Checkpoint Technologies Whitepaper Series

Writing Effective Bug Reports

White Paper

Date: January 1, 2014

http://www.checkpointech.com

checkpoint TECHNOLOGIES Software Quality. Assured.

Table of Contents

Introduction

When defects or “bugs” are discovered in a software program, either in testing or by end users, they must be described to the software developer so that the developer can:

- Recreate the behavior
- Determine the root cause of the behavior
- Repair the software

Effectively describing defects in a bug report is one of the most critical tasks that a tester performs. Developers must evaluate and try to recreate every defect reported and this costs both time and money. Being imprecise in a bug report can cause developers to waste valuable time trying to determine the meaning of the report and cost testers time and effort in information exchanges with the developer seeking clarification. Bad bug reporting can also cause legitimate defects to be dismissed as invalid. When bug reports from a tester are consistently incorrect, imprecise or confusing, the tester may lose credibility with the development team.

This white paper outlines the characteristics of a bug report and ways that testers can ensure the bug reports they file are effective. While the end user can sometimes be excused for writing a bad bug report, they are not testers after all; there is really no excuse for a professional tester to be found guilty of this. Uncovering defects is the reason software testing organizations exist.

Most companies that develop software have defect reporting standards and defect management processes in place. These can be very helpful in ensuring that a bug report is effective. No matter what process, tool or standards are in place however, at some point there is going to be a big text box, be it in e-mail or a defect tracking system, in which the tester is going to have to explain the defect.

There are other essays and papers on this topic, but a review of the literature shows that developers have written the majority of them. Some of them have an almost pleading tone with the underlying message being “please don’t waste my time by sending me useless bug reports.”

This paper was written from the point of view of a tester. The opinions and techniques described are designed to help testers maintain their standing in a software development organization and foster good relationships with their development counterparts.

Note: There are many different terms used to describe problems observed in software behavior including: anomaly, bug, defect, error, issue, failure, fault, etc. There are also different schools of thought as to what the different terms mean and when they should and should not be used. Many of the definitions are differentiated by when the behavior was observed (i.e. before or after release to the customer) and if developer error was responsible. Regardless of either of these factors, when reporting issues, the same rules apply across the board.

For the purposes of this paper the terms “defect”, “bug”, “issue” and “problem” are used interchangeably. The definition used for the term “bug” is located in the next section.

What is a Bug?

Definition of a Software Bug

In their book *The Science of Debugging*, Matt Telles and Yuan Hsieh define software bugs as “things the software does that it is not supposed to do, [or] something the software doesn't do that it is supposed to.”[1]

This definition encompasses defects caused by problems in coding (e.g. an open pointer in a program written in C causes the application to crash), problems caused by infrastructure and hardware (e.g. the server runs out of memory when multiple users use a feature at the same) and situations where the software program does not match requirements or specifications.

Bugs in the System

The creation of the term “bugs in the system” is often erroneously attributed to Grace Hopper, who reported the case of a moth trapped in a mechanical relay on an early computer at the Harvard Computational Center in 1947. This term has actually been in use in engineering since at least the 1870s.

> *“It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and* [it is] *then that "Bugs"—as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.”*[2]
>
> *- Thomas Edison, 1878*

[1] Telles, Matt and Yuan Hsieh. *The Science of Debugging*. p. 5, Scottsdale: Coriolis, 2001

[2] Edison to Puskas, 13 November 1878, Edison papers, Edison National Laboratory, U.S. National Park Service, West Orange, N.J.,

Writing Bug Reports

A True Story (Names Have Been Changed to Protect the Innocent)

John Doe was a software tester for a company that wrote financial software. His peers in the testing group held him in high regard for his broad knowledge of the application and his ability to find problems in the software. The development team, on the other hand, held John in low regard, due to the imprecise and scant documentation in his bug reports.

Time after time, when John reported defects, the development team would spend time trying to figure out what the problem was. Ultimately someone from the development team would need to hoof his way over to the test lab and sit with John to watch him recreate the problem.

After this had happened many times, the developers' complaints reached the ear of Sue, a high level manager in the development organization. Sue saw the time the developers were spending on each of John's bug reports and was appalled. While most of the problems John reported did turn out to be legitimate bugs, the value of finding them was highly diluted by all of the man hours spent trying to recreate them. Sue asked that John be removed from the testing team. John was fired shortly thereafter.

The reason for including this story is to point out that *your ability to describe defects effectively is just as important, if not more so, than your ability to find them.*

Characteristics of a Bad Bug Report

Each of these sections starts by telling you *what not to do* first. Looking at some of the most common mistakes in bug reporting can help to keep you from repeating them. Each section will then describe ways to avoid creating a bug report with negative characteristics.

Vague and Unhelpful - "It Doesn't Work"

This is an obvious mistake to avoid. Submitting bug reports that say only things like "the program doesn't work" or "the report feature is broken" are completely useless and will serve to immediately destroy any credibility you have as a tester (and probably anger the programmer looking at the report).

Even if you are including additional details about the defect you discovered, it is best to avoid language like "broken" and "doesn't work." It is much better to phrase things in a non judgmental and sequential fashion such as "When I performed actions *a* and *b,* an error was triggered in the report feature, the error dialog said *c*."

This points to a fact that is unfortunate but true: *bug reporting is a very political activity within most software development organizations.* Some developers handle defect reports better than others. The response to defect reports can vary from the helpful developer who wants to understand and fix the issue to the defensive developer who views each bug report as a personal affront. Because of this, it is important that you keep your language neutral and avoid adjectives that have negative characteristics. It is also important to remember from a metrics point of view that as a tester, each bug you discover usually works in your favor. Some developers see defects reported concerning code they created as a mark against them.

Luckily, many software development shops realize that the mission of the tester is the same as that of the developer. The ultimate objective is to deliver bug free software to production.

Not Able to Recreate

Many testers have the best of intentions when they report a bug, but they do not include enough information for the developer to recreate the defect. Often times this is the result of the tester not being sure themselves of the steps that caused the defect to manifest.

A tester could detect a defect while executing a long set of test steps and not be sure if it was the test steps, data or environment that caused the issue to appear. Unless the tester spends some time repeating steps and attempting to recreate the defect, they may send a defect report that does not give the developers enough info to cause the failure to occur. The object of a bug report is to enable the developer to cause the software to fail. The best way to ensure a defect behavior can be recreated is to recreate it.

Before submitting a bug report do the following:

- Repeat the steps you were executing when the defect appeared to see if it happens again.
- If data was entered, try subtle variations and see if it still occurs
- If there are multiple methods for performing the operation that is defective, try them all
- Try your test on another machine to see if it happens on that machine too
- Document the steps you took to recreate the defect, take screenshots of each step
- Document computer settings such as operating system and screen resolution

The bottom line here is: *do your homework.* Submitting bug reports in a casual manner can have negative consequences. Before you submit a bug report, think about the fact that by doing so you are committing the company to spending money to investigate the defect.

If a defect occurs and you cannot recreate it, it is still okay to submit a defect report. Sometimes reports of intermittent bugs can be very helpful to developers. You should still include as much information as possible about what you were doing when the error occurred.

Not Specific Enough

While this is related to the two categories above, what this section is really describing is a defect report that contains all of the steps necessary to recreate the defect, but confusing or non-specific language that causes the developer to fail to see it.

For example, a bug report description may read:

I opened the application, then opened the reports window, then selected a report from the view report dialog. When I closed it, I got the following error message…

There are a few issues here. One is, there may be multiple points of entry into the reports window. Many applications will have a menu item, a toolbar button and a keyboard shortcut for reaching a specific destination. Be specific as to the method you used to get to a window, it can make a difference to the developer.

Second issue is that data is not specified. What report did you select? Did you try the same thing with a different report?

Also, there is the troublesome pronoun 'it'. The description above describes opening three different entities before closing 'it'. Did the error occur when you closed the application, the report window or the view report dialog?

Speculation Obscuring Observation

Some testers either were developers or fancy themselves as such. This can cause their bug reports to read like programming primers, and often they are wrong. For example, a bug report may read:

I selected item x from the menu and then entered value b in edit box c. When I pressed the submit button, the application crashed because the field is declared as a number instead of a string.

Are you sure that's why the application crashed? Putting this sort of speculation into a bug report can cause the developer to do unnecessary research and send them in the wrong direction. The developer may think there was something overt in the way the application behaved that led you to the conclusion, maybe an error dialog that indicated the field was declared as a number instead of a string.

If you have guesses as to why the defect is occurring be sure to qualify them with “this could be caused by” or “perhaps this is because” or even “I think”. This will alert the developer that this is you speculating, not stating a conclusion.

Avoid Sarcasm and Humor

Injecting sarcasm and humor into bug reports may seem hilarious to you when you are writing the report. It may not seem nearly as funny to the developer when he is struggling to get the defect backlog cleared before a release. While humor is an effective tool that helps defuse stress in many situations, it is recommended that it not be used during bug reporting.

Creating Effective Bug Reports

This section details a list of things you should do to create effective bug reports and avoid some of the common problems mentioned above.

Use Concise Language in the Title and Description

Describe exactly what occurred in a clear and concise manner. Don’t use the techniques taught to you in your Thursday night creative language class to make the bug report more gripping.

Give the bug report a title meaningful to the developer, for example:

“Entering Special Characters in SSN Field Causes Error”

Do not give your bug reports titles like “Data Error” or “System Crash”.

In descriptions, try to keep all of the things we talked about in the previous section in mind including:

- Avoid vague and unhelpful descriptions
- Avoid adjectives with negative connotations
- Avoid non-qualified speculation
- Be as specific as you can about the steps you executed

Include as Much Pertinent Information as Possible

When reporting a defect, include as much information as you think it is necessary for the developer to know. Include all of the steps you took to induce the behavior and the specific data you entered. As mentioned previously, be specific as to whether you selected a menu option or pressed a toolbar button.

Take screenshots of all the steps you took to recreate the defect and attach them to the defect report. Some screenshot and testing tools allow you to record movies. Record a movie of the process and the defect and attach it. Nothing helps a developer more than seeing the defect occur.

Include the version and build of the software you are using and the environment you are running the software in, along with information about the workstation you are using. There are system tools and features in testing tools that allow you to capture system information. This can aid a developer as well.

If you were running multiple applications when the defect occurred, reference what was running. Some defects in your application may be cause by conflicts with other software.

Many software development organizations use defect-tracking software. This software will often force you to enter environmental and software information.

When You See It, Document It

If you see a defect occur, capture information about it immediately. Don’t reboot your machine or kill the application process to see if this “fixes” the problem. You may end up taking these kinds of steps and then being unable to recreate the circumstances in which the problem behavior occurred.

Call a Friend

Try your defect description on another tester or coworker. First read it to them to see if they understand it. Then see if they can recreate the steps you are describing based on your description alone (no helping).

Reference the Requirements

If the defect you are reporting is a case where the application is not meeting requirements or specifications, be sure to reference the requirement or specification that is failing. The developer may indicate the application is functioning “as designed”, when in reality the developer’s design was flawed.

Requirements based testing indicates that any time the application does not meet requirements, a defect should be opened.

Offer to Demonstrate for the Developer

Offer to go to the developer's desk to help him try and recreate the problem. Having a good relationship with developers is helpful in testing. It can also help you see circumstances from the developer's point of view and give you a better idea of what they are looking for in bug reports.

Summary

Effective bug reporting is a skill that all testers must have. This paper was intended to highlight both things to do and not to do when reporting bugs. By creating effective bug reports, a tester can ensure that their reports will be given the attention they deserve and that their reputation is maintained amongst their peers.

Most software development organizations have standards for bug reporting, but as mentioned earlier, these standards cannot in themselves produce effective bug reports. Remember that the goal is to allow the software developer to see and recreate the failure.

The skills outlined in this paper may seem obvious to some, but their importance cannot be stressed enough. The things that are at stake when creating bug reports are time, money, a tester’s credibility and even their job (remember the sad story of John Doe). Keeping this in mind, always practice due diligence when reporting bugs and try to apply the techniques discussed herein.

About Checkpoint Technologies

Checkpoint Technologies, Inc. is a solutions provider with unparalleled expertise in the areas of Quality Assurance and software testing. Incorporated in January 2003, we specialize in all areas of Quality Assurance and software testing (with a focus in functional testing, application security testing, mobile application testing and performance testing). Our best-of-breed solutions and expert services enable you to deliver higher quality software applications to market faster! We provide staff augmentation, consulting services, training, mentoring, and software product sales and support throughout North America.

Checkpoint Technologies is a Micro Focus Software Partner, Micro Focus Software Support Partner, and Micro Focus Authorized Training Partner. In addition, we also are proud to be a partner with and offer solutions developed by Atlassian, Mobile Labs, Kobiton and Tricentis.

Checkpoint Technologies is Software Quality. Assured!

Please visit us at www.checkpointech.com.

Feel free to contact us today at:
Email: info@checkpointech.com
Phone: (813) 818-8324