

# Should testing include assessment?

Independent consultant and trainer in risk and management **Felix Redmill**  
proposes an extension to the remit of testers

*A friend used to say that he would never buy anything that didn't fail.* By exaggerating a bit, he got a laugh, but he was not being flip-pant. His point was a serious one. He was speaking to those who got taken in by salesmen's wild claims of perfection. His experience as an engineer gave him good reason not to believe their assertions that their products never failed, or couldn't fail. But his statement was not a moral judgement on their obvious lies; it was a demand for evidence that there was real understanding of a product on which he would become dependent if he bought it.

If you believe that something can't fail, you are unlikely to study how it might fail, or to train technicians to service and repair it, or to spend time preparing and validating trouble-shooting documentation. Trouble-shooting documentation shows (or at least suggests) an understanding of known failure modes and how to deal with them, and the existence of trained service staff demonstrates contingency against others. My friend wanted to be sure that both were in place - to be confident that allowance had been made for both known and unidentified risks.

So far, software testing (and even system testing) is not designed to provide such confidence. It is concerned with finding faults but not with providing information on where failures are most likely to occur in the future. And it is not standard practice for testers to provide an assessment of the quality of the tested software. Could it be otherwise?

Testers get a unique view of software - and of the people who produce it. Before even examining the code itself, they can determine whether and how well it is documented. Does sloppy documentation imply an increased likelihood of bugs? Testers accumulate the data that could determine the degree of correlation. Another thing that testers do - often early and quickly - is to apply complexity-measuring tools to the code. So testers could also provide the data for determining if bug density increases with complexity. But even without empirical proof, intuitively defined good practice is for code to be carefully rather than

sloppily documented, and simple rather than complex. This makes code more easily testable, but it is also thought to influence the number of residual bugs after the testers have done their job and, therefore, the likelihood of failures during operation.

So testers are well placed to identify the attributes that are correlated with the incidence of faults, to investigate them (more or less accurately) in the software on test, and thus to make an assessment of the tested software and to report on their findings. Such an assessment would be useful not only in providing confidence (or the reverse) in the software, but also in encouraging the developers to do a better job in the first place. Designers and coders would not normally want to be associated with below-par modules and subsystems.

The notion of an assessment is familiar elsewhere. While software provides one instance of non-full-testability, there are others. For example, consider the storage of nuclear waste. This involves a number of activities. One is the selection of a suitable site from a number of possibilities. A second is the construction of the repository, having consideration of geology, land contours, the possibility of earthquakes, the properties of the materials to be used, and many other issues. And a third is the specification and creation of a transportation system to take the waste to the repository. There are numerous variables, and every proposal or plan must contain many assumptions. How can they all be tested without building the system itself? They can't be, so a thorough risk analysis is carried out.

In such safety-critical situations, thinking in terms of risk is accepted as the norm. Identifying and analysing the hazards and managing the risks are standard practice. But in non-safety-critical industries this is not so. And why should it be? If safety is not an issue, why should time and money be spent in analysing and managing risks? Yet, why shouldn't they be? Risk is not restricted to safety. It is inherent in every decision that we make - how we invest our money, what we insure and what value we insure it at, and even whether we choose to risk running out of

petrol by passing one garage and driving on in search of a cheaper one. Software-based systems carry risks, so why not extend the remit of testers from merely finding faults to reporting on how the software meets certain criteria that are correlated with faults? We know that testing is not exhaustive and that residual faults almost certainly remain, so why not provide information that offers clues to where they are most likely to be?

In doing this, the test team would go beyond merely finding bugs, but such a software assessment would not represent a risk analysis as in the nuclear-waste example. Indeed, risk need never be mentioned. Having first defined the boundaries of the various parts (subsystems, modules) of the software, a simple assessment would report on each - at initial testing and each re-test - under a number of headings, such as complexity, apparent bug density, quality of structure and documentation. Such information would not only be of use to future maintainers and new development staff who have to make changes to the software, but it would also provide necessary information to analysts who need to pay closer attention to operational risks. Such attention may not be the current order of the day, but in the future, when an understanding of risk is routinely included in the training of testers, not only testers' assessments but also deeper analyses may become routine requirements.

A confidential report that relates the various software attributes to designers and programmers, and reports on their reliability in keeping to agreed schedules, would help the development manager in targeting training, supervision, and counselling. It would lead to improved quality in development, to more efficient and effective testing, to fewer re-tests and, consequently, to substantial financial savings.

A perceived downside of a written assessment is that such openness could result in a requirement for improvements to the software before acceptance. But this wouldn't trouble developers and testers who promised high quality in the first place. To them, this added means of discovering real and potential problems is an asset. PT