How Did I Miss That Bug?

Peter Varhol and Gerie Owen

peter@petervarhol.com; gerie@gerieowen.com

Abstract

How many bugs have you missed that were clearly easy to spot?

Testers approach all phases of testing hampered by their own biases in what to look for, how to go about setting up and executing tests, and how to interpret the results. These biases cause less than outstanding testing performance and data interpretation in an environment where test teams expect results to be cut and dried. The data is good by itself, but what we measure, and what decisions we make about the data, are driven by our own understandings and misunderstandings of the project and the goals of testing.

This paper will give testers and their managers an understanding of how their own mindsets and biases influence their testing. Testers and test managers will learn how to use this understanding to develop ways of managing their thinking and improving their testing. Using principles from the social sciences, such as Kahneman's[7] framework for critical thinking and Chabris and Simons [2] findings on attention, perception and memory, we will demonstrate to you that you aren't as smart as you think.

We'll show you how to improve your test results by understanding how you think and the role that biases play in testing. We'll discuss the balance between scripted and exploratory testing and how to use each effectively, and we'll show how test managers can help their teams maintain their focus individually and as a team. Finally, we'll provide tips for managing your biases and focusing your attention in the right places throughout the test process so you won't miss that obvious bug.

Biography

Peter Varhol is a well-known writer and speaker on software and technology topics, having authored dozens of articles and spoken at a number of industry conferences and webcasts. He has advanced degrees in computer science, applied mathematics, and psychology, and is currently TestStudio Evangelist at Telerik. His past roles include technology journalist, software product manager, software developer, and university professor.

Gerie Owen is a Quality Assurance Consultant who specializes in developing and managing test teams. She has implemented various Quality Assurance methodology models, and has developed, trained and mentored new teams from their inception. Gerie manages large, complex projects involving multiple applications, coordinates test teams across multiple time zones and delivers high quality projects on time and within budget. In her everyday QA Life, Gerie brings a cohesive team approach to testing. She has also presented at several conferences and authored articles on testing and quality assurance.

1. Introduction

How did I miss that bug? We have often asked ourselves and our teams to consider that question. Software testers frequently miss bugs that are clearly easy to spot. This costs organizations financially, and negatively impacts an organization's reputation when these bugs escape into production. Testers approach all phases of testing hampered by their own biases in what look for, how to go about designing and executing tests, and interpreting results. These biases can result in incomplete testing and incorrect data interpretation in business environments where the results must be correct. The data is good by itself, but what we measure, and what decisions we make on the data, are driven by our own understandings and misunderstandings of the project and the goals of testing.

Daniel Kahneman and Amos Tversky [6] developed the idea of cognitive bias as a pattern of deviation in judgment from their research, which showed people's inability to think critically in complicated situations. People tend to use heuristics, or rules of thumb, to make decisions when the subject matter is complicated or when time is limited. Heuristics are usually focused on one aspect of the problem while ignoring others.

Our preconceived notions as testers about the application under test, the requirements, and the skill of the developers all affect our testing. Some of the biases that especially impact testers include the representative bias, the curse of knowledge, the congruence bias, the confirmation bias, the planning fallacy, and anchoring. We will discuss these in detail a little later.

What is going on when we miss the obvious bugs, the ones that are literally staring us in the face? This, too, is likely attributable to a bias — inattentional blindness, a psychological lack of attention, not associated with any vision deficits. Christopher Chabris and Daniel Simons [2] demonstrated this bias in their famous invisible gorilla test, in which subjects were so fixated on a specific task that many didn't see a person in a gorilla suit run across the screen. As testers, we become so focused on executing our test cases that we sometimes fail to see the obvious bug.

So how do we use the concepts of bias and preconceived notions to become top performing testers? Rather than allowing our biases to hamper our testing and cause us to miss bugs, we should plan and execute our testing in recognition of the fact that we do have these biases and preconceived notions. We should add additional time to our estimates and include negative test scenarios in our test planning. Although we "know" that one developer's code is "always" full of bugs, and the other developer's code "never" has bugs, we should not execute fewer tests on the presumably better developer's code.

As testers, we can help each other prevent biased testing by peer reviewing test results or running each other's test cases. In addition to executing our test cases, we should perform exploratory testing. Exploratory testing prior to running our test cases is especially useful because we have not yet made any assessments, and potentially developed biases, about the quality of the product.

How do we make sure we see the "Invisible Gorilla"? As testers, we need to approach our testing holistically; we need to focus our attention on determining if this is a quality product as opposed to just tracing our test cases to requirements and executing all of our test cases.

2. Relevant Research

In order to understand how we miss bugs, we must understand how we test. What is software testing? In its most basic form, software testing is making judgments about the quality of the software under test. It is a very complex task, involving not only objective comparisons of code to specifications but also subjective assessments regarding usability, functionality etc. It follows then that failure to find a particular bug can be viewed as an error in judgment by the tester. Therefore, to determine how testers miss bugs, we need to

understand how humans make judgments, especially in complex situations. Here is a review of the most relevant research.

In “Maps of Bounded Rationality: Psychology for Behavioral Economics,” Daniel Kahneman [3] provided a model of bounded rationality. When people need to make a decision in a complex situation, they tend to create boundaries or condense the issue into something that they can handle within their thought processes. In his model of bounded rationality, Kahneman categorizes our thought processes into distinct components, specifically, System 1 and System 2.

- System 1 thinking is intuitive, quick and emotional and is applied during our initial reactions to situations.

- System 2 thinking relies on reason, rules and objectivity and is applied when more analysis is needed, for example when calculating the answer to a math problem.

System 1 and System 2 thinking often can be in conflict and lead to biases in decision-making.

In “Judgment Under Uncertainty: Heuristics and Biases,” Kahneman and Tversky [6] examine three heuristics that people use to make judgments. Heuristics are guidelines or “rules of thumb” that people use to make decisions quickly, especially in complicated situations. The authors describe how using heuristics can lead to predispositions in our decision-making. These heuristics include:

- Representativeness

- Availability of instances or scenarios, and

- Adjustment from anchor

These heuristics are used to explain many systematic biases in judgment in complex situations.

We use representativeness when we compare current situations to previous situations. Sometimes what has happened in the past is a good guide to future decisions, but we may not realize that the situations weren’t similar enough to make the same decisions.

We use availability of instances when we make decisions based on information from our own recent experiences or those of others. For example, we may make health and lifestyle decisions based on how many of our relatives have died of strokes.

We use the anchor bias when we compare to what we are given as standard, for example manufacturers’ suggested retail price. The anchor tends to invite a comparison with other values when we should be making a more informed judgment. Just because our last project had a certain number of defects, for example, we shouldn’t use that as a standard for judging the quality of future projects.

In “The Invisible Gorilla: How Our Intuitions Deceive Us,” Chabris and Simon [2] describe their famous “invisible gorilla” experiment that showed how focused System 2 thinking could cause the inattentional blindness bias. Furthermore, they showed that people could actually be blind to their own blindness. In the experiment, subjects were asked watch a video of a basketball game and count the number of passes between players. During the game, a gorilla walked across the basketball court. Over 50% of the subjects were so focused on counting the passes that they failed to see the gorilla!

3. Prevalent Biases in Software Testing

There are several biases that most frequently influence testers, and specifically impair the ability of the tester to find bugs. Let’s take a look at some of those biases.

3.1 Representative Bias

The representative bias is used when we judge the likelihood of an occurrence in a particular situation by how closely the situation resembles similar situations. Testers may be influenced by this bias when designing data matrices, perhaps not testing data in all states or not testing enough types of data. For example, a tester might decide that if the code works for a new order and an order in process, why test it with a completed order?

3.2 The Curse of Knowledge

The curse of knowledge is when we are so knowledgeable about our subject matter that our ability to address it from a less informed, more neutral perspective is diminished. This affects software testers when they develop so much domain knowledge that they fail to test from the perspective of a new or novice user. Usability bugs are often missed due to this bias.

3.3 The Congruence Bias

The congruence bias is the tendency of experimenters to plan and execute tests on just their own hypotheses without considering alternative hypotheses. In software testing, this bias is often the root cause of missed negative test cases. Testers write test cases to validate that the functionality works according to the specifications and neglect to validate that the functionality doesn't work in ways that it should not. For example, if the specification is that a field should accept only alpha characters, the tester must also validate that the field does not accept numeric. If a tester does not create a test case for this, the congruence bias may be why this test case was missed.

3.4 The Confirmation Bias

The confirmation bias is the tendency to search for and interpret information in a way that confirms initial perceptions. In software testing, testers' initial perceptions of the quality of code, the quality of the requirements and the capabilities of developers can impact the ways in which they test. For example, testers may test a less experienced developer's code more thoroughly than a more experienced developer because finding more bugs in the less experienced developer's code will confirm the tester's expectations.

3.5 The Planning Fallacy

The planning fallacy is the tendency to underestimate how long it will take to complete tasks. We are all optimists in that regard. Software testers often fall prey to this bias when estimating test planning tasks and test execution cycles. Often, test managers will plan or agree to a schedule that barely allows time to complete all the test cases. When unexpected issues such as late code deliveries or environmental or access issues arise, the test execution falls behind schedule. Risk based testing may be implemented; testers may be rushed or encouraged to work overtime resulting in missed bugs.

3.6 The Anchoring Effect

The anchoring effect is the tendency to become fixated and rely too heavily on a piece of information which may cause us to reject other ideas or evidence that contradicts the initial information. Software testers do this often when they validate code to specifications exclusively without considering ambiguities or errors in the requirements.

3.7 Inattentional Blindness

Inattentional Blindness is the tendency to miss obvious inconsistencies when focusing specifically on a particular task or feature set. This happens in software testing when testers miss the blatantly obvious bugs like misspellings while concentrating on executing their test cases. Another example is when testers are so focused on validating the bugs delivered in a release that they miss regression defects.

Gerie did this. She was testing a Smart Grid application and the associated devices including programmable thermostats and load control devices. When a "critical event" was input, the devices were show a red signal. When Gerie tested a new release, she totally missed that the critical event didn't activate the red signal and change the pricing because she was so focused on verifying the bugs fixes turned over in the release.

4. How biases and preconceived notions negatively impact software testing

Because test managers and testers are affected by so many biases and preconceived notions, this can negatively impact the effectiveness of the entire test process and decrease defect removal efficiency. When software testers, influenced by the planning fallacy, underestimate the time for the test process, the test process begins already compromised. In the previous section, we discussed the ways in which biases influence individual testers. In this section, we will examine how these biases impact the test process. In this paper, we review the test process within the waterfall methodology systems development lifecycle, however; the same biases apply to the test process within an agile framework.

4.1 Requirements Development Phase

In the requirements phase of the System's development lifecycle, software testers commonly perform ambiguity reviews on the requirements documents. As testers perform their ambiguity reviews, preconceived notions about the design of the application may affect their ability to spot ambiguities. The **curse of knowledge** may come into play especially during requirements reviews for upgrade projects. Having already tested previous releases of the application, the testers may have become subject matter experts and miss obvious ambiguities. Also, testers may make assumptions about what is not specified as a requirement. Without realizing it, testers may fill in the missing pieces and their completions may not be correct.

4.2 Test Planning Phase

Test planning may be negatively impacted when testers are influenced by the **confirmation**, **representative** and **congruence** biases. When testers develop test cases according to their initial impressions of the application under test and the project in general, they are being influenced by the confirmation bias.

When test data matrices are being designed, the **representative** bias comes into play. Testers rely on their experiences with past projects when they make judgments when determining the data types, data states and integration points that should be included in the data matrix.

When testers miss negative test cases, it may be that the **congruence** bias is at work. If software works as we expect, we sometimes fail to consider circumstances where it shouldn't work that way.

4.3 Test Execution Phase

When testers concentrate on executing their test scripts and test managers concentrate on test coverage, they become too focused and miss obvious defects. When schedules slip and risk-based testing is implemented, testers attempt to test what they, the project team or the stakeholders deem to be the most critical functionality with the highest risk. In doing so, they perform minimal exploratory testing and miss potentially critical defects. These misses may be attributed to **inattentional blindness** and **anchoring**.

4.4 Project Closure

In the project closure phase, testers have an opportunity to analyze what parts of the test process did and didn't work well. In this phase, root cause analysis may be used to determine why certain bugs were not

found in testing. The root cause analysis includes *why* bugs were missed, however, we don't usually focus on *how*. If we focus on *how*, we miss bugs, we may find that testers' biases had an impact.

5. Why we aren't as smart as we think; how we develop biases and preconceived notions

So why is it that we aren't as smart as we think? The works of Kahneman and Tversky,[6] Ariely, [1] and Chabris and Simon [2] have shown that we aren't as smart as we think because of our bounded rationality which is defined as our need to use heuristics and frameworks when faced having to make complex decisions.

But why is it that we develop biases? Does our level of intelligence matter? Surprisingly enough, West, Meserve and Stanovich [4] at James Madison University showed that the more intelligent people are, the more susceptible they are to biases. Intelligence matters, but in itself, does not explain why we develop biases.

Is it because we don't realize that we have biases and preconceived notions and that we are employing them in our decision-making process? In "*Thinking Fast and Slow,*" Kahneman [7] admits that all of his research on the topic has not enabled him to make unbiased decisions. So although we may or may not realize that we employ biases in our decisions, knowledge of our biases may not change our decision-making approach.

An answer to why we aren't as smart as we think comes from the research of West, Meserve and Stanovich [4] which showed that we can easily identify biases in the decision-making process of others but we neglect to factor in our own biases. We actually have a bias about biases! West et al. [4] called this the bias blind spot. We evaluate our own decision-making process differently than we evaluate how others make decisions.

6. How to design an approach to effectively manage the way we think during the test process

Now that we understand why we are so susceptible to biases and preconceived notions, how do we effectively manage the way we think throughout the test process? Although the research shows that we cannot prevent biases and preconceived notions through awareness and understanding, we, as individual testers, can attempt to manage our thought processes as we approach our test projects.

In their working paper, "How Can Decision Making Be Improved," Milkman et al. [4] review the work of several researchers which suggests various ways of invoking System 2 thinking (reasoning and objectivity) more heavily than System 1 thinking (intuitive and emotional). These include evaluating multiple options simultaneously rather than accepting or rejecting each option individually, making choices when under less cognitive load and making decisions in groups rather than individually.

Will an increased focus on System 2 thinking improve testers' effectiveness at finding bugs? We believe not. In software testing, our test methodology already focuses us and requires us to use System 2 thinking. Let's review the test process:

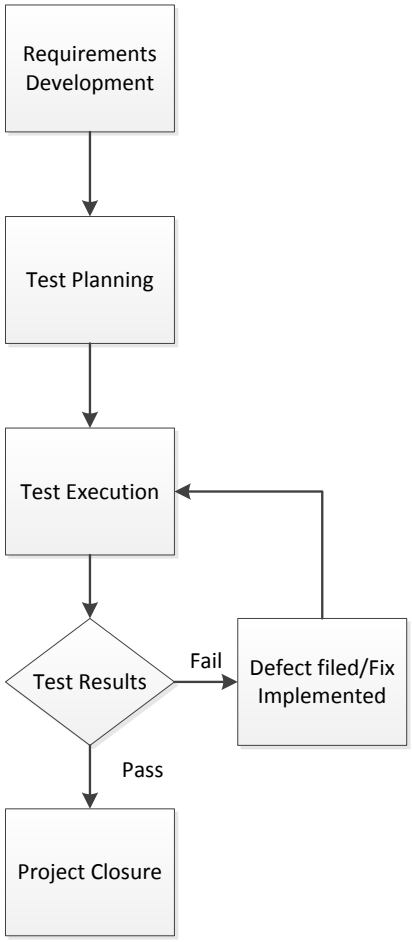

Figure 1. The testing lifecycle.

Requirements coverage, traceability and data matrices, and test case execution metrics are the tools of our trade that are designed to prevent us from missing bugs. These tools and processes are very useful in validating that the code is built to specifications and that all requirements have been coded and tested. Test methodology is the analytical framework of testing; it naturally invokes our System 2 thinking and places the tester under cognitive load. Haven't you left work exhausted after a day of writing or executing test cases?

We use the functional specifications to determine how the code should work and write test cases based on what should be the expected results. If we don't have functional specifications, we use business requirements, workflow training documents or even conversations with customers, developers or Project Management to develop our expected results. Then we compare the functionality of the application to our expected results. The determination of whether the actual results match the expected results becomes an objective assessment.

However, does validating that the application functions according to specifications mean that the application is bug free? Does 100 percent test coverage with all test cases passed mean that the application works as the business customer intended? We all know that the answer to both of these questions is not necessarily. Now the question becomes, how do we find the bugs that prevent the application from working as the customer intended. We believe this requires us to refocus on System 1 thinking (intuition).

Heuristics, when used with oracles, are quite valuable in invoking System 2 thinking. For example, if a typical user is our oracle, we might test workflows and find bugs that we might not find by executing our test cases. How many times have you reported a bug to which the developer responds: “The user would never do that!”?

Users make decisions about using applications based on look and feel as well as ease of use. Since users and potential customers use their System 1 thinking to make these decisions, it follows that to test effectively, we, as testers, must also use System 1 thinking and anticipate the emotions of the user. There is no better way to perform this testing than to consider our emotions and what they might be telling us about the application under test. For example if we are feeling frustrated and anxious, perhaps there is a performance or usability issue.

Finally, how can we find the obvious bugs, the ones we miss due to inattentional blindness? An answer becomes clear. At specific times during our testing, we need to focus less. In addition to our test cases, we need to perform exploratory testing and play with the application. We must use our intuition, in a controlled way, and when we see something we can’t believe we are seeing, we need to believe it and test it further.

7. How managers can increase their teams’ effectiveness by managing biases and preconceived notions

While testers can design an approach to manage their own biases and preconceived notions by using oracles and heuristics that employ more System 1 thinking, test managers can increase team effectiveness by fostering an environment in which the testers feel comfortable and empowered to use System 1 thinking.

Kahneman’s [7] research shows that people engage System 1 thinking in situations requiring intense mental effort, when they are emotionally engaged positively or negatively and when they feel empowered. Kahneman’s research [7] also shows that when people lack subject matter knowledge and when they believe in the power of intuition, they are more likely to use System 1 thinking in decision making.

Managers can create this type of an environment in many simple ways. Test schedules should be planned with time allotted to exploratory testing, preferably at the beginning of the test cycle before testers develop preconceived notions regarding the quality of the code.

Managers should provide a safe environment where testers are encouraged to take risks. Rather than requiring that each tester execute a predefined number of test cases each day, testers should be encouraged to test creatively, going beyond the predetermined test suite. Test managers might reward testers based on the importance or criticality of the bugs they find. Testers could be assigned to execute test cases that they did not write as an effective way of offsetting each other’s biases.

8. Summary

The quality assurance profession can help to prevent bugs that are missed due to testers’ biases and preconceived notions by promoting and encouraging the use of System 1 thinking throughout the test process. This may require a paradigm shift to our standard test methodology.

1. Testers need to understand that bias plays a large part in what we test and what we do not, and therefore what we miss, including those “obvious” defects. Understanding the types of bias will help us ensure we are considering more possibilities.

2. Management should not insist solely on automation/scripting to find the obvious bugs. Using exploratory testing is a valuable and necessary part of any testing function that wants to deliver a high quality product.

3. As an industry, we must shift our focus from requirements coverage based test execution to a more intuitive approach. Exploratory testing and business process flow testing should become the norm rather than the exception. To do this, we should experiment with new testing frameworks where risk-based testing is executed through targeted exploratory testing and is balanced with scripted testing to achieve a more effective approach to quality.

.

References

1. Ariely, D., 2009. *Predictably Irrational: The hidden forces that shape our decisions*. New York: Harper Colins Publishers.

2. Chabris, C and Daniel Simons, 2010. *The Invisible Gorilla: How Our Intuitions Deceive Us.* New York: Crown Publishers.

3. Kahneman, D. (2003). *Maps of Bounded Rationality: Psychology for Behavioral Economics*. The American Economic Review, 93(5), 1449-1475.

4. West RF, Meserve RJ, Stanovich KE , *Cognitive sophistication does not attenuate the bias blind spot.* J Pers Soc Psychol. 2012 Sep;103(3):506-19.

5. Milkman K, Clough, D., & Bazerman, M. (2008) *How Can Decision Making Be Improved?* Harvard Business School, 2008.

6. Tversky, Amos, and Daniel Kahneman. *Judgment under Uncertainty: Heuristics and Biases.* Science, New Series, Vol. 185, No. 4157. (Sep. 27, 1974), pp. 1124-1131.

7. Kahneman, Daniel. Thinking, Fast and Slow. MacMillan, 2011.