Best Practices for Effective Defect Tracking

Defect tracking is a fundamental and critical part of application lifecycle management. However, it is not uncommon for a defect tracking tool to be under utilized by software development and quality assurance teams, with much of the tool's potential functionality remaining untapped. This can be remedied by implementing best practices throughout the defect tracking process.

The notion of best practices does not commit people or companies to one inflexible, unchanging practice. Instead, best practices are a management approach based around continuous learning and continual improvement. This article covers the fundamental yet critical aspects of defect tracking through an examination of such best practices.

The Fundamentals

The fundamentals of effective defect tracking are the same whether a pencil and paper, a spreadsheet, or a full-fledged defect tracking tool is used. Implementation of the following fundamentals provides a solid foundation for success:

- **Information Capture**: Guidelines should be established that determine the minimum amount of information necessary to report a defect. For example, determine what information is needed on descriptions, products, and screenshots. Defect tracking must be simple enough so that people will use it, but cannot be oversimplified because it must capture vital information about the problem.
- **Reproduction**: To verify that a defect has been fixed, a user must first be able to reproduce it. While it is possible to fix a non-reproducible defect, it is a very difficult and time consuming task.
- **Prioritize and Schedule**: After a defect is found, it must be prioritized and scheduled to be fixed. Often this is subjectively determined by the severity of the problem as seen by the project manager.
- **Communication**: An open line of back-and-forth communication can facilitate constructive dialogue between who reported the defect and who is responsible for fixing it.
- **Environment**: Some defects only exist in specific environments. To provide thorough testing, the QA team must identify and test all possible hardware/software combinations.

Best Practices

Beyond the fundamentals of the defect tracking process, best practices should be implemented to maximize overall product development and quality assurance. The idea behind these best practices is that with proper processes, checks, and testing, a project can be rolled out and completed with fewer unforeseen complications.

Unified defect tracking

There is much to be gained by adopting a unified approach to defect tracking. Developers and testers need to follow a similar defect tracking methodology and use the same tool to make sure that the proper steps are taken to manage change. Be sure to involve and get feedback from all groups who will use the defect tracking tool, including development, quality assurance, customer service, field representatives, project managers, partners, and managers. By asking for input, each group is made a stakeholder in the system, ensuring the ongoing success of defect tracking projects.

Keep it simple

Simplicity is often overlooked when implementing a defect tracking tool. Even the most sophisticated applications do little good if they are not used. Make it easy for users to report defects by Users should be able to interact with the system easily and with as little frustration and overhead as possible. A system that is easy to use will encourage users to participate in the defect reporting process. Believe it or not a system can be configured for easy use by utilizing concepts like automation and workflow definition. It may seem like common sense but when a person is given a tool with ample features and capability often there is an overwhelming urge to utilize all the features and capabilities, which can lead to over configuration of the tool. This might frustrate users who simply want to add and track defects and not deal with an overloaded, over-configured defect tracking application.

Display information at the decision point

While it is important to capture essential information during the defect tracking process, it is also important to keep the input screens clean and easy to understand. Context-sensitive input screens greatly improve ease of use. Putting 100 fields on a single screen may overwhelm users. Information should be added as the issue moves through its lifecycle. For example, instead of displaying fix information when a defect is first entered, only display these fields when fix information is being entered. Some users may never need to enter this type of data, so the details should not be visible to everyone. Simple is always better when it comes to using the application, but make sure that the information needed to reproduce a defect and track its change history over time is not left out.

Use relevant terminology

Defect tracking tools should be simple and intuitive. When creating an input screen to capture defect information, it needs to be abundantly clear what each field represents. System administrators should spend as much time as necessary on the initial configuration of the system, working with project managers and team members to ensure the use of terminology that is familiar to the organization. If users add defects and see fields that are unclear or that they do not understand, then the defect will lack essential data. Try to capture only the most relevant information. If there are fields that do not help developers track and fix defects, or help testers understand the nature of a problem, then these fields can be removed, or at the very least made optional.

Capture report data

Users and managers will want to generate reports from the data captured by the defect tracking tool. When the application is set up, fields should also be added to capture information that helps with reporting. Information like date reported, user data, and organizational data (for example, the department that reported the defect) may not be useful when fixing a defect but will provide valuable reporting data. Users can create trend reports that track when defects are spiking, which users are reporting the most defects, which teams have the most issues, and so on. Project managers will be able to allocate and adjust resources based on these trends. For example, if the server application team is generating more defects than the client application team, then development and QA resources can be shifted to help the server team and prevent delays in the project schedule.

Write clear, reproducible defects

Growing up, Mom's favorite saying when she thought I was watching too much television was 'garbage in, garbage out'. The same phrase can be applied to defect tracking tools. It is vital to collect relevant, useful information because the benefits of the tool are only as good as the information put into it. When incorrect or incomplete information is entered, development and QA teams will waste time tracking down defects based on bad information or have to constantly request clarification about poorly documented problems.

The following fields should be required for every defect:

- **Title**: The title should be clearly written to increase the 'searchability' of the defect database. Think of how others might describe and look for the problem.
- **Summary**: A paragraph or two that describes the defect.
- **System configuration**: A defect may be found on Intel but not AMD, Internet Explorer but not Firefox, or Windows 2000 but not Windows XP. Be sure to capture the exact configuration of the system the defect was found on.
- **Steps to reproduce**: Explain how to reproduce the defect. This critical piece of information is often inadequately described, wasting time for developers and testers.
- **Expected results**: Describe how the application should work. If submitting a cosmetic defect, you may also want to attach a screenshot. If words alone do not suffice, attach a mock-up or a sketch showing how the application should look.
- **Notes**: Address anything not covered in the previous categories, such as team members to contact for additional information.

An important part of writing defects is the need to be careful with their classification. An improperly classified defect can get more attention than it deserves, wasting valuable time and resources, or an important defect might be overlooked because the severity is set too low.

Information accuracy can be hard to enforce. How do project managers make sure users provide good information? Most defect tracking tools include features to make defect reporting more accurate and to assist in capturing the information needed to find and fix a defect. Use required fields to ensure the correct level of detail is captured, configure email to notify customers when there is a problem with their defect reports, or use alerting mechanisms to notify users when the defects they reported are fixed.

Reduce ambiguity with screenshots

Software defects often have a visual component that cannot be adequately described. Attach screenshots of failures to reduce ambiguity and confusion. A screenshot can simplify the defect reporting process so dramatically that some organizations require them. Screenshots can be especially useful when team members do not speak the same language. One caveat to attaching screenshots is that high-definition image files, such as BMP or TIF, can use a large amount of disk space. Highly compressed file formats, such as PNG or GIF, are good alternatives.

Avoid defect duplication

One defect report is helpful. Ten reports about the same defect are not. To minimize duplication, users should query the database prior to submitting a defect. Make sure to run a few simple queries in the database to determine if the defect already exists. An uncluttered defect database makes it easier to manage projects and provides accurate statistical representations of product quality.

By avoiding duplication, users will not spend unnecessary time researching defects that have already been fixed. Although defects can be reported multiple times, they should be added to the same defect report. If not, they could be assigned to multiple programmers to correct without realizing the issues are duplicates. When a project manager generates statistical reports about the health of a project, the number of reported defects, the criticality of defects, or the rate at which defects are reported and fixed, the data may be skewed by duplicates. For example, a project with 15 critical defects may only have five critical defects if one issue was reported multiple times by 10 different users.

Merge duplicate defects if the defect tracking tool supports record merging. Merging does not delete information from the database and still allows users to track reported issues. Users can also easily identify which issues customers and other users have reported the most times. This information is useful when prioritizing issues and requirements. For instance, a critical defect that 10 customers reported may have a higher priority than an equally critical defect that was only reported by one customer.

Match the team's workflow

The purpose of a defect tracking workflow is to move issues from initial reporting to resolution. When a defect is reported, an organization may require the following to occur:

1. Verify the defect. Is it really a defect? Is it reproducible?
2. Allocate resources to fix the defect. How much time will development and QA need? How much will it cost? How long will it take?
3. Release the fixed defect. When will it be released? Who approves releasing the change into a build? How are code changes moved into new builds?

These questions, which affect project management, software development, QA, and release management, can all be enforced through a defect tracking workflow. For example, when a defect is added, it must be reviewed by a QA team member to ensure its accuracy and authenticity. Once QA determines that a defect exists, a project manager must prioritize and then assign the defect to a developer to fix. After the defect is fixed, QA must test and verify the fix. A build manager must then ensure the fixed defect is released to the next build. A customer may even perform customer acceptance on the issue and verify the fix. Finally, the defect is closed after the fix is verified in the latest build or release.

A defect tracking workflow can help define and enforce change management policies while providing traceability and accountability throughout the defect tracking process. The workflow will define states for the lifecycle of the defect being tracked. Each state indicates a change to the defect. How the workflow is configured will also determine what type of information is captured as the defect moves from open to closed. Some information, such as who reported it, the due date, what changed for a fix, and what version to release the defect fixes into, may also change. This information is not captured when the defect is added but is gathered as the defect moves through the workflow. It is imperative to configure an enforceable workflow that captures all the necessary information needed to manage the defect, from being reported to merging in code changes to releasing the next build containing the fix.

When implemented correctly, the workflow ensures that all users know their roles and assignments. After logging in to the defect tracking system, developers can be presented with a list of their assigned defects that need to be fixed and QA team members can be presented with a list of fixed defects that are pending testing and verification for specific builds or releases. Roles become better defined, information is exchanged more effortlessly and with better accuracy and precision, and teams in different departments will work together to find and fix defects. Instead of sending countless emails, which get lost over time, trying to explain what is happening on any given issue at any given time, now properly documented records that register dates, times, user effort, and user interaction

will all be easily accessible. This enables project members to know exactly who is doing what at any given time and, more importantly, know what has been done and what needs to be done.

Meet compliance measures

Many organizations are dealing with compliance measures, such as Sarbanes-Oxley (SOX), which put demands on information gathering, data integrity, process definition, and policy enforceability. A defect tracking tool with workflow capabilities provides a system that meets SOX compliance by capturing relevant change information throughout the lifecycle of a defect. For example, when a defect is fixed, information such as the user who fixed it, when the fix occurred, and why it was made are details that describe the change itself. These types of change descriptions, and the ability to lock down the history of these changes so they cannot be altered or corrupted at a later time, help meet compliance requirements and guidelines that often demand the ability to see unaltered change history with a reasonable description of the change (who made it, to what, why, and when). This makes an auditable record of activity available to ensure the process requirements and guidelines are being fully satisfied.

Integrate with change management

A popular feature in many defect tracking tools is the ability to integrate what is being done on the development side in code changes with what is being done in the defect tracking system. The ability to link code changes to defects will do several things:

1. Provide a better description of the change
2. Provide a new approach to release management
3. Simplify QA tasks
4. Provide better accountability
5. Provide better overall project management

The most common way developers document changes is by commenting on changes that are being committed or checked in to the source control system. Comments generally describe the nature of the change made to the source code and why the change was made. For example, a developer may comment about new functions that were added to a source code file during a check in and also specify a defect number that correlates with the check in.

The biggest problem with comments is that there is no way to ensure their usefulness. A check in comment may include statements that do not provide relevant information about the change such as, "I made a change". When this change is reviewed, it may not be easy to determine what changed or why. The ability to associate a defect with a check in or commit action provides an additional description of the change that was made. If a change explanation is needed for a modification, instead of relying solely on comments, users can simply read the defect details, which will include more information than comments alone. This information is expanded to input from the users who initially reported the problem, authored all the details, and other data such as the version initially reported in, products, components, hardware/software variables, comments about the issue, and other corollary information. This information may go much further in explaining and justifying certain types of changes than just comments.

The integration of defect tracking and configuration management can greatly simplify release management. Change is introduced into builds in specific ways, such as into one version of a modified file. With integration between defect tracking and configuration management, a release engineer can generate a query or report to identify the specific files and versions that belong to approved defect fixes for a given release. This allows the release manager to develop a system that ensures the exact components for a build are properly included.

By linking source code changes with a corresponding defect, a tester can directly access information that constitutes the defect fix. Information pertaining to specific defect fixes that exist in specific builds will be readily available to QA team members, allowing test plans to be executed more efficiently. For example, instead of getting a new copy of an entire Web site to verify a defect, a tester can easily get and test the individual files that represent the fix. When a page loads, the change may or may not be immediately obvious. Instead of spending time looking for the change, the tester can open the source code file and view the changes to the HTML. The tester can then use a differencing utility, common in most software configuration management tool, focus on specific changes no matter how small they are.

Involve customers

When possible, release the application on a limited basis to customers and users willing to beta test the software. This increases the range of QA testers well beyond what most companies can support. The more users testing the software, the more defects will be captured—and the sooner they are caught, the better. A defect tracking tool should offer a way to get information from users of the product, internal or external, without necessarily allowing them access to confidential data. Working with customers to identify

problems and resolve them is much more attractive than the alternative of waiting for the customer to report their dissatisfaction about the result of a bug fix after a new release. Very often when defects are reported something gets lost in the translation between the customer and engineer. Instead of making a code change that results in the 'this is not what I was really looking for' comment from the customer, include customers in the process, meaning solicit additional input when the defect is reported and ask them to verify bug fixes in pre-release builds, to make sure they are getting the proper results.

Evolve to a Higher Level

Best practices is a management idea which asserts that there is a technique, method, process, activity, incentive, or reward that is more effective at delivering a particular outcome than any other technique, method, or process. In other words, undertaking best practices will help an organization evolve to a higher level of productivity while maintaining high quality. By implementing the defect tracking best practices highlighted in this article, organizations will maximize overall product development while developing the agility necessary to overcome tomorrow's challenges.

Seapine Software™

www.seapine.com

5412 Courseview Dr., Suite 200
Mason, OH 45040

TEL 513-754-1655
FAX 513-754-1660