Roadblocks to Bug Reporting

Michael Stahl

Abstract

Testers sometimes find a bug, yet do not report it. The reasons vary and include technical and psychological reasons. This paper describes a list of such situations and proposes strategies to overcome them.

By making testers aware of these cases, they will be able to recognize when they take place, and apply the proposed solutions to avoid them.

The paper is based on my personal experience, and that of others.

1 Types of Roadblocks

Analysis of situations where I and others have encountered bugs but did not report them, show that there are three types of roadblocks that will cause a tester to leave a bug unreported:

- Roadblocks related to the work procedures in use by an organization
- Roadblocks related to tester's psychology
- Roadblocks related to the level of tester's confidence and understanding of "what testing is".

The following sections describe 11 such cases, sorted into the above categories. Each case contains a description, and a proposed strategy to deal with it effectively.

Note: while I believe the problems are universal, some of the solutions are based on the work-procedures used in the organization I work for. Implementation in other organizations may call for some tweaks.

1.1 Work-Procedure Related Roadblocks

Can't Reproduce = Don't Report

Description: An obvious bug is encountered (e.g. the application crashes), but the tester has no clear idea what caused this crash, and cannot supply a clear reproduction scenario. In these cases, some testers will refrain from reporting the bug, for a few reasons:

- There is no way to reproduce the bug – so it can't be fixed.

- The tester wants to avoid the inevitable argument with development, and the continued requests for additional data the tester cannot supply.

Suggested Strategy: Tweak the Bug Triage policies – and the bug-tracking system – to better deal with these situations.

- Add a state called "Suspended". Irreproducible – yet proven bugs are set to this state, marking them as bona-fide bugs that cannot be readily reproduced.
- Associate a counter field with each bug. Increment the counter each time the same issue is encountered by anyone. When the counter reaches a pre-defined limit, change the state to "active", and decide if the severity is high enough to justify the effort needed to "hunt down" the bug.
- "Suspended" bugs that are in this state for a pre-defined time without being reported again, are closed as "disappeared".

Integration Bugs

Description: Integration (in some organizations) is a combined effort of the development and test teams. Testers and developers work together in the lab, test interim "engineering builds", and attempt to bring the system to a stable and functional state. As bugs are found, the developer fixes them on the spot, and a new "engineering build" is created. Once the integration session is completed, the developer checks-in all the changes to the Source Control system, and an official code drop is created.

When a bug is found, the tester does not report it, since it gets fixed on the spot, or because it causes friction with the developer sitting just next to the tester. Some bugs, however, are hard to fix on the spot. The developer creates a workaround and integration work continues. Sometimes the session has to be stopped for whatever reason (e.g. scheduled meeting, end of the day, etc). When this happens, there is a risk that, if unreported, the bug will be forgotten and go unfixed.

Suggested Strategy: Find the middle ground. Define a work procedure for Integration work that is acceptable to both sides as "reasonable".

In my organization the rules are simple:

- If a bug is fixed on the spot and retest shows it's gone – don't report
- It the fix was delayed for whatever reason – report it.

An added rule is that we allow opening bugs on "engineering builds" but closure is only allowed on official code-drops. This guarantees that the fix has been checked-in to the main code branch.

Impossible Bugs

Description: Every tester has experienced this: As you are testing along, something VERY strange happens. Based on your knowledge of the system, this simply “COULD NOT HAVE HAPPENED”. You look at your colleague who is sitting next to you. Did you see that? It couldn't be true – right? You both nod your head and move on. What you have seen was so unreasonable – sometimes very difficult to describe, too – that there is no point reporting it. No one will believe it anyway. Everyone knows this can't happen.

Suggested Strategy: Since it DID happen, you need a way to prove (to yourself and to others) that it did indeed happen.
Following a recommendation by James Bach, I use a screen-capture tool called Spector (www.spectorsoft.com) to continuously take snapshots of my screen. The result is that I have, at any given time, a recording of the last few hours of my actions. With this data, I could, in many cases, find out what exactly happened by "replaying the past" second-by-second. This helps also in the daily testing work, as it saves the need to re-run the test to generate the needed screen-captures for a bug report.

There are other tools in the market – here are three (I have not tried them):
- Screenhunter http://www.wisdom-soft.com/sh/sh_free.htm - Free
- BBTestAssistant (www.bbsoftware.co.uk)
- TestExplorer (www.sirius-sqa.com)

Two Bugs in One Report

Description: There are many cases where two bugs are found in the same area - sometime on the same application window – and it makes a lot of sense to report them together:
Steps: A, B, C
Expected Results: X,
Actual result: Y and Z (both of which are incorrect).

It seems economical to report both in the same bug report. In any case, they need to be fixed at the same time, by the same developer. Opening two reports is adding overhead to everyone involved, and is sometimes perceived as trying to inflate the number of bugs submitted by a tester.

Suggested Strategy: Report two bugs. There is no better way to ensure that both bugs are fixed. Too many such cases end up with only one bug being fixed, and the other forgotten.
The bug reporting system can be improved to support these cases by allowing "copy and modify", so it is easy to create two bugs with almost identical reproduction steps. An option to link the two incident reports will alert the developer to the existence of closely related bugs that may be fixed together.

And if your organization is in the habit of counting how many bugs each tester submitted, STOP IT NOW. There are enough warnings in the literature against this management practice (e.g. [Kaner at al.], [Kaner], [Robinson])

"I'll Be Back"

Description: Sometimes it is very disruptive to stop the flow of tests and report a bug right when it is encountered. The tester jots down some notes about the bug, with a mental note to "report it later".
The problem starts when "later" never comes, comes when the tester forgot the bug details, or at a time when a new code drop was delivered and the bug will not be accepted since it is on a previous build.

Suggested Strategy: There is no magic here. One must make sure to "Be Back". Adopting a set of behaviors may help:

- Make a habit of not leaving for the day before reporting all bugs
- Collect any reproduction info or logs immediately when the bug occurs, so you can report it later
- Stop and report whenever possible – assuming this does not break the ongoing test

While it is clearly easier to state these rules then to follow them in dynamic and stressed situations, having these as the basic mode of operation, and deviating only when one must, will reduce the number of bugs that are "forgotten".

1.2 Tester's Psychology Related Roadblocks

Weaving a Fairy Tale

Description: Sometimes, a tester will encounter a bad program behavior and "rationalize" it as something that makes sense and is therefore not a bug. The root cause may be that the testers WANTS the application to pass the test, but cannot ignore the signal of failure. The tester will make a hypothesis regarding the cause of the problem (e.g. "I did something wrong”; “that's not how the application is designed") that explains away the failure, making it "not a bug".
This may be caused by the psychological phenomena called "Cognitive dissonance": discrepancy between what a person does (observes, in our case) and what they think or believe (the tester wants the application to pass).

Suggested Strategy: Testers need to adopt an attitude where they WANT the application to fail, and that any failure or strange behavior is a bug unless clearly proven to be an acceptable behavior. This is one of the cases where awareness about the phenomena will go a long way in preventing it.

"Stuckness"

Description: The term "Stuckness" was coined by Robert M. Pirsig in his book “Zen and the Art of Motorcycle Maintenance” [Pirsig]. Pirsig describes how a technical person gets "stuck" and frustrated when an unexpected roadblock prevents progress into doing "real work". This is something testers encounter often in the lab: The goal is to test some feature in the application, but Install does not work (or the setup fails, or a driver is corrupt, or…). Instead of getting test-work done, they are wasting time getting nowhere, fighting the "pre-requisites" parts of the test, feeling inadequate and stupid. The frustration level goes up, and no bugs are reported, since the "real" testing cannot proceed.

Suggested Strategy: "Stuckness" can be turned around once the tester diverts the attention to the blocking cause and see it as a goal by itself: "My goal is not to test feature X. My new goal is to test the Setup". Once this mental goal-change takes place, the tester becomes effective again, logging bugs against the problematic Install, or the corrupted driver. Next time around, this part of the system will work better, as the problems are fixed by development.

Frustration

Description: When a tester encounters too many cases where bug submission was accompanied by bitter arguments and sustained bruises, there is a risk that the number of submitted bugs will be reduced. This is especially true for cases where the failure is not clear-cut and may be a matter of point-of-view (e.g. usability). A reluctance to open bugs may also happen when an application has many open bugs against it, and opening yet another low-severity bug seems a waste of time since "it won't be fixed anyway".

Suggested Strategy: There are a few reasons why testers should open a bug even when the chance of it getting fixed is low:

- The total number of bugs reflects the quality of the application. Not reporting existing bugs creates a false sense of security
- While bugs may not be fixed on this version, they may be fixed the next time the relevant component goes through re-write. Non-reported bugs will never get fixed.

1.3 Tester's knowledge Related Roadblocks

"It’s the same Root-Cause"

Description: The tester encounters a bug. The tester recognizes that this bug is caused by the same problem that resulted in a previously identified bug – one that was already reported. The tester decides that since the root-cause will be fixed anyway due to the submitted bug, there is no point reporting this bug as well.

This behavior creates two risks:

- If the tester's conclusion about the "same root cause" is wrong, the second problem will not be fixed
- Even if this is indeed related to the same root-cause, the fact that a bug was not submitted creates a risk that the failed test will not be re-run to verify the fix.

Suggested Strategy: Testers must internalize the fact that, first and foremost, testers report symptoms. If something does not work, report it. If you can dig deeper and add additional information to the bug report – such as suspected root-cause – you have done a better job. But every time a tester sees a different symptom, the first step is to report it. While possibly generated by the same root-cause, it may expose that the implemented fix did not cover all the aspects of the defect, and not all symptoms have been removed.

"It’s in the Spec"

Description: When faced with clearly-written specifications, testers tend to accept what's written in it, even when it does not make sense. There is something about the "written word" that seems to make it "right".
As a result, when encountering a bad behavior, there is a risk that the tester will not report it if it is "as specified".

Suggested Strategy: The more testers are made part of Requirements and Design reviews, the less you will encounter this issue. When attending these reviews, testers are exposed to the large number of mistakes found in documents, which will make them less reverent of the written word. Testers will understand that it is within their rights to discuss and dispute the specs, and will therefore be more willing to open bugs when a behavior is "by the spec" but seems wrong.

Usability – It's Not My Job

Description: The tester runs tests of a certain feature, and encounters a usability problem that is not related to the tested feature. Some testers feel that usability is neither their field of expertise nor their ownership, and therefore refrain from submitting usability-related bugs.

Suggested Strategy: Testers should realize that part of their role is to represent the user. When a feature is cumbersome, even though it works, the user will be annoyed. The user may actually not figure out how to accomplish a task that looks trivial to a technically-oriented person (especially one who knows the system well and who is used to finding workarounds for these cumbersome tasks).
Once this realization takes place, and the organization is open to accept usability bugs from anyone, more such bugs will be submitted.

To a large degree, this principle applies not only to usability, but to any part of the application that is owned by someone other then the tester observing the bug. The attitude should be that ANY observed bug is reported by the person encountering it. It is recommended that the feature's test-owner is consulted before submission, to ensure the observed behavior is indeed a bug. If the owner cannot be readily contacted (e.g. the owner is in a different geography), report the bug anyway. Always copy the owner on the bug report, to keep the owner updated with anything that has to do with the feature.

2 Additional Notes

Apart from the suggestions in the previous section, there are a few more ways to encourage testers to report bugs, by making bug-reporting less painful:

- Use state-capture tools such as vLog (http://www.vapisoft.com/) or similar (my organization uses an internally-developed tool), so that collection of system information, logs and screen captures is less time consuming
- Use screen recorders on a regular basis
- Configure your bug-reporting system to remember the values of certain fields, on a personal basis (e.g. if I always use the same setup to test, I want the system to remember the setup information I used on my last bug report).
- Allow easy Copy & Modify of bugs. This way a tester can save a lot of time when reporting bugs whose reproduction steps are overlapping.
- Manage the Bug Triage meeting etiquette – do not allow personal attacks of testers by development (or vice-versa).

A word of Caution

While testers should be encouraged to report any bug they see, they should also be made aware that there are cases where a lengthy bug-hunt is not productive.

Take this for example: A bug is encountered. The bug does not reproduce easily, and the tester understands that even if it did reproduce, it will be classified as a low severity one, with a small chance of being fixed. How much effort should be invested in finding a reproduction scenario? If it takes too long (e.g. a day or two), it is probably not worth the effort. The time can be better used to test for more significant issues. On the other hand, a tester may encounter a bug that does not reproduce easily and isn’t sure of the impact and exposure if it does reproduce. In this case, the tester should consult with someone more senior to determine the possible exposure if this bug escapes.

Use common sense (a good advice anytime…)!

A word of Motivation

Dave Winer, a Fellow of the Berkman Center for Internet & Society, Harvard Law School, wrote in his "Scripting News" blog : "To create a usable piece of software, you have to fight for every fix, every feature, every little accommodation that will get one more person up the curve. There are no shortcuts. […] it happens because you fought for every inch" [Winer].

Understanding this is a first step in overcoming bug-reporting roadblocks.

References

[Pirsig] Zen and the Art of Motorcycle Maintenance, Robert M. Pirsig, Bantam Books, 1974, Ch. 24, p. 250 .
[Winer] http://www.scripting.com/2002/01/12.html, Dave Winer.
[Kaner at al] Lessons Learned in Software Testing (Lesson 66), Cem Kaner, James Bach, Bret Pettichord, John Wiley & Sons Inc. New York 2002.
[Kaner] Don’t Use Bug Counts to Measure Testers, Cem Kaner, Testing & Quality magazinr, May/Jun 1999 (Vol. 1, Issue 3) http://www.stickyminds.com/s.asp?F=S5133_MAGAZINE_2 ,
[Robinson] It’s Not About the Bugs, Harry Robinson, http://www.stickyminds.com/s.asp?F=S6838_COL_2

Acknowledgements

Thanks are due to Lonny Baskin (Intel) for review and valuable comments on this paper.

This paper was first published in the Proceedings of Conquest 2008 conference (https://www.isqi.org/en/conquest.html)