Effective Software Defect Tracking

Reducing Project Costs and Enhancing Quality

Bala Subramaniam
ISSRe Systems, Inc.

The costs of defective software can be as high as 50 percent of the investment in software development. Yet, the potential to improve software quality and reduce project cost is enormous. Software defect tracking can be an effective means to achieve quality at less cost. However, defect tracking is commonly misunderstood, incorrectly implemented, and often seen as an impediment and cost to the organization. This article discusses the quality costs of defective software and provides a working model to implement an effective software defect tracking system within an organization.

DEFECT TRACKING IS SOMETIMES written off as boring, repetitive, and unglamorous. Even effective defect tracking is often viewed as an unnecessary cost that impedes schedules. Yet, defect tracking is one of the most critical components of the software development and the quality assurance efforts.

When implemented well, defect tracking greatly reduces overall project costs and improves schedule performance. As a critical component in improving software quality, the potential paybacks for such processes are enormous. A Hewlett-Packard quality program reduced software errors by 75 percent and cut development time 20 percent. An Air Force systems group reports that every dollar invested to improve quality has a conservative return of $7.50.

To effectively track and manage software defects also improves customer satisfaction, creates higher productivity and quicker delivery, and leads to better operational reliability and improved morale. On the other hand, a mismanaged software defect tracking program may indeed be an unnecessary cost.

Software defects take different names in different organizations, e.g., errors, issues, bugs, defects, or incidents. Whatever they are called and whatever form they take, defects can have an astounding impact on the development phase and can continue to haunt the product through its maintenance phase.

The costs to fix software defects are high, especially if fixing requires developers to re-familiarize themselves with months-old work or if someone other than the original developer is doing the fixing. Costs also increase exponentially while moving further along the software development lifecycle. Studies at IBM demonstrate that compared to catching defects before or during coding, it is 10 times more costly to correct an error after coding and 100 times more costly to correct a production error.

Software Quality Costs

A 1996 study by The Standish Group reported that U.S. businesses invest about $250 billion in software development annually, yet a great many of these projects fail because of cost overruns. One of the significant components of project costs is software quality cost. One estimate put the cost of a single post-release defect to a large organization as high as $20,000 to $40,000.

Software quality costs are the costs associated with preventing, finding, and correcting defective software. Following are three useful definitions of quality costs [1].

Prevention Costs. These are costs of activities specifically designed to prevent poor-quality software, e.g., costs of efforts to prevent coding errors, design errors, additional document reviews to reduce mistakes in the user manuals, and code reviews to minimize badly documented or unmaintainably complex code. Most of these prevention costs do not fit within a typical testing group's budget. The programming, design, and marketing staffs spend this money.

Appraisal Costs. These are costs of activities to find defects, such as code inspections and software quality testing. Design reviews are part prevention and part appraisal. Formulating ways to strengthen the design is a prevention cost, whereas to analyze proposed designs for potential errors is an appraisal cost.

Failure Costs. These are costs that result directly from poor software quality, such as the cost to fix defects and the cost to deal with customer complaints. Failure costs can be divided into two main areas:

- **Internal failure costs:** Costs that arise before the product is delivered to the customer. Along with costs to find and fix bugs are costs associated with wasted time, missed milestones, and overtime needed to get back on schedule.
- **External failure costs:** Once the software is delivered to the customer, poor-quality software can incur customer service costs or the cost to distribute a patch for a released product. External failure costs are huge; it is much cheaper to fix defects before shipping the defective product to customers. If a product has to be shipped late because of bugs, the direct cost of late shipment includes the lost sales, whereas the lost opportunity cost of the late shipment includes the costs of delaying other projects while everyone finishes the one that is error-ridden.

User interface defects are often treated as low priority and are fixed last. This can be a mistake. Product screens may be required for effective marketing and documentation. This can result in increased costs in nondevelopment areas, lost marketing opportunities, and contractual penalties. Unfortunately, numerical estimates of lost opportunity costs and delays are difficult to make and can be controversial [2].

All the above software quality costs contribute to the total cost of poor-quality software to the organization. In aggregation, the total cost of software quality may be presented as follows:

Total Cost of Quality = Prevention + Appraisal + Internal Failure + External Failure.

What Is a Defect?

Defects are commonly defined as "failure to conform to specifications," e.g., incorrectly implemented specifications and specified requirement(s) missing from the software. However, this definition is too narrow. Discussions within the software development community consistently recognize that most failures in software products are due to errors in the specifications or requirements—as high as 80 percent of total defect costs [3]. Other studies have shown that the majority of system errors occur in the design phase [4]. Figure 1 represents the results of numerous studies that show approximately two-thirds of all detected errors can be traced to the design phase.

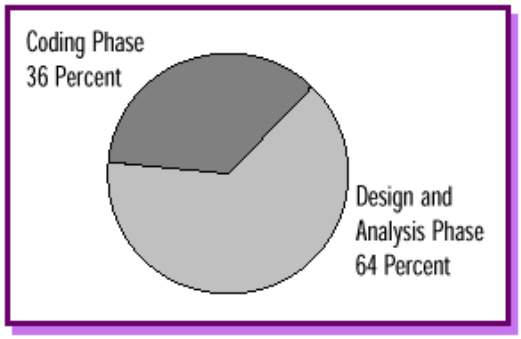

Figure 1. *Origin of software errors across industry.*

I recommend a broader definition of defect: *variance from a desired attribute.* These attributes include complete and correct requirements and specifications, designs that meet requirements, and programs that observe requirements and business rules.

Implementing an Effective Defect Tracking Process

Software quality assurance departments can play a catalytic role in implementing an effective defect tracking process. A survey conducted in 1994 by the Quality Assurance Institute found that a mere 38 percent of the organizations had formal software defect management processes, whereas 25 percent of the survey participants said their organizations lack consistent testing standards and procedures [5]. The survey also reported that although 60 percent of organizations had testing standards and procedures, some organizations admitted they were out of date and not followed. Recent surveys, nonetheless, suggest that more companies are now striving to improve their software development process through early defect identification, minimizing resolution time, hence reducing project costs.

Effective defect tracking begins with a systematic process. A structured tracking process begins with initially logging the defects, investigating the defects, then providing the structure to resolve them. Defect analysis and reporting offer a powerful means to manage defects and defect depletion trends, hence quality costs.

Integrate Software Development and Defect Tracking

Traditional approaches place testing immediately before implementation. Typically, testers receive a low-quality product at the tail end of development when there is tremendous pressure to deliver, even if the software is plagued with defects. For early defect detection and resolution to take place, defect tracking and software development efforts should begin simultaneously. It will solve a multitude of problems downstream.

Defect tracking must be implemented throughout the development lifecycle. On projects I have managed or worked on, this has always lead to fewer release defects; however, such organizational foresight is rare. The Sentry Group reported that 62 percent of all U.S. organizations do not have a formal quality assurance or test group. The report also added that a large majority of these organizations place a much higher priority on meeting schedule deadlines than producing high-quality software [6].

The Different Phases of Defect Tracking

Successful verification throughout the development cycle requires clearly defined system specification and software application business rules.

Requirements phase. Defect tracking focuses on validating that the defined requirements meet the needs and the users' expectation about functionality. Sometimes, system-specific constraints would cause the deletion of certain business requirements.

Design and analysis phase. Efforts should focus on identifying and documenting that the application design meets the business rules or field requirements as defined by the business or user requirements. For example, does the design correctly represent the expected user interface? Will it enforce the defined business rules? Would a simpler design reduce coding time and documentation of user manuals and training? Does the design have other effects on the reliability of the program?

I experienced the downstream cost of a specification error when working on one of two groups of developers who were programming complementary parts of a data-bridging program. Coding was well under way when incomplete system specifications caused transfer of data on the bridge to fail. The failure was not due to coding errors but to specification errors that were translated into program codes. Had the deficiency been discovered before coding began, we could have saved the substantial time and money required to repair the programs.

Programming phases. Defect tracking must emphasize ensuring that the programs accomplish the defined application functionality given by the requirements and design. For example, has any particular coding caused defects in other parts of the application or in the database? Is a particular feature visibly wrong?

Maintenance and enhancement phases. During the maintenance phase, effort is spent tracking ongoing user issues with the software. During enhancement phases (there could be multiple releases), defect tracking is focused on establishing that the previous release is still stable when the enhancements have been added. Figure 2 represents the philosophy of defect tracking throughout the software development process.

Introduce Defect Tracking Early

It is not difficult to introduce tracking early into the development process—it fits well into current software development processes. Today's rapid application development (RAD), the dominant approach in client-server software projects, focuses on shortened development schedules. This software development method provides early review points, delivered as "builds" or iterations of development, to ensure that requirements are met. Such a process clearly lends itself to early defect tracking, which can shadow development (see Figure 2). Each build can receive verification that it meets the defined requirements; if not, defects can be reported and resolved quickly and relatively inexpensively while the software is still "pliable." The same defect reported later in the development process may require a major "surgery" to the software product; hence, it will be more costly. Front-end defect tracking costs much less than waiting until the end.

> "Quality comes not from inspection, but improvement of the development process."
>
> — W. Edward Deming. [7]

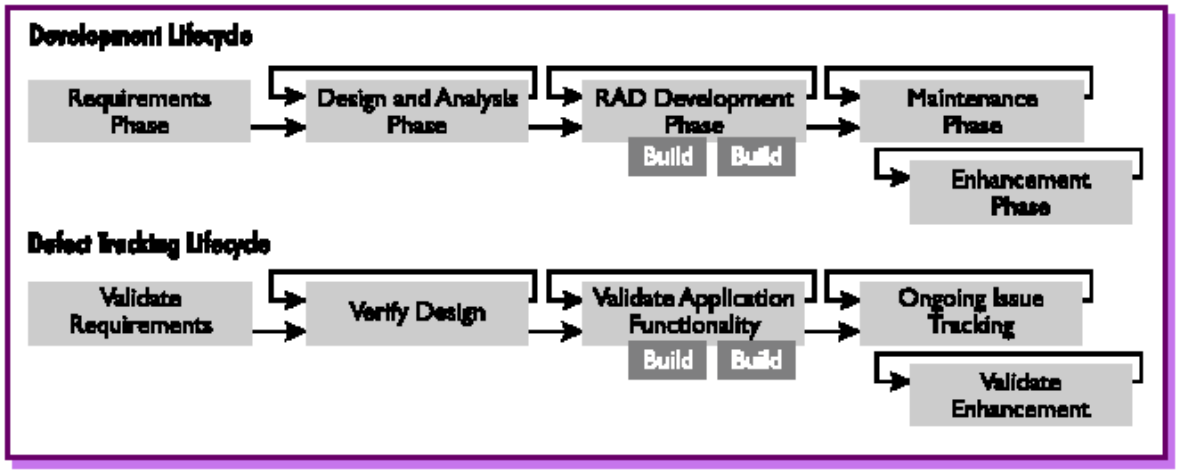

Figure 2. *Defect tracking running parallel to development lifecycle.*

An Effective Defect Tracking Process

To merely integrate defect tracking into the development process is not enough. A clearly defined defect tracking process is needed to ensure defects are handled in an organized manner from discovery through resolution. Components of this process are described in the sections that follow. This process is progressive—defect evaluation cannot be successfully performed if the earlier components (such as describing defects and prioritizing defects) were not implemented.

Defect Repository

Once a defect has been discovered, the important first step is to log the defect into a defect-tracking database or repository. When a defect is logged, it must be fully described so that it can be reproduced during debugging, prioritized based on its severity, and have resources assigned for its resolution. Defects have a number of other attributes that should be recorded, such as

- Defect number.
- Date.
- The build and test platform in which it was discovered.
- The application requirement or business rule to which it relates.
- Any supplementary notes.

It also is important that the repository offer a means to track the "life" of the defect (the resolution status) and historically report on all defects discovered and logged for the project. It pays to have this system online and available to all development staff so that the assigned parties can update the resolution progress for the defect status.

Defects Described

Your organization's defect reporting procedures should require that details about each software defect be recorded when the defect is discovered, including a description, symptoms, sequence of steps to re-create it, and severity. Defects are of various types:

- **Interface defects** include incorrectly working menu items, push buttons, and list boxes.
- **Navigational defects** could be described as a window not opening when moving from one interface screen to another.
- **Functionality defects** could be incorrect calculation of salaries in a payroll system.

Do not merely log, "Adding new customer window does not work." A detailed description, such as, "The `Save' button on `Add New Customer' window does not work," would give the developer adequate information to go straight to the specific problem and repair it. This saves time and unnecessary interruption for the developer to research the defect thus reducing the overall project cost.

Defects Prioritized

Once a defect is logged and described, appropriate resources must be allocated for its resolution. To do this, the defect must be analyzed and prioritized according to its severity. Each defect is given a priority based on its criticality. Usually, it is practical to have four priority levels:

- Resolve Immediately.
- High Priority.
- Normal Queue.
- Low Priority.

A misstatement of a requirement or a serious design flaw must be resolved immediately, before the developer translates it into codes that are implemented in the software—it is much cheaper to amend a requirement document than to make program code changes. The wrong font size for a label may be classified as "Low Priority."

The critical path for development is another determinant of defect priority. For example, if one piece of the functionality must work before the next piece is added, *any* functional defects of the first piece will be given the "Resolve Immediately" priority level. On one project I worked on, a query engine retrieved transactions matching user-specified criteria upon which further processing was performed. If the query engine had been defective, no further development (or testing) would have been practical. Therefore, all functional defects of the query engine were prioritized as "Resolved Immediately."

The urgency with which a defect has to be repaired is derived from the severity of the defect, which could be defined as follows:

- Critical.
- Important.
- Average.
- Low.

A defect that prevents the user from moving ahead in the application—a "show stopper"—is classified as "Critical," e.g., performing an event causes a general protection fault in the application. Performance defects may also be classified as "Critical" for certain software that must meet predetermined performance metrics. If the user is able to formulate work-arounds where there are defects, these defects may be classified as "Average." An overly long processing time may be classified as "Important" because although it does not prevent the user from proceeding, it is performance deficiency. Defects with severity "Average" will be repaired when the higher-category defects have been repaired and if time permits. Certain graphical user interface defects, such as placement of push buttons on the window, may be classified as "Low," since this does not impede the application functionality. Although defect priority indicates how quickly the defect must be repaired, its severity is determined by the importance of that aspect of the application in relation to the software requirements.

Structured Resolution

The defect tracking system also must ensure that the defect progresses in an appropriate sequence from discovery through resolution. Each defect is also given appropriate status; for example, a new defect is given the status of "Open," and a defect under repair would have the status of "Assigned."

As repair work progresses, the status of defects is updated to reflect its state in the resolution process. A defect that has been repaired will be submitted to the testing team through formal change control to be verified again. Only if the fix passes the regression test will it be accepted and the defect assigned a status of "Closed." Other defect statuses could include "Deferred," if the defect is not to be fixed for the current release but may be resolved in a subsequent release or "Enhancement," if a feature that is not part of the requirements has been suggested, and may be reviewed as an enhancement for later releases.

Communication

An effective defect tracking system must allow communication of the software's defects, status, or changes to members of the development team and all others concerned. This has become an increasingly crucial element because people working on the same project may not only work in different parts of a building but also may even work in a different state for a different organization. Without an effective means to communicate defects, defect tracking—and consequently achieving software quality—would be a nightmare.

E-mail is an efficient vehicle to expedite informing software engineers and all concerned of defects

as they are discovered. Software engineers could then perhaps access an online defect repository as they receive the E-mail on new and existing defects. Similarly, E-mail also serves as a reply medium to inform testers that a defect has been repaired. Some defect tracking repositories, e.g., one set up in Lotus Notes, facilitates built-in communication features that can be used by both software engineers and testers.

Commercially available defect tracking software, e.g., AutoTester and SQA Team Test Software, are more sophisticated in communicating defects and their status to individuals or as a batch. They also automatically inform respective development staff and management of defects as they are discovered.

Although E-mail provides a means to convey information about defects between the development and testing team, regular formal defect tracking meetings also help keep a close eye on the number, types, and nature of defects found, which may indicate how software quality is progressing through the resolution stage.

Defect analysis is discussed in more detail in the "Reporting" component of this defect tracking process. If the testing and development teams must work hand-in-hand toward achieving software quality, there must be continuous communication between them. Informal or verbal communication between these teams is inadequate.

Continuous Defect Resolution

It costs much less to resolve defects as soon as they are discovered—do not merely accumulate a list of defects to fix later. For example, in my current project, the software product is undergoing two transformations: The entire application architecture is being revamped and enhancements are being implemented for the next release. Revamping the architecture changes the fundamental "backbone" of the application in question, which is in itself a complex task. We have three categories of defects:

- The existing list of yet-to-be-resolved defects from the original application.
- Defects that would come about as a result of revamping the architecture.
- New defects contributed by the new enhancements.

We have divided the project into smaller deliverables and implemented defect tracking for each deliverable. If resolution were to be delayed until later, the mere complexity of the various deliverables would present an inordinate amount of challenge to resolve defects. Moreover, different software engineers are working on different deliverables and different tasks within each deliverable. At later stages, it would become a mammoth effort to merely identify and assign defects to the respective software engineers. Additionally, when they have been assigned defects to repair, the engineers have to remember what they implemented in the codes perhaps months earlier. This will incur expensive investigation time.

The best time to resolve defects is when they are discovered. This is especially true in a RAD environment, where the application is developed through several iterations or builds. Each build has an incremental amount of application functionality and related coding. Any defects discovered in a particular build should be referred to developers immediately for resolution. The functionality added in the most recent build and related program codes are still fresh in the developers' minds, which leads to faster investigation of the root cause of the defects, and therefore more efficient resolution efforts. To defer defect resolution until later in the development cycle wastes time and resources.

Defect Evaluation and Analyses

Most organizations consider it essential to constantly monitor and evaluate their performance, and this key practice is especially critical in defect removal. The overall success of your project largely hinges on effective defect resolution, so you need to know your defect removal status and the cost of achieving quality. For example, a defect trend analysis will indicate the number of defects discovered over time. This analysis may even be further subdivided for defects by status, functionality, severity, etc. Defect age analysis suggests how quickly defects are resolved by category.

The type and extent of the defect evaluation and analyses may be determined by the organization's cost objectives and delivery schedules. Following are a few suggested analyses that may be applicable to most software projects. The following measures need to be determined to analyze defects (or those chosen as part of an organization's defect analysis strategy).

- Defect status vs. priority.
- Defect status vs. severity.
- Defect status vs. application module.

- Defect age.

The above information will not be available if the earlier steps of adequately logging defects were not implemented as part of the defect tracking process. By comparing these measures from the current iteration to the results from the analysis of previous iterations, one can get an indication of defects trends, which are discussed further in the following two subsections.

Defect Evaluation. Although the evaluation of test coverage provides the measure of testing completion, an evaluation of defects discovered during testing provides the best indication of software quality. By definition, quality is the indication of how well the software meets a desired attribute. So, in this context, defects are identified as "variance from a desired attribute."

Defect evaluation may be based on methods that range from simple defect counts to rigorous statistical modeling. Rigorous evaluation can include forming a model (or setting goals) about discovery rates of defects, then fitting the actual defect rates during the testing process to the model. The results can be used to estimate the current software reliability and predict how the reliability will grow if testing and defect removal efforts continue. However, because the field's current lack of a scientific model and resources dedicated to perform such evaluations (or a tool to support them), an organization should carefully balance the cost of rigorous evaluation with the value it adds.

Defect Analysis. This means analyzing the distribution of defects over the values of one or more parameters associated with a defect. Defect analysis provides an indication of the reliability of the software. Four main defect parameters are commonly used for defect analysis:

- **Status:** the current state of the defect (open, being repaired, closed, etc.).
- **Priority:** the relative importance of addressing and resolving this defect.
- **Severity:** the relative impact of this defect to the end-user, an organization, third parties, etc.
- **Source:** what part of the software (such as a module) or requirement this defect affects.

Defect counts can be reported in two ways: (1) as a function of time, resulting in a defect trend diagram or report and (2) as a function of one or more defect parameters (like severity or status) in a defect density report. These types of analysis provide a perspective on the trends or distribution of defects that reveal the software's reliability.

Defect trends follow a fairly predictable pattern in a testing cycle. Early in the cycle, the defect rates rise quickly. Then, they reach a peak about midstream, in an adequately staffed test project, and fall at a slower rate over time. The project schedule can be reviewed in light of this trend. For instance, if the defect rates are still rising in the third week of a four-week test cycle, the project is clearly not on schedule. Other instances where the rate of closing defects is too slow (experience rated) might indicate a problem with the defect resolution process; for example, resources to fix defects or to retest and validate fixes might be inadequate.

This is an important aspect of software project management: to ensure that software quality is progressing within the planned delivery schedule. Figure 3 displays defect status by software module. In each software module, a discovered defect is given a status of Open and assigned resources for fixing.

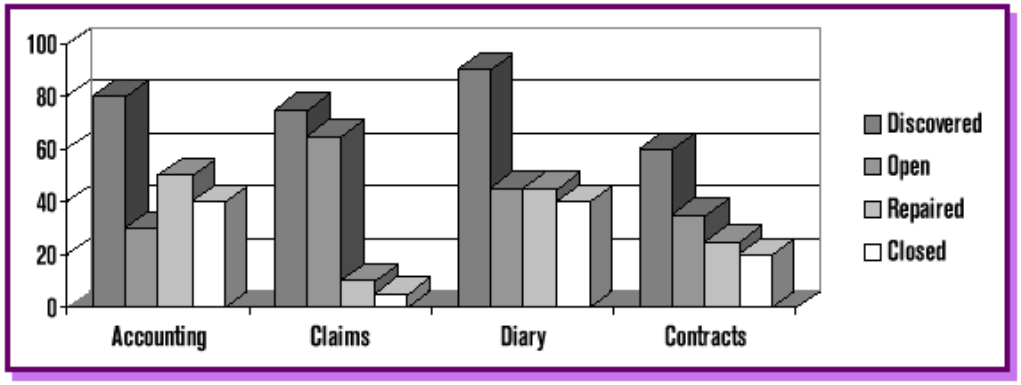

Figure 3. *Resolution efforts for an accounting system.*

In Figure 3, resolution efforts for accounting appear to be good, because there are more defects being repaired than left open. Retesting efforts seem to be adequate, because defects closed are not far behind defects repaired. However, although there is a similar retesting effort for the claims module,

there are far too many defects open, indicating that additional resources may be required for this module. The trend for the diary module suggests that both defect repair and retesting efforts are progressing well. The defect trend for the contracts module also shows that defect resolution is progressing well.

An organization could set quality criteria for how the distribution of defects over priority levels should look, e.g., "No critical defects should stay open for more than one week." It would be expected that defect discovery rates would eventually diminish as testing and fixing progresses. A threshold can be established below that in which the software can be deployed.

Defect counts can also be reported based on the source, allowing detection of weak modules and "hot spots." Parts of the software that must be fixed repeatedly indicate a fundamental design flaw. In my current project, this type of analysis helped us come to conclude that the application architecture technology needed to be revamped. The originally chosen architecture, although technically superior, was more complicated and made the application extremely delicate to changes or defects "fixes."

Defects included in an analysis of this kind have to be confirmed defects. Not all reported defects turn out to be flaws; some may be enhancement requests, out of the scope of the project, or describe a previously reported defect. These defects must be reclassified as such. However, there is value in analyzing why many duplicates or unconfirmed defects are being reported.

Defect Reporting

Defect evaluation and analysis have to be reported in a useful form to those who make decisions about resources, costs, and delivery schedules. Although each organization may want to produce different reports and different forms, there are three classes of reports:

- **Defect density or distribution reports** allow defect counts to be shown as a function of one or two defect parameters. Using the priority parameter, defect distribution may be represented as shown in Figure 4.
- **Defect age reports** are a type of defect distribution report that shows how long a defect has been in a particular state, such as Open. In any age category, defects can also be sorted by any other attribute, such as Owner (developer assigned to repair defect).
- **Defect trend reports** show defect counts by status (New, Open, or Closed) as a function of time. The trend reports can be cumulative or noncumulative and help management identify defect rates by status thus providing an indication of how well the software quality is progressing through the project cycle. Figure 5 represents a typical defect trend report.

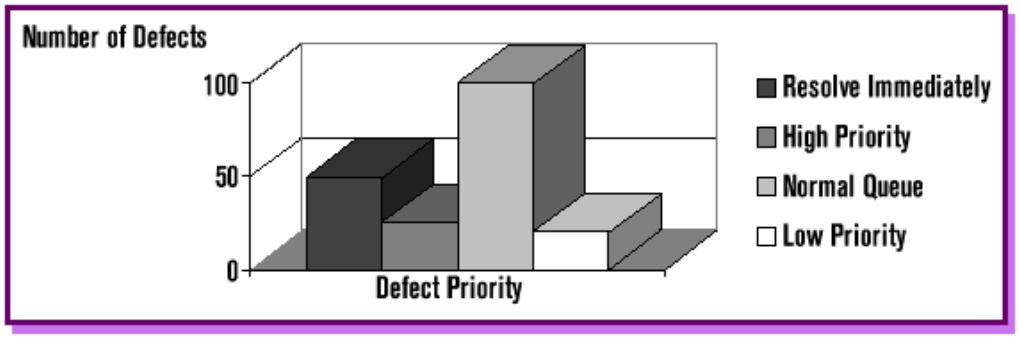

Figure 4. *Defect distribution.*

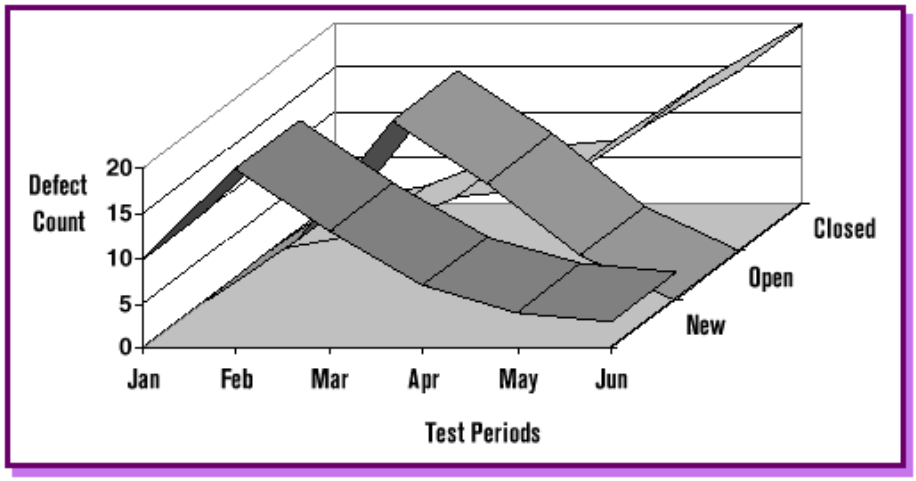

Figure 5. *Defect trend report.*

In Figure 5, the number of new defects peaked in February. Lagging behind new defects by about a month, the number of open defects was the highest in March. The defect-fixing efforts appear to be consistent throughout the project, closing all defects by June. Before an organization can produce the type of reports discussed here, a defect repository must have been established to support such analysis, i.e., provide for logging of defect description, status, priorities, etc., outlined in the "Defects Described" section of this article. To be successful, all pieces of the defect tracking process must be embraced.

Making This Approach Work

It is unwise to try to achieve too much too fast. Change is the most difficult concept to grasp or implement. The effective defect tracking model discussed here not only may call for a fundamental change in your software development process but also may require a broader concept and definition of defects and the tools to manage them. Depending on the capability of the development and quality assurance process, an organization may not want to attempt the total defect-tracking model all at once. For lower maturity organizations, an incremental adoption of the recommendations would be more successful.

A good start is to set up a simple defect-tracking repository that implements defect description, status, priority and severity, and communication. Expand that list later to include defect evaluation and analysis and reporting. This ensures that the required defect data is captured as a minimum so the organization can build on this model.

Next, the process should be widened to embrace the broader definition of a defect, and the concept of implementing defect tracking across the development process. The implementation of an effective defect tracking process should be taken through levels of maturity, which is a topic for an entire article. Once the model and process is applied to one project successfully, it can be implemented across the organization.

Conclusion

Effective defect tracking strongly contributes to enhancing software quality and reducing development project costs. Using the broader definition of a defect ensures that not only are resultant errors or nonconformance to requirements discovered but also variance from a desired attribute, including incomplete requirements, takes place. Searches for such defects can then take place across all software development phases.

By "shadowing" the software development process, defect tracking helps you identify and report potential software problems early and acts as a catalyst for problems to be addressed. By facilitating discovery of defects earlier in the development cycle, effective defect tracking is a critical key to lower costs, enhanced software quality, and reducing overall project cost. However, to achieve this requires a fundamental change in the ideology behind quality assurance and the software development process as well as the introduction of the necessary tools to track and manage defects. The defect-tracking model discussed in this article will be useful for organizations moving in this direction. Careful planning and phased adoption of this model can make this approach a powerful software quality strategy.

About the Author

Bala Subramaniam is director of quality assurance at ISSRe Systems, Inc., in New York. He has 15 years managerial and technical experience, during which he has worked for medium and large software companies including IBM. His special interests include the definition and implementation of quality assurance methods and software process improvement programs. He also is experienced in designing effective automated test methods to test complex mission-critical software functionality and business cycles. He has a master's degree in business administration (finance) from Birmingham Business School in Great Britain and is a certified software test engineer (Quality Assurance Institute).

ISSRe Systems, Inc.
200 Business Park Drive
Armonk, NY 10504
Voice: 914-273-7777
Fax: 914-273-7796
E-mail: balas@issre.com

References

1. Campanella, J., ed., *Principles of Quality Costs*, ASQC Quality Press, 1990.
2. Juran, J.M. and Frank M. Gryna, *Juran's Quality Control Handbook*, 4th ed., McGraw-Hill, New York, pp. 4.9-4.12.
3. ANSI/IEEE Standard 982.1-1988, *IEEE Standard Dictionary of Measures to Produce Reliable Software,* Institute of Electrical and Electronics Engineers, p. 13.
4. Perry, W., "Structured Approach to Testing," *Effective Methods for Software Testing*, John Wiley & Sons, New York, 1995.
5. Perry, W., "1994 Survey Results on Software Testing," *Effective Methods for Software Testing*, John Wiley & Sons, New York, 1995.
6. *Automated Software Quality Directions*, The Sentry Group, February 1998.
7. Deming, W. Edward, *Out of the Crisis,* MIT Press, Cambridge, Mass., 1982.