# Use the Rejected Defect Ratio to Improve Bug Reporting

https://www.stickyminds.com/article/use-rejected-defect-ratio-improve-bug-reporting

By **Michael Stahl** - April 22, 2019
**Share URL**

**Summary:**
There are many metrics to measure the effectiveness of a testing team. One is the rejected defect ratio, or the number of rejected bug reports divided by the total submitted bug reports. You may think you want zero rejected bugs, but there are several reasons that's not the case. Let's look at types of rejected bugs, see how they contribute to the rejected defect ratio, and explore the right ratio for your team.

There are many metrics to **measure the effectiveness of a testing team**. One such metric is the rejected defect ratio, which is the number of rejected bug reports divided by the total submitted bug reports.

There are three categories of rejected bugs:

- Irreproducible bugs
- Incorrect bugs
- Duplicate bugs

We'll look at these types of rejected bugs, see how they contribute to the rejected defect ratio, and explore the right ratio for your team of testers. (You may think you want zero rejected bug reports, but I'll make an argument for why that's not the case.)

Let's start with the bugs themselves first.

## Irreproducible Bugs

There are two types of irreproducible bugs. The first is a bug that is indeed hard to reproduce. It may be a bug that occurs as a result of an interaction among a few parameters, some of which you are not even aware of.

For example, let's say you ran a number of tests in succession, and one of the tests, unknown to you, changed a configuration parameter from its default value, A, to some other value, B. The bug happens only when the configuration parameter holds value B, and the input value is C. When attempting to reproduce the bug, most likely you would want to start from a known state, so you would initialize the system (or maybe do a clean install). Consequently, the bug will not occur, as the configuration parameter now holds the default value, A.

Another case of this kind of irreproducible bug is where the test indeed uncovered a defect, but some data is missing from the reproduction information: a step, a specific input value, or an understanding that the bug occurs only under a certain order of actions. The result is that following the reproduction steps don't cause the bug to occur.

In both the above cases, however, there is indeed a defect in the product!

The second type of irreproducible bug is those that don't reoccur because there is no bug. The tester may have seen something and misinterpreted it, or the system used for testing may have suffered from some problem such as a defective hardware component, an incompatible driver, or incorrect permissions settings. Attempts to reproduce the bug on a well-built system fail.

Both these types of bugs are commonly closed in bug report systems as "Rejected—can't reproduce."

## Incorrect Bugs

These types of bugs are the cases where the tester thought the product should behave in a certain way and reported a bug when the product behavior did not meet the expectations. However, a deeper study of the requirements reveals that the tester's expectations were wrong and the product functioned correctly. The product was right and the tester, who was not familiar enough with the requirements, was wrong.

These bugs are commonly closed in bug report systems as "Rejected—not a bug" or "Rejected—by design" (that is, the behavior is following the design).

## Duplicate Bugs

These are bugs that someone reported already and someone else reported again. A **bug is a duplicate** only if the symptoms are the same. If the root cause is the same but the symptoms are different, it is not a duplicate!

These bugs are commonly closed in bug report systems as "Rejected—duplicate."

## How Rejected Bugs Affect the Team

Obviously, an incorrect bug creates waste: the time that the tester invests in reproducing the bug and reporting it, the time that the bug triage participants invest in reading and understanding the bug, and the time that developers invest in trying to reproduce an irreproducible bug or in fixing (and unfixing) something that did not need fixing to begin with.

Apart from the fact that the rejected defect ratio, or RDR, is a measure of inefficiency of the test team, it also says something about the professionality of the testers. A bug that can't be reproduced due to missing details in the report means that the testers were not meticulous in the report and didn't bother to ensure that the bug reproduces reliably by following the steps they wrote. Alternatively, for bugs that reproduce infrequently, the testers failed to make a note about the low reproduction rate in the bug report.

An incorrect bug shows that the testers are not fluent or don't fully understand the product's requirements. A duplicate bug shows that the testers did not do the minimal search in the bug database to check if the bug was already reported. Alternatively, it means that the tester who reported the bug first did not include the proper keywords in the title to make it easy to find by other testers.

When someone rejects a bug I reported, I get insulted. As far as I am concerned, I was called unprofessional. On one hand, it means I will fight for my bugs. When my report gets rejected, I usually go through several steps:

- I check again that the bug reproduces on my system and update the reproduction steps if I missed anything
- If my misunderstanding of the requirements was due to an ambiguous requirement or incorrect documentation, I will insist that the bug be marked as a documentation bug and closed only when the documentation is improved
- If I think that the product's behavior, while meeting the requirements, is incorrect, I will argue about the requirements with the architects and the developers and try to convince them that the requirements need to be updated (I am, after all, representing the customer's view!)
- If the bug is rejected as a duplicate, I will make sure it was not tagged that way due to a "same root cause" claim

On the other hand, it makes me careful. If I am not totally sure that something I spotted is a defect, I will invest some more time before reporting. I often will ask a colleague if I understand the requirements correctly, or I will check whether the bug reproduces on someone else's setup.

## The Case against No Rejected Bugs

A test team should monitor and strive to reduce the level of RDR. The question is, what level of RDR is a good goal?

On the face of it, it seems that 0 percent is a good target, but I disagree. I think that a certain level of RDR is actually healthy because when the RDR is too close to zero, the test team suffers from problems that are no less disturbing than a high RDR.

A test team will have to invest effort in order to reach a very low RDR. Every rejected bug will be analyzed to find what went wrong, and every tester who reported a rejected bug will need to explain what happened and how this mistake can be avoided in the future. The result will be that testers report only bugs that they are absolutely sure about.

If they notice a behavior that they think hurts the product's usability, they will prefer to accept the behavior, rather than having to justify why they opened a bug on something that was defined as a requirement. If they have proof that a bug occurred but don't have a good reproduction scenario, they will prefer not to report it; they really don't need the grief. If they encounter a low-severity bug, they may decide not to report it because low bugs are not always fixed, so why risk reporting something that may be eventually rejected?

In short, striving for a very low RDR generates unhealthy stress in the test team and increases the chance that some bugs will go unreported.

We want testers who not only report clear-cut bugs, but also warn the project of any situation that seems suspicious. We want testers who place high importance on making sure no bug escapes, even at the cost of some duplicate reports—better that than testers who spend an hour checking if the bug they just identified was already reported, in fear of the consequences of redundant reporting. We want testers who feel comfortable questioning the written word of the architecture or requirements specs, even if it means some of their bugs will be tagged as "not a bug."

We want testers who are not afraid to make a mistake every now and then. That means we need an equilibrium, and some level of RDR is considered reasonable.

## Finding the Optimum Rejected Defect Ratio

My rule of thumb for an RDR is 15 percent. This value is based on my experience working with a test team that everyone agreed was a good and effective team. This was our RDR during a number of consecutive projects, while another team that worked on the same projects and in parallel to us—though was less knowledgeable about the product and considered less effective—had a 30 percent RDR.

I don't think there is a strong justification to this value except my gut feeling. It's certainly not scientific. I won't be arguing too much with a team that targets 10 percent or 20 percent, but I think that tolerating 30 percent or setting the goal at 5 percent are both problematic.

Eventually it's a local decision that should be made by the test team based on the product, the team's expertize level, the development model, the quality of the development team, and more. I do strongly suggest that you track your RDR and decide if you need to do something active about it. If it's too high or too low, you may want to consider a corrective action.