Rachel Kibler, August 2020
Twitter: @racheljoi
Website: http://www.racheljoi.com

Cognitive Biases in Software Testing, a Brief List

Availability heuristic – The people who are loudest often get seen as the most important or the most prevalent issues. Or the kinds of bugs that we see most become the kinds of bugs that we find most, to the detriment of finding other types of bugs.

Confirmation bias/Cognitive dissonance – We see what we want to see. If we have good relationships with our developers, and they normally write good code, we may not look for problems as much with their code. When we encounter something that doesn't fit into this, we may reject it easily.

Planning fallacy – This is one that plagues all of us. We underestimate the time it will take to complete something. Yes, testing is done when we decide it's done, which can be a time box or a level of confidence, but when we say something will take two hours, it could easily end up taking three or even a day.

Sunk cost fallacy – Sometimes software is so buggy that we might want to advocate for going back and trying it a different way. But we don't, because we've already put so much effort into this way. Automation tools and automation suites can easily fall victim to this fallacy too.

Primacy effect – The first thing we hear takes precedence over the rest of what we hear. (In a list of words, we tend to remember words towards the beginning than words towards the end.) The first thing we talk about with our testing may be the thing we latch onto, forgetting the rest, even if it's more important.

Anchoring – When we start with a baseline, wherever it is, deviations from that can seem amazing or upsetting or not really a change. In testing, I found what I thought was a bug, checked with a developer, and found out it was working as intended. That anchored me there, and slight deviations from it seemed less important, less like a cluster of bugs.

Framing effect – Depending on how information is presented, we may see it different ways. If a new story or epic is posed as exciting or a potential slog, that affects how we see it, even if the rest of the information is the same. We can do better or worse testing depending on how things are framed. We can practice reframing things in our conversations with others, to elicit a different effect based on how that person responds to communication.

Congruence bias – We find patterns where they may be different. When given a series of 2 4 6 and asked to find the pattern, it could be even numbers ascending by two, or any numbers ascending by two, or single-digit numbers, or many kinds of patterns, or none at all. We may see a pattern where none exists, or think we understand the pattern when we don't.

Conformity bias/Deindividuation – We end up going along with the team because it's easier. It's hard to say no to our team, especially when we say no a lot, so sometimes we just say yes, don't raise concerns, and go with the flow.

Dunning-Kruger Effect – Those with incredibly low skill in an area may not realize how much effort is involved. Consider the couch potato watching television, yelling at the quarterback for making mistakes. In software, this may be a manager who fails to understand how tricky a technology change will be, or a product owner who does not see the risks associated with that change, and thus the challenge of testing it.

Heusserability Bias – The inverse of Dunning-Kruger, where a highly skilled person fails to see how others could struggle where they succeed. In software, this is the expert that designs and runs a process that actually works for them, but fails to understand that other people would fail following the same process. This comes in two forms, the “perfect example” that the expert can perform but people lack the skill or expertise to follow, or the “idealized process”, that is designed on paper but does not work in process. (If the idealized process is more of a naive process, that may be Dunning-Kruger in action.)