The Defect Management Process

Phil Jones

Introduction
According to ISEB, the test process “comprises planning, specification, execution, recording, checking for completion and test closure activities.” In ensuring that this objective is met, rigorous testing of the product will have to take place either manually, by automation, or even both ways in order to eliminate any defects. In a perfect world (making the tester’s life far too easy), all software will be delivered to test with no defects whatsoever. All the tests can be carried out with no faults ever detected and there would be no need to upset the responsible developer. However, this ideal world is unfortunately a fantasy; although on rare occasions tester may find themselves in a position where no defects are found, in the vast majority of cases this is simply not so. That is why it is so important to have a defect management process in place: that way, when defects are inevitably detected, the testers know exactly how to identify and manage them, streamlining the testing process and increasing its efficiency.

When the Defect is Detected
What would the process be when testers find this inevitable defect? If all testing has halted, the most urgent defects should obviously be those of higher severity, which must be fixed by the tester at the earliest opportunity. If Test Director or a similar defect-tracking application is used, the fault needs to be recorded. Details of the fault along with any supporting information can be included here, along with details of whom or which team is responsible for investigating and fixing the fault. As soon as this is done, notification of the fault by e-mail to the project team is advisable so that everyone is on the same page. Once a defect has been brought to the attention of the developer, he or she must decide whether or not the defect is valid. Delays in acknowledging a defect can be very costly in both time and money. Further details may be required by the developer from the tester for investigation and rectification. Aside from that there is little a tester can do until notified that the defect has been fixed and a retest is requested.

Once a tester has been notified that a bug has been fixed, the test that found the original defect will need to be re-run. If the test passes on this occasion, the defect can be closed in the tracking application again, storing the test details ran and, if possible, the explanation from the developer as to why the fault occurred in the first instance. If available, this information should also be noted even if the retest fails so that a full history of events is built up. If a retest does fail then full details need to be reported again to the responsible developer. This process can continue until the defect is fixed satisfactorily. Only once a defect has been fixed and passed subsequent retests can it be closed.

As will be discussed in detail soon, in practice and depending on the severity of the defect, software can be promoted to a live environment with unfixed defects. Throughout the testing phase of a project,
meetings (usually weekly) will take place with the project team. During these meetings an opportunity to discuss open defects may present itself; the team then decides how much of a risk not fixing a defect is to the success of a project. If it is deemed that a defect will have little operational risk, then the team may accept that the software can be promoted to the live environment and the defect becomes a ‘known defect’. The tester will note this in the test report and all written correspondence accepting the fault should be stored and included in the report as well. Note that the severity of a defect can be changed at any time as long as the defect is open. This would enable the project to go live with only low-severity defects.

**How is the Defect Managed?**

So how is the defect managed once it has been detected? Once identified, the tester will need to assess the severity of the defect. Guidelines are usually provided to help the tester in this task and could range from a critical defect on one extreme to a cosmetic fault on the other. A critical defect or “showstopper” would typically be a situation where the tester is unable to carry out any testing. Unlikely as this sounds, it does happen and often with very serious consequences. In less severe circumstances, removing the offending software from the server can be enough to remedy the situation. In the most extreme circumstances, the new software may damage the existing software on the server to the extent that everything needs to be reinstalled from scratch. This will put a strain on the time constraint placed on the tester for test completion. Of greater concern is the offending software damaging other applications interfacing with the server on which the software was placed. This type of showstopper could result in a lengthy period of downtime before testing could in any way resume.

At the other end of the scale we have the cosmetic defect. It’s worth remembering at this point that in some industries, cosmetic defects, as such, don’t exist and every defect will need to be fixed. An example of this would be the advertising industry and more specifically television advertising. Any glitches in a commercial that’s going to be broadcast nationwide could have commercial consequences for an organization. But for other industries, cosmetic defects can be common though the number of such defects will need to be limited and will often be flagged to be fixed at a later date. The testers’ responsibility is to make this minor defect known to the project team and for the team to decide what action must be taken: whether its severity needs to be raised, the defect needs to be fixed for the live date, whether a fix can wait to a later date, etc.

The majority of defects, however, will fall somewhere in between these two extremes. A tester will have to make a judgment call in assigning a severity level suitable for the defect. Again, there are guidelines to assist in this task and it is worth remembering that the severity level can be amended at any point. In many cases, however, assigning a severity level may not be difficult. If something doesn’t work as it’s supposed to, then the severity will be high and a fix will be required. For example, if other tests are able to be performed, a defect can be high-level without being a showstopper. On the other hand, if something does work as it’s supposed to but the ongoing process is not as expected, then this defect could be assigned a lower priority. An example of this may be a customer making a purchase on the
web. After successfully completing the purchase, the customer may expect to be automatically redirected to the company’s homepage. If this doesn’t happen and the customer needs to click on a link to return to the home page, this could be regarded as a minor defect. The product is not working as designed, but it is “livable,” at least in the short term. The project team may decide that it’s not the end of the world if a user only had to perform one extra step. Time and cost pressure may also play a part in preventing this type of defect being fixed before a launch date.

**When is a Defect Not a Defect?**

So when is a defect not a defect? Test environments are notorious for not completely replicating live environments. It is not uncommon for interconnecting systems to be “subbed out” for that which is being tested. This is usually done due to limitations in resources and costs. So if a test which impacts one of these interconnecting systems does not produce the expected result, when the developer responds “It’s only because XYZ is stubbed out. It wouldn’t happen in the live environment,” what is the tester to do? The challenge for the tester is to ascertain if this explanation is acceptable. It may indeed be more acceptable if the test was part of a regression, for example. As always, if in doubt, the tester should raise the defect and make the project team aware of the situation.

This brings us to another type of defect – the known defect. This is simply a defect which is known to exist, usually those of a low severity which have been identified as part of another project. What does the tester do when told “Oh, yes, we know about that one?” It’s safe to say that even if a defect is known, the tester should again make the project team aware of the fault. Even if the defect was raised as a low priority for another project, by no means does it mean that it’s a low priority in the current project. The project team still needs to be made aware of the fault and a decision regarding the need to fix must be made accordingly.

Begrudgingly for the tester, a defect may have to be closed due to an inability to reproduce the defect. When a tester is unable to reproduce a defect and the event appears to be an isolated incident, there will be a tendency for the developer to assume the defect is invalid - that the defect is caused by user error or some misunderstanding. With very little information to go on, the developer may feel that nothing can be done. In such circumstances, if the tester feels the need to investigate further, the test needs to be rerun and rerun (along with some debugging program) and testers must wait for the incident to reoccur. Then the tester can go back to the developer with the correct supporting information. The incident may only occur “once in a blue moon,” but the fact that it is occurring at all is enough to warrant further investigation by the project team.

A developer can also attempt to “explain away” a defect by simply saying “It’s always worked like that.” This may indeed be the case and may be confirmed amongst others. It is still worth the tester raising awareness of the perceived fault. There may have been a misunderstanding in the requirements that would not have been picked up until the testing stage had been reached. On the other hand, the testers concern may be justified as the system may well have “always worked like that” but that’s not the case for the future.
At the end of the day, if the tester is in any doubt whether a defect is genuine or not, it is best to err on the side of caution and make the error known. If need be, the project team together can decide whether the defect requires fixing if it does not appear to be serious. The earlier a defect is detected, the sooner the fix can be retested and, assuming all’s well, the defect can be closed. The way the defect is managed is also conducive to early closure. A developer may need to be constantly reminded of the urgent need for a fix in order for the tester to meet the testing deadline. The tester will be fully involved in the life of the defect until a resolution has been agreed upon. Once a resolution has been approved for all defects raised, then one of the criteria has been met for the product to move out of the testing phase in preparation for promotion to the live environment.