# Rapid Software Testing Guide to Making Good Bug Reports

*By James Bach, Satisfice, Inc. v.1.0*

Bug reporting is a very important part of testing. The bug report, whether oral or written, is the single most visible outcome of testing. The quality of your reports is probably the single most important factor that determines your credibility as a tester. This document describes bug reporting from the perspective of Rapid Software Testing methodology.

Throughout this guide I will distinguish between what is expected of "**supporting testers**" (anyone helping the test effort but not committed to the testing role or perhaps even that project) and "**responsible testers**" (those who assume the role of tester on a project and commit themselves to doing that job well).

## What is a bug?

A *bug* is anything about the product that threatens its value (in the mind of someone whose opinion matters). Sometimes these are called *defects*, but I don't recommend using that term, since it sounds accusatory, and it implies that testers may only report things that they are absolutely sure are wrong with the product. Bug is a time-honored and somewhat vague term, which suits our need to report things that are possible threats to the value of the product, even things we might be wrong about.

Bugs are not about our own personal feelings. You can use your feelings to find a bug, but then you better find a good reason why someone else would have the same feelings. That's because your feelings don't really matter. A tester is an agent for the people who *do* matter, usually, because the people who matter probably have other things to do than test software. Thus, we must learn about the people who do matter: the users, the customers, the bosses, society, and the standards that are relevant to *them*.

## What is a bug report? What are its elements?

A *bug report* is any report about what might be a bug in the product. Bug reports can be formal or informal, written or oral.

Underlying even the simplest bug report is a certain structure of thought that involves the following four elements:

- **An experience of the product.** Usually a direct observation of the product itself, but also possibly a simulation, or even a mental simulation based on a specification of some kind.
- **An understanding of the product.** Possibly grounded in shared experiences, cultural understandings, specific conversations or formal specifications.
- **An understanding of potential trouble.** You must be able to recognize a problem when it occurs. The means by which you recognize a problem is called an *oracle*. An oracle can be a principle, specification, example, tool, or even a person. All bug reports are based on some sort of oracle, and maybe many different oracles.
- **An understanding of the potential magnitude of particular troubles.** Just the fact that a behavior is a threat to the value of the product is not necessarily interesting. Your clients

need to know: is it a big bug or a little bug? You should be ready to say how important a bug it could be. This is related to how likely it is to be seen and how much damage it could do if it occurs. More on that below.

The most basic bug report is a statement to the effect that "here's something I think may be wrong with the product." A real bug report could be as simple as pointing at a screen and saying "Uh oh, look at that." In fact, that may be all you need to do in the case where you are testing for a friend standing next to you and you both have a strong shared mental model of what the product should be and do.

Even if you make a very informal bug report, be ready to make a more complete and explicit report, if questioned. You should anticipate the common questions that developers and managers may ask, such as:

### Basic Information (expected from anyone)

- What seems to be the problem?
- What exactly did you see happen?
- What were you doing right before you saw the problem? Were you doing anything else interesting at the same time?
- Have you seen the problem more than once? If so, what other times have you seen it?
- Did you take any screenshots or preserve any other data about the problem?
- What good reason do you have for thinking it is wrong?
- What version of the product were you testing? What platform are you using?

### Investigation Details (expected from responsible testers)

- Is it already reported?
- Have you tried to reproduce the effect? What are the steps to reproduce it?
- Have you tried simple variations of the input or operations that led to the symptoms? Have you tried on different platforms or in different modes?
- Have you checked to see that the product is actually installed properly and running? Is your system is properly configured?
- Did this problem exist in the earlier versions of the product?
- Do you have a specific suggestion of what should happen instead? How could it be better?
- How easy is it to make the bug happen? What is the worst consequence you can think of that could result from it?

Not all of these questions apply for any given bug, but your credibility as a tester depends on being able to answer them in the cases where they do apply.

## Bug Investigation

**Supporting testers** are generally not expected to investigate bugs beyond what is necessary to make a clear report.

**Responsible testers** are expected to investigate bugs to the point they can make clear, relevant, and concise reports. However, for puzzling or difficult to reproduce bugs, it is often is the case that the developer will have immediate insight into the underlying causes. For that reason, I recommend the *10-minute heuristic*: investigate a puzzling bug for no more than about 10-minutes before checking in with a developer about it (assuming the developer is available). If the developer is also puzzled, continue the investigation, otherwise, if the developer claims to know what is happening, just report what you know and move on to find the next bug. That will mean you are finding more bugs instead of making ever nicer-looking reports.

The goals of bug investigation are:

- Find a reliable way to reproduce the problem.
- Isolate the bug by identifying and eliminating extraneous factors, discovering the limits of the bug, and collecting evidence about its underlying causes.
- Generalize the report by discovering the broadest interpretation and impacts of the bug.
- Preserve relevant and necessary data that will make the developer's investigation easier.

Bug investigation often requires technical knowledge, product knowledge, and analytical skills that are beyond the scope of this guide to explain. That's why we don't expect much investigation to be done by **supporting testers** coming in to the project temporarily.

## Formal vs. Informal Bug Reporting

Consider three kinds of bug reporting: MIP'ing, formal reports, and black flagging:

- **MIP.** This stands for "mention in passing." To MIP a bug is to report it in a sentence or two by voice, email, or instant message. There is no expectation of formally tracking such reports, and there is no template for them. The main reason to MIP a bug is when you are fairly confident that what you are reporting is not a bug, and yet you have a suspicion that it could be. MIP'ing helps preserve your credibility because if a MIP'ed bug turns out to be a real issue, you can fairly say that you did report it, and because if it is not a bug, you can fairly say that at least you didn't create unwieldy paperwork for the team. MIP'ing is a very good way to work when you are pairing with a developer for early testing.

  MIP'ing is an excellent protocol for bug reporting when running a mass testing event with **supporting testers** when developers or other experts are in the room with them. The supporters then serve as "bug scouts", while the experts perform investigations and take responsibility for formal reporting, if required.

- **Formal Reports.** This means recording written bug reports in some sort of tracking system. Formal bug reporting is slower and more expensive than MIP'ing, but has obvious advantages when you are dealing with large numbers of bugs. Expecting **supporting testers** to do formal bug reporting may not be reasonable, but if they do, someone with responsibility will need to edit them. Poor quality formal bug reports can erode the credibility of the testing effort.

- **Black Flagging.** This means going beyond reporting a bug to the extent that testers raise an alarm about a serious underlying problem in the development effort. This may be necessary for safety or security bugs that can only occur when there is a breakdown of development discipline. Black flagging is for when you suspect that a bug is part of a much larger group of bugs that may not have been found yet, or may not yet have been put into the code.

### Elements of a Basic Formal Bug Report

It varies from project to project, but here are the most common fields found in a formal bug report:

- **Title.** A short statement that expresses the essence of the bug.
  - One sentence long; twelve words or less.
  - It must be distinctive, not generic (i.e. don't write "the product crashes"). The title should uniquely identify the bug among all the other bug report titles.
  - Try to put the actual problem right at the beginning (e.g. "Crash when logging in as admin" rather than "When logging in as admin, product crashes") because it's easier to read when looking at a list of bugs.

- **Description.** Any other information about the specific failure mode and behavior of the system that members of the team need to know about the bug.
  - Keep it short. Give reasonable details about the bug, but don't include information that everyone on the team will certainly already know. If the problem is very obvious (e.g. "Company name spelled wrong on the home page") then you hardly need to write a description.
  - Write in professional language. Don't use "SMS" jargon. Spell words correctly.
  - Provide steps to reproduce *if those steps are not obvious*. Don't provide steps that are obvious (e.g. "1. Connect to the Internet 2. Start the browser").
  - Indicate your oracles. That means say why you think this is a bug, *unless this is obvious*. Don't say silly things like "the product should not crash." That sounds insulting and it adds nothing to the report.
  - Note any workaround for the bug that you know about.

- **Version.** This is the latest version of the product in which you detected the bug. If you also checked earlier versions, note that in the description.

- **Environment.** The platform you were testing on. Typically this is your hardware, browser and operating system.

- **Attachments.** For all but the easiest to understand bug reports, it will be much appreciated if you attach screenshots of the problem, or even small videos. Also include links to any critical data files needed to reproduce the problem.

In addition to the basic fields, your bug tracking system may have other fields, as well. It will autofill the ID, Reporter, and Date Reported fields. Then there is Status, Severity, and Priority, which follow a protocol that is specific to your company and your project, so I won't discuss them here.

### Give the bug report a good focus.

- **Report the most compelling version of the bug.** Bug reporting is a kind of sales job. You must frame the report in the most compelling (yet truthful) way in order to maximize the chance that it will be understood and acted upon. Try to focus on the aspect of the bug that can have the most user impact, or the most negative public impact for your company.
- **Avoid reporting multiple problems in one bug report.** Unless multiple problems are probably the symptoms of one underlying fault in the product, they should be separated into distinct bug reports. This is because it's very easy for a developer to fix one problem, while accidentally forgetting to fix others that are listed in the same report.
- **Avoid reporting the same problem in multiple reports.** It is often difficult to tell whether two problems that seem similar are genuinely the same problem. So, make your best guess or consult with the developer to be sure.

## Assessing the Magnitude of a Bug

A tester is the first judge of how "big" the bug is. This is true even for **supporting testers**, to some degree. But for **responsible testers** it is a very important part of your work.

What makes a bug important? Basically four things:

- **How frequently does it appear; or how easy is it to make happen?** A bug that is seen often or by a lot of users is going to be more important, all other things being equal.

- **How much damage does it do when it occurs?** A bug that deletes data may be more important than one that merely creates confusion in the user interface. However, there are no hard rules about what specific symptoms constitute "more damage." Consider the importance of the user who is affected, and how upset they may be because of the bug.
- **What does the bug imply about other potential risks?** A bug may be especially important because it implies that there is a big problem in the development process itself that may have resulted in many similar bugs that are not yet found (see "black flagging", above).
- **What bad publicity might come from the bug?** Bad feelings and bad reputation can accrue from bugs even if the objective severity is not that bad. Consider how a bug might sound when people complain about it on social media. Consider how it might erode trust in your company's brand.

## Common Mistakes Testers Make in Bug Reporting

Watch out for these problems in your reports:

- **Poorly worded title.** The title is rambling, incoherent, generic, too long, or otherwise not representative of the substance of the bug report.
- **Wrong severity.** Testers usually make the first guess at setting a severity or priority for a bug. This is subject to change by developers or managers, but it's important for testers not to inflate or undervalue a bug by too much.
- **Reporting an unsupported opinion.** A personal opinion with insufficient grounding or evidence to support it is no basis to report a bug. The tester is not an authority; the tester is an agent for people who are authorities.
- **Reporting something that is not a problem.** Even if the bug is based on an oracle other than personal opinion, it may still be an incorrect oracle. The product may actually be intended to work the way that it does.
- **Not enough information to reproduce.** There is not enough information to enable the developer to verify the existence of the problem.
- **User error.** The report is based on a mistaken test, such as when an observation or operation was not done properly.
- **Reporting the wrong thing.** Sometimes testers will report a small problem that sits in the shadow of a much more important problem. This can happen when they don't take a moment to consider the bigger picture of the bug.
- **Annoying or disrespectful reporting.** The report is written in a manner that will irritate the developer or manager and erode credibility of the tester. This happens usually when the report is written in a sloppy way, or seems to denigrate the developer.
- **Terse report.** If there are not enough words, it's too hard to figure out what is being reported.
- **Unnecessary text.** Including information that is already common knowledge can make it seem like you are writing for the sake of filling in fields, rather than with the intent to communicate clearly.
- **Multiple reports.** More than one bug report packed into one record.