# PRINCIPLES OF PROPER BUG REPORTING

Justin Spears

## ABSTRACT

This white paper provides a detailed synopsis of what makes a good bug report. This includes examples of how to write them and suggestions on proper formatting and details.

May 2016

## EXECUTIVE SUMMARY

*Testing leads to failure, and failure leads to understanding.*
*- Burt Rutan*

Filing a proper bug report is essential to product quality. Without good information the bug owner has nothing to go on. Think about asking for directions; you expect clear and concise pieces of information. Without such clarity you will be lost. The aim of this white paper is to provide clarity and direction for filing proper bug reports.

## THE DETAILS……

Anyone who has written a piece of software will undoubtedly recount tails of poorly constructed, dubious and questionable bug reports. It may have lacked specificity, been confusing, not have had proper data, didn't show how to reproduce the problem or simply didn't elaborate on what the problem really was. No matter why the developer thought the bug was bad; the point was they thought it was bad. To work effectively in the software development life cycle bug reporters must learn how to craft bug reports in a way that becomes logical, clear and irrefutable to the reviewer.

Think of filing a good bug like applying for a job. You only have a limited amount of time and resources to show a potential employer what you have to offer and that you are worth their time. Filing a proper bug requires the same mentality. You want to put the most useful information front-and-center. Make your bug report coherent and cogent enough to make the reviewer understand the importance of the problem and want to look into it. Just like you present yourself in the best possible light to a potential employer you want your bug report to be clear, professional and devoid of confusion.

Below I will lie out the basic guidelines of what makes a good bug report and provide examples.

### Clear and Concise Summary

Much in the same way a job interviewer judges you based on your attire the summary or title of your report is the first impression of this bug (and you as the submitter) to the receiver. Summary/titles needs to be short, precise, accurate and not generic. They cannot be too long but they need to provide enough specific information to initially understand whom to route this issue to. Some examples of good and bad titles follow:

**Good**: *Unable to create a password that contains foreign characters.*
**Bad**: *My desired password doesn't work.*

**Good**: *Using the side scrollbar creates delays when more than 3 pages are shown.*
**Bad:** *Scrolling down is slow.*

Precise and Accurate Descriptions

During that face-to-face job interview, after the initial handshake it is time to get down to business. Much in the same way interviewers want to understand what you know and how you can communicate the bug owner needs clear, precise, detailed and accurate bug descriptions. One does not need to write another version of *War and Peace* but limiting the detail to a single line isn't good either.  There are four key elements to a proper description that are essential to creating a useful bug.

1. **Steps to reproduce**

Every bug you file should ideally be reproducible. If you cannot reproduce the issue how do you expect the bug owner to do it? In most Software Development organizations time between releases is very limited and only priority bugs can get immediate attention. So in every instance providing detailed steps to reproduce the issue is key. Make sure to provide all details, even the trivial ones. Make sure to also include any required data to reproduce it (e.g. account names, strings, IP's, etc).

If you have taken any steps towards troubleshooting the problem now is the time to inform the bug owner. Please be as detailed as possible in what you have tried and the results of those experiments.

2. **Expected Behavior**

When filing a bug it is important to be clear on your expectations. Sometimes what you think should happen and what the developer thinks should happen are two different things. A good rule of thumb is to provide two lines, expected and actual outcome:

> *Expected outcome*: When pressing the "commit" button I expect a response window to open confirming my entries.
> *Actual outcome*: I pressed the "commit" button and the page refreshed.

3. **Consequences of the bug**

It may sound trivial, but it is important to describe what will happen if the user hits this issue. In many cases, when developing in isolation, coders simply don't see the holistic view as an end-user or tester does.

4. **Details about your environment**
While the environment you are working in is probably standard to you, that may not be the case to someone else. Providing a detailed understanding of your environment empowers the bug owner to determine if this issue is caused by or related to your specific environment. Information such as software versions, IP addresses, login credentials, etc can all be very helpful. If it does not violate your local security policies providing access information to your environment can significantly reduce time-to-resolution.

## Attachments, Logs and Screenshots

You're midway through your job interview now and you've impressed the interviewer with your breadth of knowledge. But now the interviewer, as they all do, wants to know if you can back up any of that smooth talking. In that same vane the bug owner needs some evidence to support your claims. This is the time to provide as much collateral material as possible. Screenshots are invaluable when describing UI-related issues. If you are describing a backend problem then logging is essential. In both cases having a solid understand of the version and configuration is critical. Consider filing a bug against Firefox.

If you encounter an error in Firefox while opening a page there are three things you need to provide. First, is a screenshot of the error message. This is probably the most obvious item to people. Second, any associated logging that is generated. Like any other application Firefox generates logs in its backend; these should be provided. Finally, the version and configuration information is vital. How many times has simply upgrading to a new version been the solution?

## Accuracy in categorizing fields

Whether at the beginning or end of your job interview there is undoubtedly some data that you will need to provide. It may be references, educational history, contact information or something else entirely. No matter what was requested it is critical that data be accurate. If an employer finds your information to be incorrect you've lost your chances to work there. Akin to the detailed job application is the categorized data necessary to properly file a bug. Within the confines of your department or group most enterprise companies are going to use a structured bug tracking system (such as Jira). Within this tracking system there are going to be fields that categorize your issues. Examples of these fields are Version, Component, Type, Priority, Severity and so on. It is important to mark these fields as accurately as possible. Now in many cases the bug submitter may not know exactly how to categorize something – that makes it even more important to get the remaining fields right. The more correctly categorized fields there are the more likely the bug will get routed, analyzed and solved efficiently. Nobody wants to watch emails fly back and fourth from the dreaded game of “bug pong.”

Comments and follow-ups
Like any good prospective employee after the interview ends you know it is good form to follow-up, provide thanks or maybe ask/answer a question. This is also good practice for any bugs you have opened. You should continue to follow-up on each one of your bugs, make sure the owner is not waiting on you for more information and be diligent about providing clarity when needed. This is especially true when a bug owner doesn’t understand the bug, and either delays or rejects it. By following up you better the chances for a quick resolution to the problem.

A bug tracker is only as good as the bugs that get filed. It may take a few extra moments of your life, but it's a worthwhile investment up front that will save time in the end. Following this simple set of guidelines puts you well on the path to creating bugs that leave no room for doubt or question. It makes the transition from tester to developer simple and increases the likelihood of having a higher quality product. For those who prefer a more succinct approach below is an example of a well-formatted bug.

## EXAMPLE BUG REPORT

**Title**:
Application crash on clicking the “create” button when creating a new user.

**Fields**:

Type: Bug
Priority: Critical
Affected Version: Release 5.1
Components: UI

**Description**:
When trying to create a new user there is a problem. Upon entering the information in the fields and clicking the "Create" button the application blue screens and becomes unresponsive.

Steps to Reproduce:
1. In the application choose the "create new user" button on the top right corner
2. In the new window fill in the first name, last name and username fields
3. Click the create button
4. Observe the blue screen and the error message related to "*ORA1290 Exception: Insert values Error…*" Full details and information can be found in the attached screenshot and application logs

After the error happened I tried the following with no change in status:
a. restarting my browser
b. rebooting the application
c. logging in as a different user

I considered entering the debug kernel but due to interference on other work was not able to do that. If you would like to access my system and perform that experiment we can certainly make arrangements to do so.

Outcome:
Expected: the new user would be created and I would be returned to the main page
Actual: the application crashed

Consequences:
No user accounts can be created thus rendering the application useless for anyone not sharing the default account. This creates both a scaling and security nightmare.

Environment:
Ubuntu 13.41 running Firefox v44.0.2
VPN 232, running IP range 153.22.18.x
Running all commands as a guest user
Access to lab can be provided; please contact me for the jumphost IP address and the login credentials.

**Attachments, Logs and Screenshots**
To this bug I have attached the following:
1. a screenshot of the application crash showing the error message
2. the application logs which show the actual error around line 1421

3. a screenshot of the version and configuration information

Is there any additional collateral material I can provide?