that there are no good metrics available, using which we can quantify the reliability of a product. Besides this problem, there are several other problems which arise when we want to accurately measure the reliability of a software product. Even though no known ways exist to accurately quantify the reliability of a software product, we will nonetheless discuss some of the available metrics for quantifying the reliability of a software product. We will also discuss how we can model the reliability growth and predict the reliability of a software product while the testing phase of the product is still going on. In this chapter, we will also examine software quality assurance aspects of a software product.

Software quality assurance (SQA) is one of the most talked about topics in recent years in software industry circles. The aim of SQA is to help an organization develop high quality software products in a repeatable manner. A *repeatable software development organization* is the one where the software development process is person-independent. In a non-repeatable software development organization, a project becomes successful primarily due to the initiative, effort, brilliance, and enthusiasm displayed by certain individuals. Thus, in a non-repeatable software development organization, the success to a great extent is dependent on who the team members are and hence the successful development of one product does not automatically imply that the next product will also be successful. One of the primary objectives of quality assurance is repeatable software development. We first address software reliability aspects before discussing issues involved in software quality assurance.

9.1 SOFTWARE RELIABILITY

Reliability of a software product essentially denotes its *trustworthiness* or *dependability*. Alternatively, reliability of a software product can also be defined as the probability of the product working "correctly" over a given period of time.

Intuitively we can conclude that a software product having a large number of defects would be very unreliable. It is also clear to us that the reliability of a system would improve if the number of defects in it is reduced. However, there is no simple relationship between the observed system reliability and the number of latent software defects in the system. Removing errors from parts of a software which are rarely used would make little difference to the perceived reliability. It has been experimentally observed by analyzing the behaviour of a large number of programs that 90% of the execution time of a typical program is spent on executing only 10% of the instructions in the program. These *most used* 10% instructions are often called the *core* of the program. The rest 90% of the program statements are called *non-core* and are executed only during 10% of the total execution time. It may not therefore be surprising to note that removing 60% defects from the least used parts of a system would lead to only 3% improvement in reliability. It is thus clear that improvement in the overall reliability of a program due to the correction of a single error would depend on whether the error belonged to the core part or the non-core part of the program.

Apart from the fact that reliability depends on location of the error in the program, it also depends on how the product is used, i.e. its *execution profile*. If we select input data to the system such that only the "correctly" implemented functions are executed, none of the errors will be exposed and the perceived reliability of the product will be high. On the other hand, if we select the input data such that only the functions containing errors are invoked, the perceived reliability of the system will be very low. Different users use a software product in different ways. So the defects which show up for one user, may not show up for another. Therefore, the reliability figure of a software product is clearly observer-dependent and cannot be determined absolutely.

9.1.1 Reliability Metrics

Since different categories of software products have different reliability requirements, it is necessary that the level of reliability be specified in the SRS (software requirements specification) document. In order to be able to do this, we need some metrics to quantitatively express the reliability of a software product. A good reliability measure should be observer-independent, so that different people can agree on the degree of reliability that a system has. For example, there are precise techniques for measuring performance, which would result in obtaining the same performance value irrespective of who is carrying out the performance measurement. However, in practice, it is very difficult to formulate a precise technique for measuring reliability. The next best thing is to have measures that correlate with reliability. For example, the number of well-designed test cases a system process correlates with its correctness. The worst case is to have no measures at all except the subjective judgement. We first define three reliability metrics which can be used to quantify the reliability of software products.

1. **Rate of occurrence of failure (ROCOF).** The ROCOF measures the frequency of occurrence of unexpected behaviours (i.e. failures). The ROCOF measure of a software product can be obtained by observing the behaviour of a software product in operation over a specified time interval and then calculating the total number of failures during this interval.

2. **Mean time to failure (MTTF).** The MTTF is the average time interval between two successive failures, observed over a large number of failures.

3. **Availability.** Availability of a system is a measure of likelihood of the system being available for use over any given period of time. Thus, this metric not only considers the number of failures occurring during a time interval, but also takes into account the repair time (down time) of a system when a failure occurs. This metric is important for systems like telecommunication systems, operating systems, etc. which are supposed to be never down and where repair and restart time are significant and loss of service during that time is important.

All the above reliability metrics are centred around the probability of system failures but take no account of the consequences of failures. In other words, these reliability models take no account of the relative severity of different failures. Failures which are transient and whose consequences are not serious are of little practical importance in the operational use of a software product. These types of failures can at best be minor irritants. On the other hand, more severe types of failures may render the system totally unusable. In order to study the reliability of a software product, it is therefore necessary to classify failures into their various types. A possible classification of failures is as follows:

1. **Transient.** Transient failures occur only for certain input values while invoking a function of the system.

2. **Permanent.** Permanent failures occur for all input values while invoking a function of the system.

3. **Recoverable.** When recoverable failures occur, the system recovers with or without operator intervention.

4. **Unrecoverable.** In unrecoverable failures, the system may need to be restarted.

5. **Cosmetic.** Such failures can cause minor irritants, but do not lead to incorrect results. An example of a cosmetic failure may be the case where the mouse button has to be clicked twice instead of once to invoke a given function through the graphical user interface.