Submitting bug reports

In all likelihood, the output you’ll produce most regularly in your early QA career is the humble “bug report”. A bug report is where you write about an error you’ve found, with enough details to hopefully enable somebody to fix it.

It’s a very important output for the product in terms of making sure we’re building something good, but on a personal level, it’s actually so much more than that. In my opinion, each bug report you write is an opportunity for you to show your value - from day one. Failing to do this properly, for each and every bug, is a missed opportunity to advance your career. Moreover, submitting low quality reports can be outright unhelpful, and *will* make you unpopular.

Fortunately, getting this right is actually really easy, and we’ll look at some of the key things you should be doing in this chapter. The biggest rule though? You have to *care*.

Care about your bug reports!

By “care”, I mean you have to put the effort in. I’ve seen many examples of poor bug reports from people that I know could tell me what belongs in a good bug report if I asked them. Yet the reports they submitted were poor anyway. These were often people I wouldn’t have considered lazy, they just didn’t *take care* to write good reports. It goes to show that *knowing* what makes a good report doesn’t mean you’ll write them.

You really do need to take the *time* and make the *effort* required to submit good bug reports. Fail to do this at your own peril.

Let’s compare writing a bug report to giving somebody a present. You could find the perfect present, but if you tell the person you’re giving it to that you’ve left it to be collected from the middle of a busy motorway, during rush hour, during a torrential downpour, you’re causing pain, likely more than the present is worth, and they won’t be pleased with you. On the other hand, you can choose to lovingly wrap it, tie it with a pretty bow and hand deliver it with a card and a smile. This analogy is a little weird, but really not so far off. Executing your tests absolutely must result in easy, useful output that can be acted upon by other members of your team, or it could actually be worse than useless.

For a software test engineer, submitting bug reports should be your “bread and butter”. This should be such a core part of your skill set that you can nail it without even thinking about it. On a day where nothing else has gone right, you were late, it rained and you had no coat, the project is running behind schedule and you forgot to feed your cat, you should still be able to count on submitting high quality bug reports. When you’re at your absolute worst, the *feedback you give* should still be stellar.

The positive view of this is that consistently submitting high quality reports will help your team and earn you their respect and gratitude. I’ve seen it, many times, good and bad. In my early career I believe this aspect of my testing arsenal was probably the biggest asset I had to offer at a time when I was frankly still struggling to find my feet, as any newbie always is! If you want to do well as a tester, start by mastering this.

What is a bug report?

A bug report is a method of communicating to people that something is broken - or needs improving - in the thing that you are testing. In some projects, “submitting a bug report” might be as simple as writing a sentence on a post-it note and sticking it to a board. In others, it will be a comprehensive write-up that you enter into specialist software, such as JIRA or Bugzilla.

What is the point of the bug report?

Before we ask what information belongs in a bug report, we should ask what the point of a bug report is. This may sound basic, but it helps ground *why* you include the information you do. Hopefully you can suggest an answer for the point of a bug report yourself?

The most obvious aim of a bug report is to enable somebody to come along, and fix the thing that you said was broken. Great, let’s keep this firmly in mind, but let’s also think about other potential outcomes. Many bug reports don’t actually result in the bug being fixed.

Each of the possible alternate actions resulting from a bug report sheds some light on additional aims of the information you include. Why might a bug report not result in something being fixed, you ask? This could be for a variety of reasons. For instance:

- The thing you said was a bug, wasn’t actually a bug. Your bug report might say: “The word ‘Cheese’ on the screen is wrong. The correct spelling is ‘Cheeeeese’”. Of course, ‘Cheese’ is correct, and so in this case, your bug would be determined to be “Invalid”, and nobody would fix anything, since there is nothing to fix.
- You might have correctly pointed out something broken, but your team chooses not to fix it because it isn’t that serious, and/or there isn’t time before a release. Perhaps some pictures on a webpage have some very minor issues, and nobody minds very much.
- You might have correctly pointed out a bug which **is** serious, but your team still chooses not to fix, because they judge (or ideally know) that your customers won’t mind. This might happen if there is a way to work around the serious problem. For example: A very important button doesn’t work, but there’s another button on a different page which does the same thing. The customer would rather have the software as it is today, and have you fix the broken button later.

- The bug might initially be rejected because somebody else tries the thing you said was broken, and find it works when they try it.

There will be other reasons I haven’t covered here. What is important, is that any bug report you submit has to help people work out which course of action (fix/not fix, etc…) is the most appropriate, and enable them to carry that out. The step of deciding the appropriate actions for a bug report is commonly called “triage”, and often involves various stakeholders of the project.

Notably, one of the people that *might* have an action resulting from the bug report is **you**. If somebody fixes the problem, you could likely be the person who needs to go and verify that it has been fixed later on. Sometimes this could be days, weeks or months later, and you may have forgotten what the problem was. You’ll be relying on your report to remind you.

What should be in a bug report?

Before we move on, why not try answering this yourself? Try writing a list of all of the things you think you *might* want to include when describing a problem. Try to imagine a problem that would result in each of the resolutions we mentioned above, and what information might have helped reach or avoid that decision. Bear in mind that some of the things you should include will be different depending on the error or the situation in which you find it.

When you’re ready, move onto the next page to see my list of “things you should include”.

Things to include:

Depending on the bug, you’ll want to include some/all of the following:

- Summary: Roughly what is the problem?
- Severity: How serious do you think the problem is? Should somebody drop everything to read what you’ve written right now, or can it stand to wait? Severity is generally understood to be a product of *how bad* the problem is when you see it, and *how likely* it is to be seen.
- Steps: A detailed list of instructions that lead to reproducing the issue.
- Expected behaviour: What *should* have happened?
- Observed behaviour: What *did* happen?
- Version information: What version of the software were you testing?
- Environment information: What is relevant about your environment, that might change the results?
    - What version of Windows did you use?
    - Or which browser was involved?
    - Anything special about your machine?
    - Any other details of the setup that are relevant to your testing?
- Screenshots: A picture says a thousand words!
- Logs: Does the application keep a record of things that happen? To either confirm what you said, or offer more clues about what went wrong.
- Date/Time: Any logs you provide might cover hours or even days of use. Confirming *when* you saw the bug helps people know which log entries to look at.
- Other attachments: Was there any data or files you needed to perform the tests?
- Anything else? For instance:
    - How often have you seen it?
    - Is there a workaround(another way to do the thing that you couldn’t do with the steps above)?
    - Does it reproduce in other similar environments? Everywhere? Only your computer? Perhaps only on a Friday and if it’s raining?
    - When did it start happening?
        - If you’re testing a different version every day, was it new today? Or can you make it happen on older versions too?
    - What *does* work? (If 2 + 2 didn’t work on a calculator, does 1 + 1 work?)

Well done if you got a significant number of these, or if you thought of any that aren’t listed. The point of all of these answers is that they’re helping to paint a detailed picture of the issue, beyond just a single sentence which could be easily misunderstood.

The *exact* information to include will vary for each bug. For example, if we’re testing a physical calculator, you might want to include the serial number on the back so that people know *which* calculator you used. It might be a problem that only affects *that* calculator.

The question “What should be in a bug report” is often posed in interviews. The correct answer is that “all relevant” information should be included, consisting of things like the list above. The question then becomes “what is relevant”.

In general, things which are relevant are anything which could be used to enable:

- Developers to reproduce and fix the issue.
- Management to decide how serious the issue is, if/when the issue should be fixed, and whether it should prevent the release of the product.
- Other people who have experienced the same issue to identify that it is the same issue. This saves them from submitting a new report for the issue, and makes it easier for the team to track how frequently the problem is seen.
- Other people who have experienced a *similar* issue, which is not actually the same, to distinguish this issue from theirs, so that they do not mistakenly believe their issue has already been raised, and so would not raise it. It is often the case that different bugs result in similar visible behaviour, but require different fixes. We’ll look at an example of this shortly.
- You, or somebody else, to be able to confirm at some later point whether the issue has been fixed.

The skill is in selecting which information best meets those needs, whilst also considering the fact that too much information can distract people from the information that was important. It’s also worth stating that spending longer than is actually necessary on writing bug reports is time you could have spent being productive elsewhere, such as by finding more bugs!

In my experience, it is more common to see reports that do not include enough information than that include too much, but it is certainly not merely a question of quantity. For each bug report you should be asking which information is most likely to be useful.

Worked Example: Investigating an issue

Consider the following scenario and resulting bug:

- You are testing a new calculator application on a Windows machine.
- The application is installed using an “msi” installer - an “msi” is a file that you can “double click” to see a “wizard” which walks you through installing the application.
- The application can run on many different versions of Windows.
- You have been testing iterations of this application for a week.
- Each day, the developers give you as many as 3-4 different versions of the application to test. Each time, you have uninstalled the old one, installed the new one, and run tests on that. You test new features, but also run some tests to check nothing old is broken in the process - these latter tests are known as regression tests.

- The calculator has multiple modes which result in different displays (user interfaces) being shown to the user, such as normal, statistical, etc… which have different buttons available.
- The version you are currently testing is labelled version 0.26.
- This latest version is supposed to add support for doing “indices”. Indices are when numbers are multiplied by themselves a certain number of times - like 2^2 (2 x 2), 2^3 (2 x 2 x 2), 3^5 (3 x 3 x 3 x 3 x 3) and so on.
- The developer informs you that they had to rewrite some of the multiplication code, which was first added several versions ago, in order to add this functionality.
- This first test you run is for 2^2 (2 squared, which means 2 x 2), which correctly gave you 4.
- Your second test is to try 2 x 2 using the multiplication button rather than the 2^2 buttons. The calculator gives you the answer 8. This is unexpected, since 2 x 2 is actually 4. This is a bug!

How do you write the bug report? Let’s consider:

Before writing the bug report, you should attempt to reproduce the problem. Notably, in doing so, you will often realise that you have not understood the bug entirely, or made mistakes in your steps. This can save you and others time later on. Note: If the application records logs, it’s usually a good idea to grab them *right now*, before you do anything else, and when the product has literally just experienced the bug. You can potentially include these in the report later.

OK, so let’s imagine that you grabbed logs, and then tried to reproduce the problem, but this time the calculator correctly displayed “4” as the answer, even though it previously said “8”. Hmmm. This is different to the first time you ran the test. How curious... Why would it do that?

Confused, let’s suppose that you try the steps again, and now the calculator again incorrectly displays “8” for 2 x 2, like before. After a bit more investigation, suppose that you manage to establish that it always gives “8” as the answer to every second thing you try. No matter what you ask it - not just if you do 2 x 2 - for every second thing you ask, the calculator claims the answer is “8”.

This bug that I‘ve just asked you to imagine would be weird, for sure, and I won’t try to rationalise what might cause this to happen. For the sake of the example, let’s just imagine that that was what you found. I’ve certainly seen weirder things than this.

By attempting to reproduce the issue, we realise that the original problem you thought you saw (that 2 x 2 comes out as 8), does not actually describe the problem with the calculator. In fact, had you written the original steps in your bug, since the calculator only makes a mistake every second equation, if the developer simply typed 2 x 2 into the calculator as the first equation they did, they would have told you everything worked for them. They would not have seen the problem you did unless they tried two equations, and would not have known to do so.

This might be a contrived example, but it demonstrates the concept that attempting to reproduce an issue often reveals key details, and so can save time from misunderstandings and erroneous reports.

Worked Example: Writing the report (general method)

Having thoroughly investigated the issue in the last section, we have satisfied ourselves that we understand the bug. We are now ready to write the report. For this case, it would involve writing some of:

1: A one sentence summary of the bug.
2: The perceived severity of the bug (many bug-tracking applications have a box for this). This particular bug seems very bad. It is likely to be the kind of thing that would stop a release, and warrants fixing quickly. This would tend to attract a “Major”/“Critical” severity/priority, or similar.
3: Write up the exact steps for reproducing the bug.
4: Write the expected behaviour.
5: Write the observed behaviour.
6: Include version information: Here, this is 0.26.
7: You might include information about the machine you were testing on. For instance: Windows 7, 64-bit. On the face of it, it seems unlikely that using Windows 10 instead of Windows 7 is going to affect whether you can reproduce this bug, but it just might.
8: Include a screenshot. This application has multiple modes with different UI. A screenshot would categorically prove which mode you were in at the time you saw the bug. It might also show the evidence on screen that after typing 2 x 2, you saw 8.
9: Logs. If the application produces logs, these might contain error information which is useful to the developer.
10: Particularly if providing logs, the time of reproduction would be useful, since it can save a developer trawling through logs that might document hours of testing. Even knowing approximately when you saw it might save considerable time. The screenshot might also have helped here, I note, since it might include the time on your desktop!
11: Mention that this behaviour was definitely working in the previous version (0.25), and that this bug is therefore a regression.
12: Possibly worth including the fact that the new functionality in this build appeared to work (2^2 did correctly give 4 when you tried it, so long as it wasn’t the second equation!).

Worked Example: Writing the report (specific example)

All told, what I’d write here might look something like this:

Overview:
Every second equation gives “8” as the answer.

Priority:
Critical.

Repro:
1: Open a new calculator window.
2: Type any equation (for instance, 2 x 2)
3: Observe that the correct answer is given. (for instance, “4”).
4: Type any equation (for instance, 2 x 2 again, but you could use anything)

Expected behaviour:
Correct answer. For 2 x 2, this is “4”.

Observed behaviour:
“8”.

Environment:
Calculator version 0.26 (Adds support for indices)
Windows 7, 64-bit

Notes:

- Issue does not reproduce in version 0.25.
- Indices appear to work so far on this build (testing not yet complete).
- In “general” application mode. See attached screenshot.
- Logs attached. I had a quick scan, no obvious errors in the logs around the time I saw this: 6.15pm.
- Doesn’t just affect 2 x 2. I tried others. Try with 1 x 1, etc… you’ll see the same thing. First answer is correct but the second always says “8”, and so on.

Exercise:

Imagine another bug in this calculator application. If you’d prefer, you can instead pick a *real* bug you already know about in something you’ve seen in your daily life. Your favourite phone game? A broken bit of your car? The self-checkout machine at your local supermarket?

Imagine the steps you would take to confirm that the bug is what you think it is, and write a sample bug report, taking care to include all of the things that might enable somebody else to first see what is broken, and then hopefully to fix it as easily as possible. Try to imagine all the ways they might misunderstand you, or miss the bug by accidentally checking something different to you, and consider if there are ways your bug report can prevent this.

Key Points

- Knowing what belongs in a bug report is not enough. You have to care enough to actually make the effort required.
- High quality bug reports will likely make you *popular* and help your career, in addition to helping improve software.
- The purpose of a bug report is to enable decision makers to triage your bug appropriately, and then other people to carry out any follow-up actions as necessary.
- There is a potentially endless list of relevant information to include to help accurately capture the nature of the bug, and help in any resulting actions.
- The information you ought to include is anything that will assist in triage and any resulting actions for the bug.
- Sometimes “less is more”. Too much information, or information not written clearly, might distract from important information.
- Too much time spent investigating/writing can be bad too, since you could be doing something else productive!
- Finding a bug is just the beginning. Can you *reproduce* the issue? Do you really understand it? What *exactly* doesn’t work, and what *does* work? The issue may not be what you originally thought it was…