Why Pay High Prices for Defect Management?

Quality Costs Money, But A Lack Of Quality Costs More. Defect Management Is the Key To Any Project's 'Mileage.'

If you want quality costs to stop guzzling money, the most effective thing you can do is optimize your project's defect life cycle. Here's how.

By Trevor Atkins

Testing is much more than just finding bugs to squash. It's not an event, but a set of diverse activities playing a critical role in identifying problems of varied types throughout the project life cycle, far in advance of public access to the software.

Tracking the real costs of software failure, such as patches, support and rework, can be difficult, but it's clear that effective testing can help optimize the cost of quality. Within the activities of testing, defect management needs thoughtful consideration to ensure that communication is as efficient and collaborative as possible and turnaround is prompt.

Photograph by Jose Gil

In his article "Quality Cost Analysis: Benefits and Risks," Cem Kaner defines quality costs as those costs associated with preventing, finding and correcting defective work. He notes that these costs can be on the order of 20 to 40 percent of sales.

Kaner outlines four types of costs that when added together make up the overall cost of the current quality of the application. The "total cost of quality" is the sum of prevention, plus appraisal, plus internal failure, plus external failure.

Appraisal costs are associated with activities designed to find quality problems, such as code inspections and any type of testing. Subsequent to detecting a quality problem are the costs associated with what happens next—for example, how that quality report is handled, how it's investigated and how it's determined to be resolved.

The Defect Life Cycle

The defect life cycle describes how a quality report or defect is handled, investigated and resolved. The defect life cycle is one of the foundation processes in any company that produces software. It's primarily this process that describes how testing and development departments interact around issue or defect reports. However, in some cases, the entire project team will become involved.

The defect life cycle typically focuses on how the setting and tracking of the severity and/or priority of a defect is done. The severity rating is related to the likelihood that the defect will be encountered in the field and to the impact or cost of that encounter. Severity emphasizes the effort to reduce external failures as they relate to the total cost of quality within the organization.

Test managers must seek out potential improvements in all areas included in the formula for the total cost of quality. They must define efficient methods of communication between different organizational groups on the project team, and they must define the supporting processes.

As this illustration demonstrates, when decisions about product behavior and functionality arise, the test group interfaces with all members of the project team: development, project management, business analysts and product management.

The defect life cycle can be a very effective tool for managing the complex communication that occurs as a software system enters the testing cycle. If planned carefully, the defect life cycle can keep testing cycles running efficiently.

What Is a Defect?

In defining the defect life cycle, it's important to understand what a quality report, or defect, is. Our team uses traditional definitions of a defect, beyond simply evaluating conformation to the requirements:

- *"A software error is present when the program does not do what its end user reasonably expects it to do."*—G.J. Myers, "Software Reliability: Principles & Practices"
- *"The extent to which a program has bugs is measured by the extent to which it fails to be useful. This is a fundamentally human measure."*—B. Beizer, "Software System Testing and Quality Assurance"

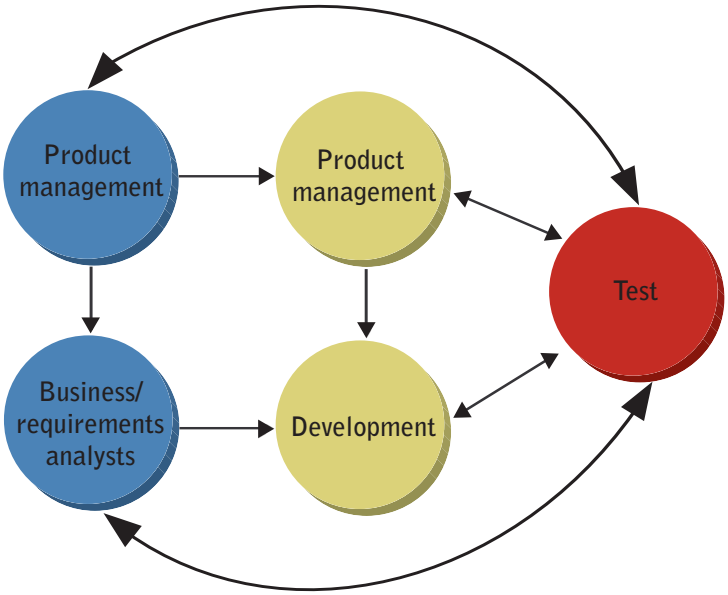
Incorporating these definitions into testing allows the project team to use the requirements as the product roadmap and, at the same time, take notice of broader quality issues along the way.

Attributes of a Defect Report

Many attributes can be ascribed to a defect in order to classify, organize and analyze associated issues. Aside from having a unique *Identifier* (DefectID), a *Description* of the issue with *Reproduction Steps*, and *Expected and Actual Results*, a defect report might include some of the

Trevor Atkins is co-founder and vice president, operations, of QA Labs. Formerly a quality systems strategist at Paradigm Development Corp., he holds a degree in applied science in electrical engineering from the University of British Columbia.

1: PROJECT TEAM INTERFACES



following descriptions:

- Status
- Assigned To
- Priority
- Severity
- Functional Area
- Feature
- How Found
- Type of Environment
- Resolution
- Opened Version
- Opened By
- Opened Date
- Related Test Cases/Requirements
- History or Audit Trail

By including these attributes in the recording of defects, and as part of the defect life cycle, the information can be used to make observations and draw conclusions, typically using metrics [IEEE Standard for a Software Quality Metrics Methodology, IEEE Std 1061-1998 (Revision of IEEE Std 1061-1992)].

Using Defect Resolution

In the defect life cycle, all defects must be resolved at one time or another, or by definition they have not been addressed. Figure 1 highlights the Resolved state in a simplistic defect life cycle.

Three facets of the defect life cycle that are not clear from this simplistic diagram are when it is resolved; when it is determined that it is not fixed; and when it can be closed. The Resolution attribute can be used to address this issue.

Not all defects are fixed by development. For example, developers may resolve a defect as Not a Bug; By Design; Not Reproducible; Duplicate Bug, etc., as the reason for moving the defect out of their queues.

In his paper "How To Win Friends, Influence Programmers, and Stomp Bugs," Kaner suggests the following list of options for the resolution of a defect in your defect database:

Fixed: The programmer says it's fixed. Check it.

Cannot Reproduce: The programmer can't make the failure happen. Add details, and notify the programmer. Also known as "Not Repro."

Deferred: It's a bug, but it will be fixed later.

As Designed: The program works as it's supposed to. Also known as "By Design."

Need Info: The programmer needs more info about the bug.

Duplicate: This is a repeat of another bug report (cross-reference it on this report.)

Withdrawn: The person who reported this bug is withdrawing the report.

The resolution type can contain other options in addition to the above, such as Enhancement, Spec Issue, Not Implemented, Feature Request and Third Party.

Resolution Philosophy

In the Resolution attribute we can capture much of the philosophy, or underlying ideology, of the defect resolution process. The core of this philosophy is to obtain a more detailed picture of the defect counts, allowing an improved analysis of that data for more accurate and useful metrics. Using the resolution type options provided above, we can investigate some straightforward interpretations of their use:

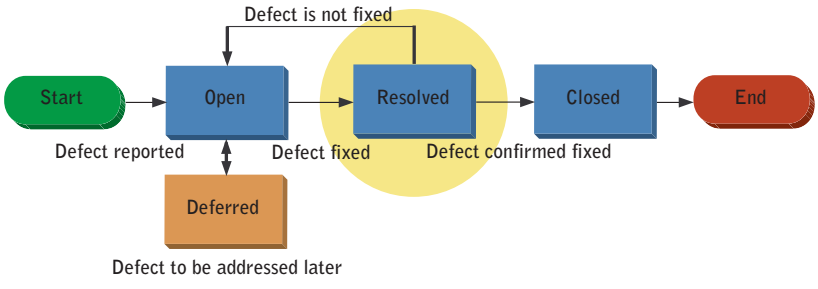
Fixed implies that there really was a problem in the code and it has been addressed.

By Design implies that the tester may not have the latest information about the functionality—or may not have the necessary understanding of the product.

Enhancement implies that the tester has not found a defect per se, but that the issue is a new feature or feature modification request. In other words, this is not a defect, but it has been implemented in the current release (as opposed to those that have been deferred). This information is valuable for the future, as these records can then be distinguished from the others for easy collection and inclusion in the requirements document and Help files.

Not Repro implies that there is not enough information in the report for the developer to be able to reproduce the defect, and that the tester needs to clarify or add information or withdraw the defect. There may be preconditions or hardware setup required for observ-

2: SIMPLIFIED DEFECT LIFE CYCLE



ing this defect that need to be added to the report, or pointed out to the developer if already in the report (such as the build number or environment information). This resolution should appear only as a transitory value before the true resolution is set.

You could create similar definitions to the basic examples noted above if you were looking at just the surface of the defects. However, no doubt you've experienced instances (such as deadline pressure) where limited or poorly defined resolutions have resulted in some of those available resolutions being used inappropriately, particularly as developers try to clear out their queues. In these cases, it often becomes the job of the tester, through the mechanism of the defect life cycle, to make sure the defects get to the right audience and aren't simply let go.

"Need More Info" and "Cannot Reproduce" are examples of resolutions that can create excessive dialogue or "churn" between developers and testers. The examination of how many defects get these kinds of resolutions, as well as the reasons why, can provide good insight into training opportunities within the project team, which can help to reduce this kind of rework. These investigations can also result in improvements to the application or the introduction of new tools that can help with future problem diagnoses.

"By Design" may be a defect that should not have been logged, as implied above. But what if the design is flawed? Referring back to the definition of a defect can be helpful in deciding the next step. Having the necessary decision gate for this resolution in the defect life cycle will allow that step to occur.

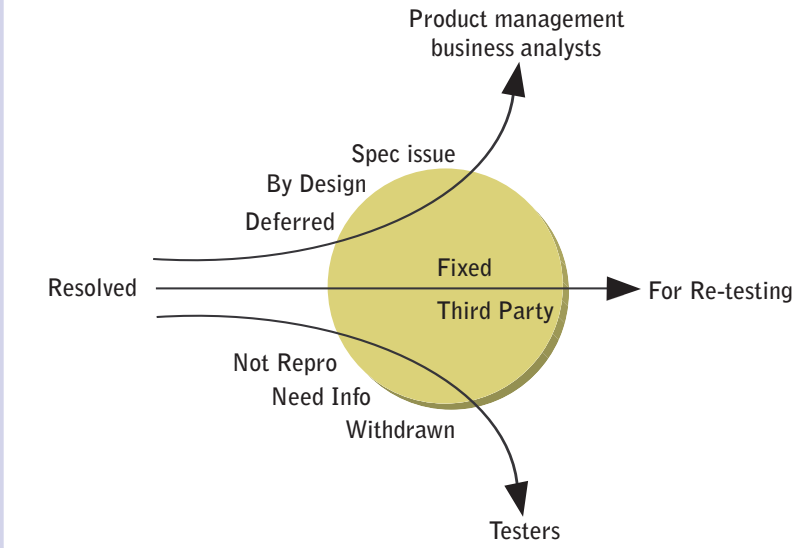
A defined and visible defect life cycle process that incorporates support for defect resolution will drive efficient communication while simultaneously enabling more accurate interpretations from collected data. For example, if a defect is resolved "As Designed" or "Deferred," perhaps that defect is then assigned to the business analyst responsible for that functional area. That person can then confirm the issue before it goes back to the tester for review. Alternatively, it might even be escalated to the product manager for review.

"Duplicate" defects can indicate a number of scenarios: a higher likelihood that the defect will be encountered; poor organization in terms of resource effort overlap; poor processes for making sure the majority of duplicates are caught before going to development; or even poor training. Given the number of possible implications, review of the defects with this resolution may need to be incorporated into "bug triage," to help set the appropriate priority and the project milestone reviews, and "post mortem," to determine process and training improvement requirements.

tions choices—how the defect life cycle workflow can be influenced by appropriate use of the Resolution attribute to make sure the right questions are asked by the right people. Ultimately, this strategy ensures that the product will be of a higher quality and that cost savings will be realized as a result of more efficient communication and decision-making.

Just as defect reports provide valuable insight into the commonly occurring types or classes of defects, analysis of

3: HOW RESOLUTIONS INFLUENCE THE DEFECT LIFE CYCLE



Process Implications

An effective defect life cycle ensures that defects won't be investigated for the wrong reasons and that the correct defects will be logged. Referring to the last example above, it's important to log as few duplicate defects as possible to minimize valuable development time spent investigating the issues.

Before reporting and assigning a suspected defect to a developer, the tester should review all currently open defects (or search for similar defects via keywords). If it is determined that the defect is a duplicate, the tester can then add any additional information to the existing defect, note whether the differences are of a greater degree, log a new defect record and associate the two.

This process will ensure that even if duplicate defects get logged, there is a valid process in place that not only illustrates that the testers are taking proper care in their defect logging but also minimizes the amount of time spent by the developers on investigating quality issues.

Figure 3 shows—for just a few resolu-

other attributes can provide useful information for driving improvements. Without specific tracking of defect resolutions, the true defect find rate and defect clustering in the code will be obscured by Duplicates, Not Repros, By Designs, Enhancements and Feature Requests.

Recording and analyzing this information will help ensure that you will be able to investigate and address the root causes of these quality costs. An adaptive approach to testing processes, communication and training can show that you have a strong and capable test team. Including a Resolution attribute with well-thought-out choices as part of your defect database, and more specifically within the workflows of your defect life cycle, will help you achieve this. ☒

REFERENCES

1. Beizer, B., "Software System Testing and Quality Assurance," Van Nostrand Reinhold, 1984
2. Kaner, Cem, "How To Win Friends, Influence Programmers, and Stomp Bugs, " www.kaner.com/pdfs/BugAdvocacy.pdf
2. Kaner, Cem, "Quality Cost Analysis: Benefits and Risks," in Software QA, Vol. 3, No. 1, 1996, p. 23
3. Myers, G.J., "Software Reliability: Principles & Practices," John Wiley & Sons, 1976