The Fine Art of Writing a Good Bug Report

In a speech at Quality Week '99, Roger Sherman, a Microsoft test manager, identified the leading cause of bug report closure as "unreproducible." This is a regrettable circumstance, since such bug reports waste precious time during tight development schedules, add absolutely nothing to product quality, and lead to frustration and bad feelings between development engineers and test engineers. Sometimes, these bug reports arise from transient or random events, inconsistency of tools and configurations between test and development, or a vague definition of "correct" behavior under the tested conditions, but many bug reports closed as unreproducible are unclear, misleading, or just plain wrong.

Fortunately, I have learned some tricks for writing great bug reports that get management attention, communicate clearly to developers, and get fixed. Not only do these techniques provide solid technical payoffs in terms of a greater proportion of bugs fixed, they also communicate to development and to management that testers are serious about helping developers fix bugs. Writing bugs reports using these methods on projects I have managed, only around one out of eight of bug reports are closed without a fix.

Applying the following ten tips will help you achieve better bug reports:

1. Structure. A tester who uses a deliberate, careful approach to testing, and takes careful notes, tends to have a good idea of what's going on with the system under test. When failures occur, he knows when the first signs of failure manifested themselves.
2. Reproduce. The tester should check reproducibility of a failure before writing a bug report. If the problem doesn't recur, she should still write the bug report, but she must note the sporadic nature of the behavior. A good rule of thumb is three attempts to recreate the failure before writing the report. Documenting a clean set of steps to reproduce the problem addresses the issue of reproducibility head-on.
3. Isolate. After reproducing the failure, the tester should then proceed to isolate the bug. This refers to changing certain variables, such as system configuration, that may alter the symptom of the failure. This information gives developers a head start on debugging.
4. Generalize. After the tester has an isolated and reproducible case, he should try to generalize the problem. Does the same failure occur in other modules or locations? Can he find more severe occurrences of the same fault?
5. Compare. If a tester has previously verified the underlying test condition in the test case that found the bug, the tester should check these prior results to see if the condition passed in earlier runs. If so, then the bug is likely a case of regression, where a once-working feature now fails. Note that test conditions often occur in more than one test case, so this step can involves more work than just checking past runs of the same test case. Also, if you have a reference platform, repeat the test there and note result.

  rex_black@acm.org

6. Summarize. The first line of the bug report, the failure summary, is the most critical. The tester should spend some time thinking through how the failure observed will affect the customer. This not only allows the tester to write a bug report that hooks the reader and communicates clearly to management, but also helps with setting bug report priority.
7. Condense. With a first draft of the bug report written, the tester should reread it, focusing on eliminating extraneous steps or words. Cryptic commentary is, of course, not the goal, but the report should not wear out its welcome by droning on endlessly about irrelevant details or steps which need not be performed to repeat the failure.
8. Disambiguate. In addition to eliminating wordiness, the tester should go through the report to make sure it is not subject to misinterpretation. Some words or phrases are vague, misleading, or subjective, and should be avoided. Clear, indisputable statements of fact are the goal.
9. Neutralize. Being the bearer of bad news presents the tester with the challenge of delicate presentation. Bug reports should be fair-minded in their wording. Attacking individual developers, criticizing the underlying error, attempting humor, or using sarcasm can create ill will with developers and divert attention from the bigger goal, increasing the quality of the product. The cautious tester confines her bug reports to statements of fact.
10. Review. Once the tester feels the bug report is the best one he can write, he should submit it to one or more test peers for a review. The reviewing peers should make suggestions, ask clarifying questions, and even, if appropriate, challenge the tester's assertion that the behavior is buggy. The test team should only submit the best possible bug report, given the time constraints appropriate to the priority of the bug.

A bug report should be an accurate, concise, thoroughly-edited, well-conceived, high-quality technical document. The test team needs to focus on the task of writing bug reports, and the test leads and manager must make it clear to each member of the test team that writing good bug reports is a primary job responsibility. Quality indicators for a well-tuned bug reporting process include:

- Clarity to management, particularly at the summary level;
- Utility to the development team, primarily in terms of giving the developer all the information needed to effectively debug the problem;
- Brevity of the bug lifecycle from opened to closed, reducing cycles where developers return poor quality reports for more information, leading to tester rework.

Improving the bug reporting process does require an effort, but provides significant payoffs. First, a crisp process improves the test team's communications with senior and peer management, which enhances the team's credibility and professional standing, and can encourage management to invest more resources in testing. Second, the smooth handoff to developers promotes positive relationships. Third, shorter bug lifecycles are more efficient, so the time invested up front writing a good bug report is repaid in time not wasted rewriting a poor bug report. These payoffs help the development process achieve better product quality through effective communication and efficient workflows.

  rex_black@acm.org