Trends in Reliability Testing

By Stuart Reid

Introduction

Reliability testing is perceived by many to belong in the domain of safety-critical applications, such as fly-by-wire systems, but perhaps surprisingly it is really an aspect of all testing. This paper explains why reliability is the most important quality attribute and how reliability testing can be used on any project.

What is Reliability?

Software reliability definitions are all based on the concept of failure, so first we need to consider how software fails. Initially a developer makes a mistake and this leads to there being a fault in the software. A simple example could be that a decimal point has been misplaced when typing in a numerical value.

Many faults remain hidden forever and never cause a failure. If the 'right' set of conditions coincide, however, to activate the fault then the software fails. Software reliability, then, is the ability of software to perform without failure under specified conditions for a specified period of time (or for a specified number of transactions, for systems that do not run continuously). The term 'specified conditions' here refers to the environment in which the software is to be used and its expected use.

Reliability is Everything!

A popular misconception is that reliability is measured by the time between system crashes. But reliability in fact covers all kinds of failure, and includes not only more subtle functional failures, but also failure to meet non-functional requirements, such as those associated with performance, usability, and security. A web server's reliability will, for instance, necessarily include consideration of response times, number of supported users and scalability, as well as correctness of functionality.

Reliability Testing - The Traditional Approach

Software reliability theory dates back to the early 1970s and differs from its hardware counterpart in that while in hardware the principal cause of failure is physical deterioration, in software there is no such deterioration. The 'ideal' trend in software is that reliability increases as the faults that cause failures are removed. The traditional, and most direct, approach to reliability testing is to use this trend of reliability growth to both estimate current reliability and predict future reliability. This is done by testing the software in conditions as close as possible to those expected to be encountered operationally (both the test environment and the test inputs should

mimic expected use). When tests fail, the corresponding faults are debugged and testing restarts on the now less faulty and more reliable software. For continuously-running software, plotting the time between failures allows the reliability growth curve of the software to be determined, while for transaction-based software plotting the number of failures for a given number of transactions for a sequence of improving versions provides a similar result (see Figure 1). With reliability *growth* modelling, we predict ahead from the current measured value of reliability. This means that the delivered version of the software is normally the last measured version, but with the fault(s) detected in the last test period removed, so that the *predicted* reliability at least satisfies the required reliability.

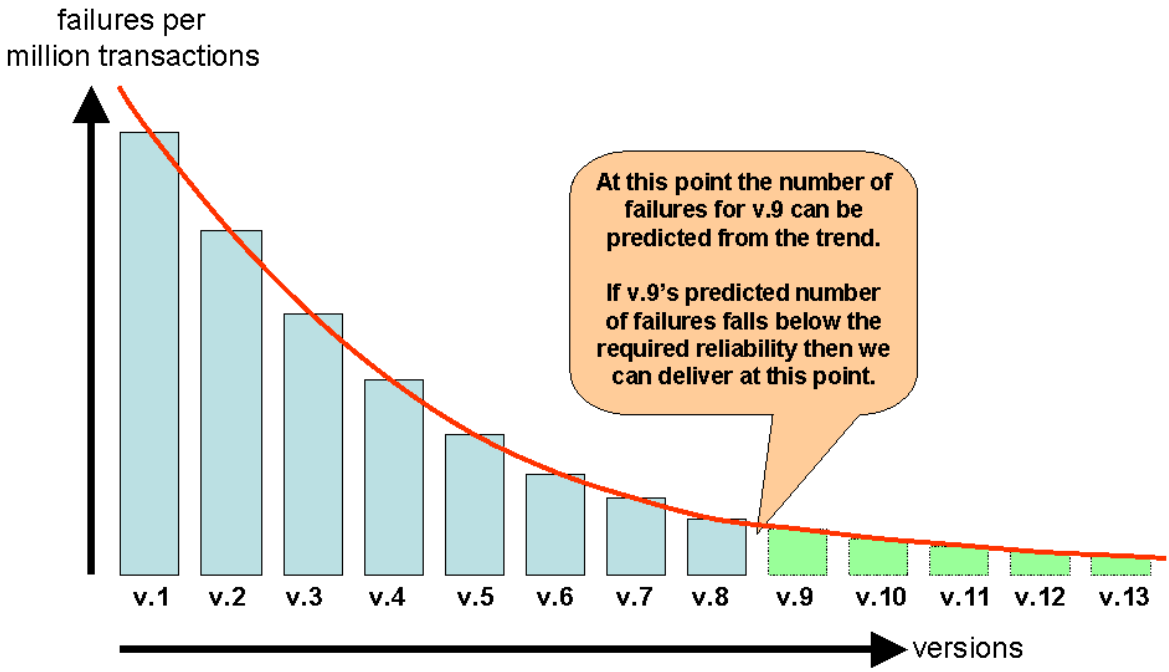

Figure 1

Reliability Growth Models - Difficulties

Reliability testing based on growth models is theoretically sound and its successful use has been reported on many projects. But it suffers from a number of practical difficulties. Surprisingly, given the mandated use of reliability testing for safety-critical applications, reliability growth models cannot, in practice, provide predictions any higher than a mean time to failure (MTTF) of 10^6 hours. This is because to predict that level of reliability requires a previous failure-free test period in the region of half that. Despite this, some airliner avionics software has a required (but unmeasurable) MTTF of 10^9 hours (~114,000 years)!

As each project is unique, its *actual* reliability growth curve is also unique and the reliability tester selects a model from a library that most closely matches the failure data already collected. Given the complexity of the test-debug process being performed, all of these models necessarily include a number of simplifying

assumptions. One typical assumption is that all failures are equally important, even though we know that different failures can have different levels of impact on the system, and also that the same failure may affect diverse users in different ways. Other unrealistic assumptions are that fault removal is always perfectly performed and also that there is a one-to-one mapping between faults and failures.

Not only are reliability growth models based on several unrealistic assumptions, a further major problem with them is in generating test inputs that reflect how the software will be used operationally. These test inputs are generated from operational profiles.

Operational Profiles

The main difference between the tests generated for reliability growth modelling and those for defect testing is that rather than use test case design techniques, such as boundary value analysis or branch testing, an operational profile is used. The operational profile allows test inputs to be generated that mirror those inputs expected from real users. It *should* describe all possible types of user input and provide the relative probability of each type. This allows a test input generation program to generate inputs automatically, normally using a form of random input generator.

But what happens if the operational profile does not reflect actual use of the software? Not surprisingly the value of the reliability prediction then becomes very questionable. It is easy to imagine unexpectedly high use of a particular function in actual operational use that had barely featured in the original operational profile (e.g. text messaging on mobile phones), or vice versa. Also, where there are several classes of users, it may be that each class will use the software differently, leading to very complex profiles.

These and other problems, such as the availability and accuracy of an oracle able to identify non-crashing failures, mean that reliability testing using growth models is only practiced by a small minority of projects. So, should the rest of industry ignore reliability, or is there an alternative…?

Fault-Based Reliability Testing

One alternative is a less direct form of reliability testing - indirect in two ways - it is not based on mimicking expected use and it is based on faults rather than failures. Although reliability is defined in terms of failures occurring in an operational environment, we have seen that the traditional approach to reliability testing is perceived by many as impractical. Also it can only be performed after system testing has reached a stage where the software is relatively stable. The alternative is to derive a measure of reliability from trends in fault detection achieved throughout

system testing. Thus, the system testing that would have been performed anyway doubles as reliability testing.

As test inputs are now not based on an operational profile and because not all faults lead to failures, this means that a fault-based prediction of reliability is necessarily approximate. However, not having to define an operational profile is an obvious benefit and there is also no danger of being misled by a mistaken profile. Also, as we will see, this approach can provide useful information from early in the system test phase and there is little overhead in costs other than the analysis of fault data, which should be collected anyway.

The S-Curve

The 'classic' testing S-curve showing the rate at which software faults are detected during system testing is shown in figure 2. The rate of detection is initially slow due to system test start-up, after which it straightens out as testing progresses, until, hopefully, the majority of faults are found when the rate of detection necessarily slows. The curve should be asymptotic to a line representing the total number of faults in the software. It is the distance from the current point in the project to this line that gives an indication of the number of faults left in the software and thus its current reliability.

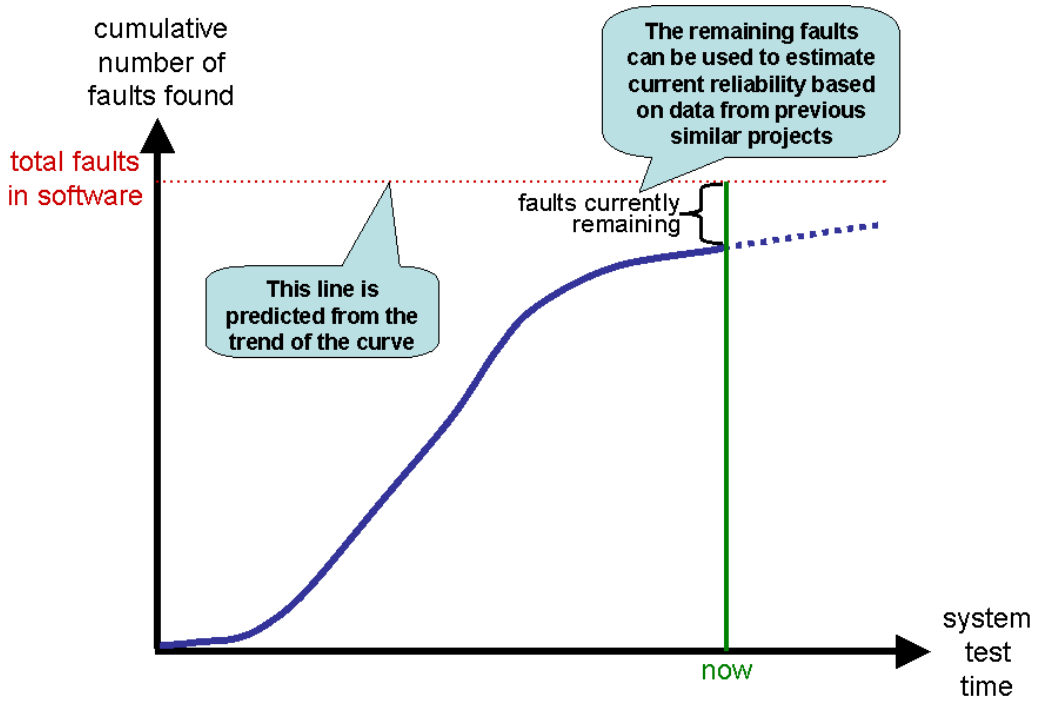

Figure 2

The fault detection trend represented by the S-curve also provides an excellent means of monitoring the progress of system testing. For example, deviations from the

expected curve can be used to trigger corrective actions, although the order in which system testing is performed will partly determine its shape. If the curve is to follow the 'ideal' then all forms of system testing would need to be performed in parallel. Also, all subsystems would need to be tested in parallel. When subsystems are tested sequentially then normally different fault detection rates for the different subsystems will lead to corresponding changes in the curve.

Conclusion

For safety-related software, reliability testing will need to comply with the relevant regulatory standards, which will mean using the traditional reliability growth model approach (even when it cannot measure the required level). But, for all other projects, a fault-based approach can be used at an earlier stage in the life cycle at relatively low cost. Using the fault detection trend during system testing to estimate reliability is not going to guarantee accuracy, but then it won't cost the earth either. It will also ensure that all non-functional aspects that are tested during system testing are considered, which is rare with traditional reliability testing using growth models.