Defect Tracking Patterns
Steven E. Newton

Roles

Customer

The Customer role may be played by a real end user customer, but could likely be someone in QA, or any other team member. The individual entering the bug report is the system is playing the Customer Role.

Programmer

The Programmer is the role taken by the person who is assigned to write code or otherwise modify system to fix the defect. A Programmer would at a minimum unit test the fix, and run any unit test suites, and may, depending on the team, run regression and integration tests.

Tester

The role of Tester is to verify the fix and validate it against the defect report. The same person may take both the Programmer and Tester roles, but the goals are different. For the Tester, the implemented fix is not considered done until it provides a satisfactory resolution to the problem report

Dispatcher

The Dispatcher has the role of setting priorities and assigning bugs to be worked on, then following up on work. The Dispatcher also coordinates the life cycle events of a bug report between various other roles.

Keeping these roles in mind while reading the patterns is an important aid in understanding them.

The Patterns

- **Tracking Tool**
- **Automated TODO List**
- **Problem History**
- **Bug Ownership**
- **Priority and Severity**
- **A Bug's Life Cycle**
- **Fixed Is Not Closed**
- **Incidents and Faults**
- **Reviews Before Big Changes**
- **Tests Capture Bugs**

Tracking Tool

Problem

Post-it notes, programmer's memories, even email, are all unreliable repositories for bug reports and feature requests. All of them are also difficult to coordinate on a team of more than 2 people working together for more than a few days. Select a tool that is easy to use and very configurable.

Therefore:

Use an automated tracking tool. The tool doesn't have to be sophisticated -- it can be a spreadsheet in simple cases -- but it must be shared.

Discussion

Bug tracking software has features to help manage work. Users can create filters and sorts to assist workflow. Filter on the field Detected By with your id to show the bugs you've filed. Filter on Assigned To to see your responsibilities. Sort by Severity to track the top problems.

Automated TODO List

Problem

Programmers and testers need a way to know what tasks they are responsible for and what priority those tasks have.

Therefore:

The defect tracking system should have sort and filter features that allow maintaining an automated TODO list. Each item in the bug tracking system is assigned to one person and has a priority associated with it. The list of bugs becomes a personal, automatically updated and prioritized, TODO list.

As a developer, filter on Assigned To to see your responsibilities. Sort by Priority to track the top problems.

Related To: Use A Tracking Tool; Differentiate Priority And Severity

Problem History

Problem

A bug report has a life and grows with changes. Bug state is not static, and tricky defects can take several iterations of work to find and fix properly. In the context of this ongoing work, knowing what changed and why, and what worked or didn't, help maintain a sense of history to the code.

Therefore:

Document all work on a problem in the bug tracking tool. Use the description and comments field to make note of important issues and facts. Attach files with error output, screenshots, or even snippets of code patches to document what was done and why. Records of what was worked on and resolved represent a knowledge base.

Discussion

The related idea of micro-decision awareness asks only that when making a decision, however small, you are aware of the fact. Recording that decision in the bug tracking software preserves that decision.

Bug Ownership

Problem

Untracked bugs can lead to wasted or duplicate effort. The wrong programmer attempts to fix it or multiple programmers all work on different solutions at the same time without coordination. Or a programmer works on an already-fixed bug not knowing that the fix is in. The bug isn't really a bug, it's a new piece of functionality, leading to scope creep. Low priority, easy, or "fun" bugs get fixed before the higher priority bugs.

Therefore:

Define and use a bug lifecycle. Assign bugs to one person at a time, and give that individual responsibility for resolving it or assigning it elsewhere.

However

Adhering tightly to this pattern requires discipline in using the bug tracking system. The Use a Tracking Tool and Document Work patterns are intimately tied to this pattern's effectiveness.

Discussion

As the customer, watch your bugs and respond to queries for additional information or candidate fixes. If a bug you reported is no longer an issue, close it. If you are using a bug tracking system that allows it, you may watch bugs you do not own but have a stake in. As the developer, update your bugs when you work on them. If you get stuck, assign them to another developer. If you need additional information or want comments from the submitter, assign it back to them with appropriate status.

Identify someone to play the Dispatcher role, and modify priority and assignments as necessary. The Dispatcher role can streamline the process of moving a bug through its lifecycle.

Priority and Severity

Problem

The effect of a bug on the software does not automatically correlate with the priority for fixing it. A severe bug that crashes the software only once in a blue moon for 1% of the users is lower priority than a mishandled error condition resulting in the need to re-enter a portion of the input for every user every time.

Therefore:

Define a range for Severity as "how bad does it hurt?" according to project requirements. Define Priority according to business value and project timelines. High priority bugs should be first to be fixed. Track priority and severity separately, then triage appropriately.

Discussion

Importance and Destructiveness are alternate words which may be clearer to some.

An example of the difference and interaction is in bugs that are purely cosmetic problems, misspellings in dialogs, redraw issues, etc. These can be priority 1 fairly often, because they are frequently very annoying to users, and fixing them is generally easy and doesn't destabilize things.

The Developer, when determining what to work on next, would pick the bugs with the highest priority, regardless of severity. The Dispatcher is responsible for setting these appropriately, especially in the case where the Customer marks everything Very Severe and Highest Priority.

A Bug's Life Cycle

Problem

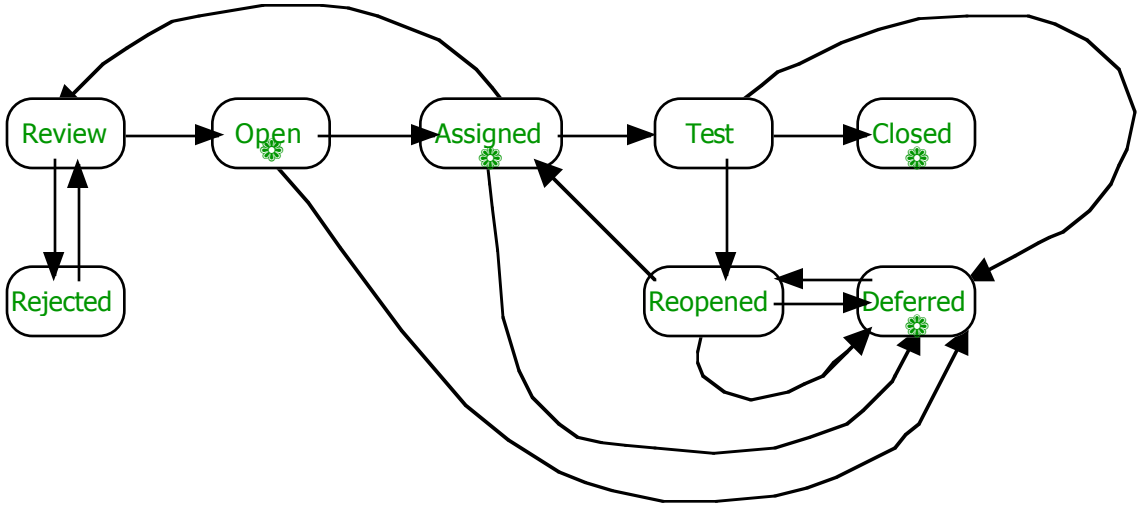
Identifying and fixing bugs involves several steps to coordinate as the bug changes states and assignees. Any team member with a stake in the defect needs to know what stage the defect is in at any time.

Therefore:

Define and follow a life-cycle for bugs.

Discussion

A defect can only be assigned to one person at a time. That bug may flow back to the reporter with a misdiagnosis or fix that doesn't work. The assignee may assign the defect to another programmer more familiar with the fault or for review. It may go back and forth from reporter to programmer a few times before it is fixed, retested, and resolved.



Related to: Document Work; Coordinate Efforts

Fixed Is Not Closed

Problem

Programmers who work on bugs need feedback to know if the changes fixed the bug as reported. Those changes need to be verified by a third-party and the work on the bug brought to resolution.

Therefore:

A fixed bug is not a closed bug. Once the programmer has made code changes and checked in the results, that programmer can mark the bug fixed, but cannot close it. The change should be verified to see if the bug is fixed.

Discussion
The resolution of a bug should be different from the status. "Will not fix" or "Unable to reproduce" are among the other resolutions besides fixed that can close a bug.

The Customer who submitted the bug should close it, or the Dispatcher role can coordinate fixed and closing.

Related to: A Bug’s Life Cycle

Incidents and Faults

Problem

A single defect one place in the code can cause different apparent problems to different users. It is also possible that a single reported problem may be traceable of multiple different places in the code.

Therefore:

Several reported failures that arise out of the same fault should be tracked together. Consider splitting a single report with multiple causes into a report for each fault, and relate them as above.

However

The bug tracking software must support the ability to associate bugs as duplicates of and dependent on other bugs. Splitting a single bug or otherwise tracking related bugs without good support from the tool requires too much manual overhead.

Reviews Before Big Changes

Problem

A bug or group of related bugs can result in a change that affects large sections of the codebase, or significant changes to one component. Common practice would encourage small, incremental changes and continuous integration, but sometimes a large set of changes must happen all at once.

Therefore:

Before the programmer commits the changes to the team source repository, assign the defect report(s) to a reviewer, typically a team lead, senior programmer, or module owner, for an extra-careful check. The reviewer can examine the proposed fix for compatibility, suitability, and overall conformance with direction of development.

However

Reviewing only before "big" changes is an anti-pattern, and may be a sign the team is not following Coordinate Efforts.

Discussion

The team is expected to be doing normal reviews all the time, either formal or informal, or through pair programming. When you do see big changes, it's usually in the context of some kind of delayed integration, such as a version control branch merge or an upgrade to some third-party software.

Reviewers should be timely in their responses, and reassign the reviewed bug back to the programmer with approval or suggestions for changes. Once reviewed, the programmer can go ahead and integrate the fix.

Related To: Coordinate Efforts

Tests Capture Bugs

Problem

Code should not be changed unless there is a failing test.

Therefore:

When bugs are found, capture them in a test. Copy the code for the failing test into the bug report. Not the whole setup and fixture, but just enough to make the problem clear.

Discussion

If you have a bug, write a new case in your programmer tests that reproduces the bug, and fails because of it. The test needs to be as narrow as possible, at the unit testing level.