Investigating Bugs: A Testing Skills Study

By James Bach
Principal Consultant, Satisfice, Inc.

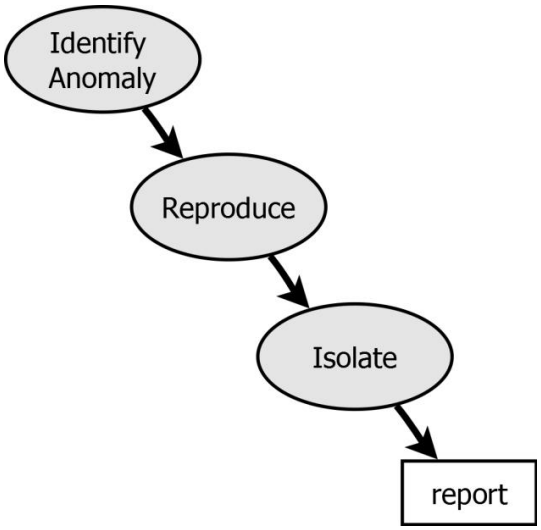
Ask any experienced tester how he does his work, and the answer is likely to be extremely vague ("Um, you know. I use my experience to... Um... black box the test case plan and such..."), or extremely false ("Our testing consists of detailed formal test procedures that are derived from written requirements"). Forget about bad testers, even *good* testers are notoriously bad at explaining what they do. Doing testing, describing testing, and teaching testing are all different things. No wonder that the IEEE testing standards are a joke (a very old joke, at this point), and based on talking with people involved in the upcoming ISO standard, it will be no improvement.

If we truly wish to develop our craft toward greater professional competence and integrity, then before we can worry how testing should be, we must be able to say how it is. We must study testers at work. Let me illustrate.

Years ago, I was hired by a company that makes printers to help them develop a professional testing culture. Instead of bringing in all my favorite testing practices, I started by observing the behavior of the most respected testers in their organization. I divided my study plan into segments, the first of which was bug investigation.

The organization identified one team of three testers (two testers and a test lead) that had a great reputation for bug investigation. This team was responsible for testing paper handling features of the printer. I wanted to see what made them so good. To get the most accurate picture of their practices, I became a participant-observer in that team for one week and worked with them on their bug reports.

Stated Procedure for Investigating Bugs



I sat down with George, the test lead, and asked him how he did bug investigation. He said, "Don't ask me to change anything."

"Good news, George," I replied. "I'm not that kind of consultant." But after some feather smoothing, he did answer:

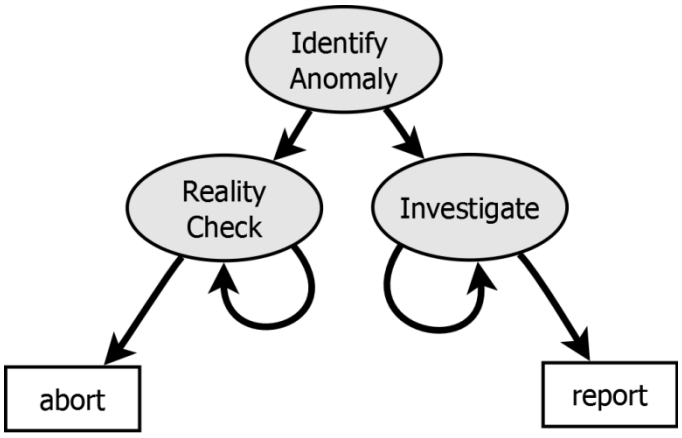
1) *Notice if automation checks flags a problem.* The automation system alerts the testers that something did not perform as expected.
2) *Reproduce the problem.* The tester executes the test again to recreate the symptoms of the problem.
3) *Isolate the problem.* The tester edits the test script, cutting it down to the minimum required to exhibit the problem.
4) *Pass the problem to the test lead for investigation and reporting.* The tester delivers the edited test script to the test lead. The test lead investigates the problem to determine, as best he can, its dynamics and causes. The test lead then writes the bug report and submits it.

This description is typical of how testers claim to investigate and report bugs. It's only unusual in that the test lead performs the investigation and writes the actual bug report. However, there was another way in which this description was unfortunately typical: *it's not true.*

I knew it couldn't be true, because it's a process description that anyone in that company would claim to use, yet only this team was respected for the quality of its work. There must be something more to their process that I wasn't being told.

Sure enough, during my observations and further conversations with the team, I found the actual process used in the team to be much more sophisticated and collaborative than their stated process. Their actual process ranks among the best I have seen at any company. (I first said that in the year 2000, and it remains true in the year 2011).

Observed Process of Investigating Bugs



What I observed in practice was an exploratory investigation carried out by the whole team. When an anomaly worth investigating was spotted by a particular tester, the other two came over and

they engaged together in two parallel loops of inquiry: *investigating the bug* and *questioning the status and merit of the investigation itself*. There were two possible outcomes, aborting the investigation or reporting the bug. The actual bug report was written by the test lead.

Part I: Identification

1. Notice a problem (during an automated check or any other situation).

   It is a general practice in the industry to construct tests that have specific expected results. This team took that idea further. Although they did establish certain specific expectations in advance, they also looked at whatever happened, as a whole, and asked themselves if it made sense. This extended even to the pattern of clicks and whirrs the printer made as it processed paper, and the timing of messages on the control panel. I call this the *Explainability Heuristic*: any unexplained behavior may be a potential bug, therefore we attempt to explain whatever happens.

2. Recall what you were doing just prior to the occurrence of the problem.
3. Examine symptoms of the problem without disturbing system state.

   Prior to starting a full investigation of a problem that may be difficult to reproduce, the testers capture as much volatile information as they can about it. This includes reviewing their actions that may have triggered the problem and examining the problem symptoms while disturbing the state of the printer as little as possible.

   During identification, the testers transition from a defocused behavior of observing whatever might be important to the focused task of investigating specific states and behaviors.

Part IIa: Reality Check Loop

The tester must decide whether to pursue the investigation or move back into open testing. So, prior to launching a bug investigation, and repeatedly during the investigation, these questions are asked.

1. Could this be tester error?
2. Could this be correct behavior?
3. Is this problem, or some variant of it, already known?
4. Is there someone else who can help us?
5. Is this worth investigating right now?
6. Do we know enough right now to report it?

The test lead is usually consulted on these questions before the investigation begins. But testers apply their own judgment if the test lead is not available. Contrary to George's first description of how his team worked, the testers on his team used initiative and routinely made their own decisions about what to do next.

In the event that an investigation is suspended because of the difficulty of reproducing it, it may still be reported as an intermittent problem. Whether or not it's reported, the testers will preserve their

notes and be on the lookout for the problem as they continue testing. Some investigations go on like that for months.

Part IIb: Investigation Loop

Once it's determined that the anomaly is worth looking into, the investigation begins in earnest. As I observed them, bug investigations were not a linear execution of predefined steps. Instead, they proceeded as an exploratory process of gathering data, explaining data, and confirming explanations. The exploration was focused on reproducing the problem and answering certain key questions about it. When they were answered well enough, or when the amount of time and energy spent on the problem exceeded its importance (that's what the reality check loop is all about) the investigation ended and the bug was reported. Sometimes investigations continued after making an initial report. This was done so that the developers could begin to work on the bug in parallel with testers' efforts to give them better information.

The investigation process is marked by a series of focusing questions that are repeatedly asked and progressively answered:

1. How can the problem be reproduced?

The testers not only reproduce the problem, they try to find if there are other ways to make it happen. They progressively isolate the problem by discerning and eliminating steps that are not required to trigger it. They look for the most straightforward and general method of making it happen. They also seek to eliminate special conditions or tools that are not generally known or available, so that anyone who reads the bug report, at any later time, will have the ability to reproduce the problem.

2. What are the symptoms of the problem?

Apart from identifying and clarifying its obvious symptoms, the testers are alert for symptoms that may not be immediately obvious. They also look for other problems that may be triggered or exacerbated by this problem.

3. How severe could the problem be?

The testers try to analyze the severity of the problem in terms of how it would affect a user in the field or create a support issue for the company. They look for instances of the problem that may be more severe than the one originally discovered. They look for ways to reproduce the problem that are most plausible to occur in the field.

The testers also consider what this kind of problem may indicate about other the potential problems not yet discovered. This helps them assure that their test process is oriented toward areas of greatest technical risk.

4. What might be causing the problem?

The most interesting element I observed in the team's process of bug investigation is their application of technical insight about printer mechanisms (both hardware and firmware) to guide

their investigation of the problems. In the course of investigation, the testers did not merely manipulate variables and factors arbitrarily. They investigated systematically based on their understanding the most likely variables involved. They also consulted with developers to refine their understanding of printer firmware dynamics.

Although there is no set formula for investigating problems, I observed that the testers relied upon their knowledge of printer mechanisms and their experience of past problems to organize their investigation strategy. So, maybe that's the formula: learn about how the printer works and pay attention to patterns of failure over time.

Part III: Reporting

Although I participated in bug investigation, I did not personally observe the process of writing a bug report in this team. The testers reported that they sometimes wrote a draft of the bug report themselves, but that all reports were edited and completed by the test lead.

Supporting Factors that Make the Process Work

Bug Investigation Philosophy

Apart from the process they follow, I found that there was a tacit philosophy of bug investigation in the team that seems to permeate and support their work. Here are some of the principles of that philosophy:

- We expect testers to learn the purposes and operational details of each test.
- We expect testers, over time, to gain a comprehensive expectation of the behavior of the product, and to follow-up on any anomalous behavior they detect at any time.
- Each bug is investigated by all members of the team.
- Bug investigation is primarily our job, not the developers. If we do our job well, then developers will be able to do their jobs better, and they will respect us for helping them.
- Testers should develop and use resources and tools that help in bug investigation.
- Ask for help. Someone else may know the answer or have an important clue. Seek advice from outside the team.
- We expect testers to use initiative in investigation and consult with the test lead as they go.

Individual Initiative and Team Collaboration

During the period I observed, the testers in the team treated each bug investigation as a group process. I had seen this before, and rarely since. They also consulted with testers outside their team, and with developers. Their attitude seemed to be that someone in the next cube may have information that will save them a lot of time and trouble.

The testers also showed personal initiative. They did not seem worried about crossing some forbidden line or running afoul of some corporate rule during the course their investigation. They appeared to take ownership of the problems they were investigating. The test lead told me that he encouraged initiative in his testers, and that he expected the testers, over time, to learn how all the tests worked and how the printers worked. In separate interviews, the testers confirmed that

sentiment, and stated that they felt that the resulting working conditions in their team were better than in most other teams they had served on at that company.

Observed Skills

I saw each of the following skills exhibited to some extent in each of the testers in the team. And in my opinion, the method of the investigation used in the team requires competence in these skills.

- *Skepticism.* Skepticism might be called the fear of certainty. It can be seen as central to the challenge of thinking scientifically; thinking like a tester. Good testers avoid sweeping claims about the product, because any claim may be refuted with the execution of the next test.
- *Performing an open investigation.* An open investigation is a self-managed investigation with general goals and few explicit constraints. An open investigation involves coordinating with clients, consulting with colleagues, collecting information, making conjectures, refuting or confirming conjectures, identifying issues, discerning and performing tasks, and reporting results. An open investigation in conjunction with testing is commonly called "exploratory testing."
- *Understanding external and internal product functionality.* Bug investigation requires a sufficient understanding of both external and internal workings of the technology. This knowledge is gained over time and over the course of many investigations, and through studying documentation, exploratory testing, or by observing other testers at work.
- *Consulting with developers or other testers.* Vital information needed to investigate problems is scattered among many minds. Good testers develop an understanding of the network of people who may be able to offer help, and know to approach them and efficiently elicit the information they need. In the case of developers, testers need the ability to discuss and question software architecture.
- *Test factoring.* When anomalous behavior is observed in the product, the ability to isolate the factors that may be causing that behavior is at the heart of the investigation process. This includes insight about what factors may be causally related, the ability to form hypotheses about them and to relate those hypotheses to observable behavior.
- *Experiment design.* Testers must be able to reason about factors and find methods of controlling, isolating, and observing those factors so as to corroborate or refute hypotheses about the product.
- *Noticing problems.* A tester can know how the product should function and yet still not notice a malfunction. Being alert for problems, even in the middle of investigating other problems, and even in the absence of an explicit and complete specification, is a skill by itself. This requires a good knowledge of applicable oracles, including tool-based oracles.
- *Assessing problem severity.* This requires understanding the relationship between the technology, the hypothetical user, the project situation, other known problems, and the risk associated with problems that may lie hidden behind the one being investigated. This skill also requires the ability to imagine and articulate problem severity in terms of plausible scenarios.
- *Identifying and using technical documentation.* Bug investigation often requires spot learning about the product. With printing technology, that can mean poring through any of

thousands of pages of technical documents. Testers need to know where and how to find relevant information.

- *Recording and maintaining information about problems.* The testers must deliver information about a problem in an organized and coherent form in order for the test lead to confirm it and write the report. This includes the ability to make and maintain notes.
- *Identifying and using tools.* Tools that may aid testing are scattered all about. Enterprising testers should be constantly on the lookout for tools that might aid in the execution of tests or diagnosis of problems. Testers must have the ability and initiative to teach themselves how to use such tools.
- *Identifying similar known problems.* In order to know if a similar problem is already known, the testers must know who to check with and how to search the bug tracking system. This also requires enough technical insight to determine when two apparently dissimilar symptoms are in fact related.
- *Managing simultaneous investigations.* Rarely do we have the luxury of working on one thing at a time. That goes double when it comes to investigating intermittent problems. Such investigations can go on for weeks, so testers must have the ability to maintain their notes and report status over the long term. They must be able to switch among investigations and not let them be forgotten.
- *Escalation.* Since these investigations are largely self-managed, it's important to know when and how to alert management to issues and decisions that rightly belong at a higher level of responsibility.

Case Study: The Frozen Control Panel

This is an example of an actual investigation in the team that took place while I watched. It appears to be typical of other investigations I had been told about or personally observed. The important aspect of this case is not the conclusion— we could not reproduce this problem— but rather the initiative, teamwork, and resourcefulness of the testers. This investigation is documented in as much detail as we could remember in order to provide a feel for richness and flow of an exploratory testing process.

1. While Clay was running one of the paper handling tests, he encountered a printer lockup. Clay called Ken and James over to observe and assist.
2. Clay had been running an automated check that included many steps. After executing it once he started it again. This time, it began executing, then stopped, apparently waiting for a response from the printer. At that point Clay noticed that the printer was frozen.
3. Clay asked Ken if he knew about the problem and whether he thought the problem was worth investigating.
4. Without resetting the printer, Ken examined the surrounding symptoms of the problem:
    - Check control panel display (display showed "Tray 5 Empty" continuously).
    - Check ready and data light status (both were lit and steady).
    - Open and close a tray (display did not react; engine lifted the tray).
    - Open and close a door (display did not react; engine performed paper path check).
    - Try control panel buttons (display did not react to any buttons).

5. Ken and Clay examined the test output in the terminal window and discovered that the test harness tool had stopped during its initialization, before any script code had been executed.
6. After a brief conference, Ken and Clay decided that the problem was worth investigating and conjectured that it may be due to an interaction between the timing of control panel display messages and messages sent to the printer.
7. Ken performed a cold reset of the printer.
8. Clay restarted the test tool. The problem did not recur.
9. Clay edited the test script down to the last few operations. He executed the modified script several times. The problem did not recur.
10. To test the hypothesis that the problem was related to the timing of alternating "READY" and "TRAY 5 EMPTY" displays on the control panel, Ken and Clay coordinated with each other to start executing the test tool at various different timings with respect to the state of the control panel display. No problem occurred.
11. We went to see a firmware developer on the control panel team, and asked him what might account for these symptoms. He seemed eager to help. He suggested that the problem might be a deadlock condition with the engine, or it might be a hang of the control panel code itself. He also suggested that we review recent changes to the firmware codebase, and that we attempt to reproduce the problem without using the test tool. During the course of this conversation, the developer drew some basic architectural diagrams to help explain what could be going on. We questioned him about the dynamics of his diagram.
12. The developer also conjectured that the problem could have been leftover data from a previous test.
13. Then we went to see a fellow who once supported the test tool. With his help, we scrolled through the source code enough to determine that all the messages displayed by the tool before it halted were issued prior to contacting the printer. Thus, it was possible that whatever happened could have been triggered by the first communication with the printer during tool initialization. However, we were unable to locate the tool routine that actually communicated with the printer.
14. Because the control panel locked up with the data light on, we knew that it was unlikely to have been in that state at the end of the previous successful test case, since that case had reported success, and left no data in the printer.
15. We looked for a way to eavesdrop on the exact communication between test tool and the printer, but found there was no easy way to do that.
16. We called upon another tester, Steve, for help, and together we wrote a shell script, then a Perl script, that endlessly looped while executing the test tool with an empty script file. At first we thought of introducing a random delay, but Clay argued that a fixed delay might better cover the timing relationships with the printer, due to the slight difference between the fixed time of the test and the presumably fixed response time of the printer.
17. We ran the script for about an hour. The printer never locked up.
18. While watching the control panel react to our script, Clay saw a brief flicker of an unexpected message on the display. We spent some time looking for a recurrence of that event, but did not see one.
19. We then went to visit a control panel tester to get his ideas on whether a problem like ours had been seen before, and how important a problem it could be. He advised us that such a

problem would be quite important, but that he knew of no such problem currently outstanding.

20. Clay independently searched the bug tracking system for control panel problems, and found nothing similar, either.
21. After several hours of all this, we were out of easy ideas. So, we called off our investigation until the test lead returned to advise us.

The Study of Skill is Difficult

It's quite difficult to study the anatomy of a practice, and the skills that practice requires. You can't know at first exactly what to watch, and what to ignore. Anthropologists learn to watch behavior for long periods of time, and to relentlessly consider the possibility of researcher bias. And just the act of studying a set of skills makes people nervous and possibly change their behavior. I had to agree not to release any information about the progress of my study until I cleared it with the people I was studying.

Still, even a modest one-week study like this one can have profound positive effects on the team. When I gave this report to the team for approval, the team was a bit stunned at how much my description differed from their self-description. One of the testers asked me if he could staple it to his résumé. Perhaps there is even more depth to the skills of bug investigation than I have identified so far, but this is the sort of thing we must begin to do in our field. Observe testers at work and go beneath the grossly general descriptions. See what testers really do. Then maybe we can truly begin to build a deep and nuanced vision of professional software testing.