

## DEVELOPERS AND TESTERS – WHO SHOULD DO WHAT?

In the last issue of *tNews* we talked about the "test readiness" criteria for software hand-over from development to the independent testers. This included the requirement for the developers to have completed code-level unit testing. In this *tHints*, we consider in more depth the question of what testing the developers ought to do.

### THE PROBLEM

On many projects, the division of testing labour is not clear. This can lead to gaps in test coverage, as well as strained relationships between the developers and independent testers. Developers sometimes complain that the testers lack sufficient technical understanding of the implementation to properly test the system. Testers sometimes complain that the developers give them poor quality software.

We find that this conflict is almost always caused by a lack of a clear understanding of the respective roles and responsibilities of these two groups.

In fact, projects can actually exacerbate this situation by unintentionally "rewarding" developers for delivering poor quality code to testers. If the only measure of progress on the part of developers is "code handed over to test", then they can be seen to achieve this sooner by rushing through their own levels of testing.

### THE BIG PICTURE

Every software development project needs a Master Test Plan (or some equivalent), which documents the overall test strategy. An important part of this strategy is deciding what testing the developers should do, and what testing independent testers should do. There is no "right answer" as to who should do what – this depends on many factors, including:

- the criticality of the software
- the development process being used
- the skills and resources available within each group.

Finding the right balance involves weighing up the advantages of independence and specialist testing expertise, against the disadvantages of duplication of effort and overcoming the learning curve. Testing done by developers can be very efficient, because they know their own code intimately and consequently know where it is most likely to "break". On the other hand, if they leave out some functionality or misinterpret the requirements, then they are quite likely to make exactly the same mistake during testing (and so their testing may not be very effective).

### AN APPROACH

We have found an approach that aims to get the right people doing the right testing:

- Make the developers responsible for the *internal* quality attributes of the software. That is:
  - Their testing should concentrate on the *implementation*: does each line of code do what it was intended to do, and are all the exception conditions handled appropriately?
  - The code they deliver should be "bullet-proof": does it do what it does reliably, and without unexpected consequences?
  - Their testing should be structural, using white-box test techniques.
  - Their measures of completeness should be related to code coverage (or its variants such as decision or path coverage).

- Make the independent testers responsible for the *external* quality attributes of the software. That is:
  - Their testing should concentrate on the functionality: does the system do what it is supposed to do, in a clear and useable manner?
  - Their testing should be driven by requirements and user-defined scenarios, using black-box test techniques.
  - Their measures of completeness should be related to requirements and scenario coverage.
- Ensure that project managers measure (and reward) progress on the basis of “code able to be meaningfully tested” rather than just code handed over on schedule.

This division of responsibility is easy to understand, relatively straightforward to implement, and is surprisingly effective. And in our experience, understanding the big picture of roles and responsibilities is the key to effective and efficient testing.