WHITEPAPER

sqs.com

Why Do We Make Mistakes?

The Psychology of Testing

Author: Andrew R Brown
Senior Consultant, Automation

SQS Group Limited UK

Published: September 2016

Transforming the World Through Quality

Contents

ANDREW R BROWN
Senior Consultant, Automation
andrew.brown@sqs.com

Dr Andrew R Brown joined SQS in 2011 and has delivered a number of test automation projects for our clients in the UK and Europe. He is an accomplished trainer and led the upgrade of the selenium automation training course. Prior to joining SQS, he was test manager at HMV. He holds a degree in physics and maths, an MBA, and a PhD in semiconductors.

Introduction

This work began as a fortunate discovery that followed on from a software failure.

A customer complained that an automated test pack had failed to pick up a defect. They were uninterested in the explanation that automated scripts can only find what they are programmed to find. The customer pointed out, not unreasonably, that an automated test suite costing thousands of pounds had failed to detect what a manual tester costing a few pounds per hour should have immediately spotted. The customer had a point, as the defect was so obvious it was impossible to look at the screen and fail to see it.

Except that several people had done just that.

The developer missed it and passed it to the system tester. Not only did he miss it, but he used the latest technology to capture a screenshot containing the defect, attaching it as test evidence for the test manager to look at but not see. Similarly, it was looked at but not seen during user acceptance testing.

The realisation that several independent people had looked at but not seen the same defect started a train of thought. Why do we miss something that appears totally obvious when later pointed out to us? Why do we carry out actions that, with hindsight,

appear bizarre? Why do users faithfully recall events that, on closer investigation, could not possibly have occurred?

The answers to these questions are relevant to more than simply reducing defects. Our world is becoming increasingly complex as systems become more connected. The Internet of Things will place demands on our thinking and perception, as apparently unrelated systems chain together to produce effects that we cannot easily anticipate using our current ways of thinking.

Current situation

The software industry has come a long way in its first half century of existence. It has improved the lives of almost every person on the planet, helped us build machines that can look inside our own minds, and then helped us understand what we have seen. Software has even enabled humans to travel to the moon.

Yet, despite the many achievements of the software industry and despite the many innovations that have led to improved quality, every software program still contains multiple software defects. What is more, many defects we see today look remarkably similar to those present in the earliest software programs.

If an industry continues to make the same mistakes that it was making fifty years earlier, should this not give pause for reflection? Are we missing something? Is there a fundamental problem, present from the very start, that has not yet been addressed, or perhaps not even recognised?

Perhaps we need to look back to fundamentals. In the case of software testing, we can return to one of the very first questions:

What is a software defect?

This question can be answered in many ways. However, one answer does hint at an unresolved problem:

A software defect is **an error in a human thought process** *that becomes committed into code.*

Defects and thought processes

If a software defect is caused by an error in a thought process, then understanding thought processes ought to be central to software development. Yet how much effort has our industry put into this area?

To look for any unrecognised, fundamental problem, let us return to the birth of the software testing industry, an industry that may never have existed if some early decisions had been taken differently.

The software industry first attempted to resolve defects by making developers responsible for testing their own programs. Testing by developers was better than not testing at all, but unfortunately it suffered from two major shortcomings. Firstly, people find it difficult to identify faults in their own work, since if they could see a fault, then they would simply fix it. Secondly, this solution introduced a conflict of incentives, as developers are rewarded for delivering a working program, rather than discovering reasons why their program is not ready to deliver.

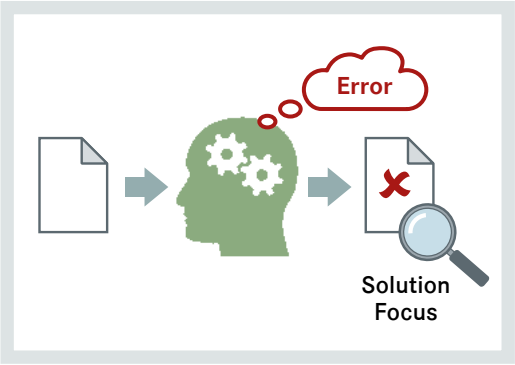
How did the software industry try to resolve this?

The industry introduced independent test teams. This helped address the shortcomings of the original solution. However, it also introduced new problems, such as communication issues and conflicts between different teams' goals.

Let us ask – did either solution remove the problem it was trying to solve?

We now notice something remarkable; *neither solution removed the cause of the problem*. In fact, neither solution even reduced the cause of the problem.

Getting developers to test their own code did not prevent them having erroneous thoughts. Similarly, adding an independent test team did not help developers to see faults in their own work.



Defects are due to thoughts. We fix our defects but not our thoughts.

This is a pattern common throughout software development. We encounter a shortcoming, but instead of focusing upon correcting the thought process that caused it, instead we add a correcting step. In doing so we add further complexity.

Why do we attempt to resolve problems in such a roundabout way? Why not resolve the error directly, when it is still inside our minds?

There are good reasons why we have not focused on resolving the error directly. It is extremely difficult to alter our way of thinking, such an approach has no guarantee of success, and in any case we do not have access to our thought processes [1]. However, these are reasons why it is challenging, rather than impossible.

The software industry: what might have been

It would be interesting to imagine how the software industry may have turned out if, instead of consigning armies of testers to run thousands of tests to uncover a few defects inadvertently buried by developers, we had approached this problem by trying to understand the thought patterns that lead to a defect being created in a person's mind, and then sought to modify those patterns.

The test industry may not have existed at all, as code contained too few defects to be worthwhile testing. Alternately, we may have found that equipping developers with an understanding of how some thought patterns give rise to defects meant that they were better at spotting flaws in their own work than independent testers, and also better than business analysts at translating requirements into specifications.

However, our present understanding of how our mind works relies upon recent advances in cognitive sciences, which were not available at the time the software test industry was created.

What advances in cognitive sciences have we seen? Can we use those advances to understand why we make mistakes and, more importantly, can we avoid making those mistakes?

Each of us believes that we behave rationally. We believe that when we perceive a situation, we select the best or most rational solution, maximising our own benefit. However, psychologists, notably Kahneman [2] and Tversky, identified that we frequently deviate from rational behaviour, doing so in predictable ways. Psychologists have named these deviations cognitive biases.

Cognitive biases

A cognitive bias is a systematic deviation from rational thought. Biases can be due to heuristics (short cuts), which allow faster processing when timeliness is more important than accuracy. They can also be due to human processing limitations or motivational factors [2].

What can we glean from this definition?

First, the good news. Biases are merely the by-product of some really useful short cuts that help us through life at a reasonable pace, despite our mental processing limitations. We are better off putting up with these biases than the alternative of not having the short cuts. Secondly, biases produce a systematic deviation. This means that they are predictable, both in the circumstances when we are vulnerable and also in the direction of the distortion.

Now the bad news. As the by-product of useful short cuts, cognitive biases will be difficult to remove whilst still retaining the benefits. Also, we humans are probably stuck with those mental processing limitations.

There are approximately 200 recognised cognitive biases. Most days we are influenced by several without ever being conscious of any of them. This is because biases act on our subconscious mind, which makes them difficult to rectify.

It also makes them insidious. Despite our best intentions to evaluate information and take decisions in a rational, unbiased manner, our cognitive biases act, distorting and discarding information before it is even accessible to our consciousness.

How can our cognitive biases affect software development? Here are some examples:

Inattentional blindness

Have you ever failed to spot something that appears blindingly obvious when later pointed out to you? Inattentional blindness occurs when we fail to recognise something that is unexpected but is in plain sight. For example, it happens to testers focused on an attentionally demanding task, such as checking for the brief appearance of a transient message, but failing to notice that the message content is wrong.

Confirmation bias

A support engineer at a software house was asked to refresh the test site data for a client, a building society in Yorkshire. He checked that he was on the correct client, then began the data refresh, first by dropping the database tables. A few minutes later, the helpdesk received a frantic phone call from the client, asking what was happening to their live site.

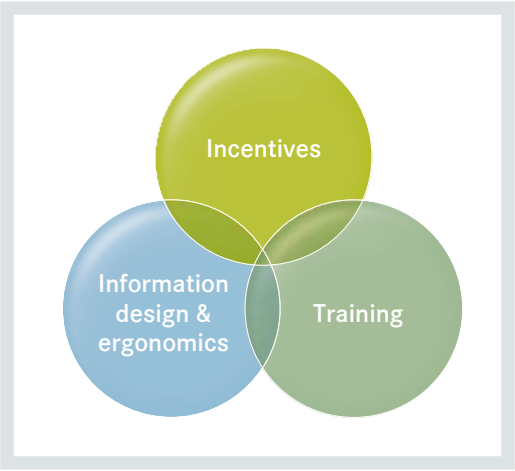
The support engineer believed that he was on the client's test site, and used information he saw on the screen to confirm his beliefs. Unfortunately, everything he took as confirmation that he was on the client test site was also true of the client live site, with catastrophic results.

Confirmation bias, is where we pursue, interpret, and recall information in a way that confirms our pre-existing beliefs, whilst ignoring or discounting contradictory information. Other cognitive biases relevant to software development include:

Bias	Description
Anchoring effect	Relying too heavily on one 'anchor' piece of information (usually the first piece). An example is when a test phase starts well, but drifts unnoticed into failure. People's thoughts are anchored by the initial, positive, data and fail to give sufficient weight to new, contradictory data.
Bias blind spot	Seeing that others suffer a given bias, but failing to see the same bias in our own behaviour.
Curse of knowledge	When a better informed person cannot understand a problem from the perspective of a lesser-informed person. For example, when developers cannot appreciate why users don't understand why the system cannot work in the way they want it to.
Endowment effect	Valuing items we possess more than items we do not. The result of this effect can be seen in regression packs full of similar, low value tests, whilst critical parts of the system remain uncovered.
False memory (confabulation)	Producing a fabricated, distorted or misinterpreted memory, but without any conscious intent to deceive. Confabulation may account for those helpdesk and bug reports that cannot be reproduced. Alternately, your spouse's version of the events leading up to your new Audi becoming a total write-off may contain some elements of confabulation.
Hindsight bias	Seeing past events as being predictable at the time they occurred. The 'I-knew-it-all-along' effect. Remember the poor engineer who inadvertently trashed the live site of the building society in Yorkshire? The subsequent investigation was full of conclusions reached with the benefit of hindsight bias.
Information bias	Seeking out additional information, even when it cannot affect the outcome. The first five payment tests failed. Do you really need to run more tests to know that the payments system is broken?
Outcome bias	Judging a decision by its outcome, rather than the quality of the decision, based on the information available at the time. This effect is frequently seen in conjunction with the Hindsight Bias during failure investigation meetings and project retrospectives.
Planning fallacy	The tendency to underestimate task completion times. Have you ever sat in a planning meeting where everyone agreed that a project should take 6 months, despite everyone knowing that similar projects had taken 9-18 months?

How can we reduce the effect of cognitive biases?

Currently, there are three recognised approaches, as depicted in the figure below.



Incentives

This is the approach that has been used most frequently in the past. It involves structuring rewards and penalties so as to override a cognitive bias.

De-biasing through incentives is effective in a wide variety of situations. However, this approach has disadvantages.

Firstly, testers want to do a good job, so they are already incentivised. Hence, the effect of incentives may be limited to ensuring the organisation does not create incentives for testers not to find defects. This may sound obvious, perhaps even trite. However, it is surprising how frequently organisations inadvertently do this.

Another criticism of the incentives approach is that it merely counteracts one distortion with another. Since we cannot be certain that the incentive distortion will exactly balance out the original bias, we face the possibility of under or over-compensation.

Finally, as seen below, an incentive may have the side effect of encouraging undesirable behaviour.

An incentive approach is best used when we are primarily interested in achieving a particular outcome for a specific event and are uninterested in developing a long-term capability. Whilst this approach may have its place in certain situations, it is unlikely to represent the entire solution.

Information design & test ergonomics

This approach focuses on modifying the way information is presented so that it does not induce bias.

An example of how information design can reduce a specific cognitive bias was demonstrated to a symposium of doctors at a medical conference [3].

A patient participates in a routine mammography screening. She tests positive. Alarmed, she wants to know what are the chances that she has breast cancer. Providing a patient with an accurate understanding of what this result means is obviously an important part of patient care.

When the relevant information was given to the physicians attending the conference, but expressed in percentage probabilities, only 21 % managed to select the correct answer. This percentage was actually lower than the percentage expected purely by chance. However, after the information was redesigned to be expressed as natural frequencies and the audience were taken through a 75-minute training session, 87% selected the correct answer.

Admittedly, medical people represent an intelligent segment of the population and they were guided through a training session. Nevertheless, an improvement from 21 % to 87% is a convincing demonstration of the potential for de-biasing through information design.

Training

De-biasing through training holds the promise of an approach that is unclouded by distorting a bias with an incentive. Also, it does not rely upon a careful modification of the information structure, which in any case may not be possible. A further benefit of a training approach is that it may deliver a transferable skill that is useful beyond the immediate situation.

Unfortunately, early attempts at cognitive de-biasing using training met with disappointing results [4]. Consequently, attempts to de-bias by training were not pursued with the same vigour as other approaches. However, this approach has recently received renewed interest from the intelligence analysis community.

This community faces complex situations, with limited access to much critical source material. The information is often fragmentary, of dubious authenticity, unknown relevance, and is difficult to verify. An adversary's goal is often unclear, and differences in social values may mean that an action could be meaningless to you but make perfect sense to your adversary. Finally, it is perfectly legitimate for an adversary to deceive you through the deliberate planting of false information, as occurred during the Cuban missile crisis.

It is hardly surprising that this community has experienced major intelligence failures, such as Pearl Harbour, the Cuban missile crisis, and Iraq WMD [5,6].

Faced with these difficulties, a plethora of different cognitive biases distort the intelligence analysis processes. It is therefore understandable that the intelligence analysis community has begun to explore cognitive de-biasing through training.

A recent study into the immediate and lasting effects of de-biasing training was carried out [7]. Half of the participants were exposed to an instructional video, and the remainder took part in a computer-based video game. The authors reported significant and lasting de-biasing effects across six cognitive biases studied.

Following the reported results of de-biasing using training and information design interventions, this study has chosen to investigate cognitive biases using these two approaches.

How long should a training intervention be? We found that a typical intervention to reduce vulnerability to a specific bias, say the bias blind spot, could be achieved in approximately three hours. As there are approximately 200 cognitive biases, this might suggest a significant training requirement. However, the total time required is much less for two reasons. Firstly, only a few of these 200 biases directly affect software development. Secondly, concepts and learnings are common across many de-biasing interventions. The result is that a minimal, yet effective, de-biasing package can be delivered in 6-8 hours, with a more comprehensive package requiring 15-20 hours.

Experiments and interventions

Changing ourselves is hard. Implementing a training intervention to address cognitive biases turned out to be considerably more difficult, and the results were far less predictable, than implementing previous software training interventions, such as teaching Selenium.

One difficulty arose because de-biasing tries to improve a defective system (our biased mind), but using that same system as the de-biasing tool. This could be likened to using a defective ruler to measure its own length.

Another difficulty is that when communicating personal shortcomings to participants, they tend to reject unwelcome messages. We therefore found it necessary to strike a balance between confronting participants with an explicit message and risking outright rejection of that message, or delivering a more palatable message, but not one that was recognised as requiring any action.

Once these difficulties were understood, the interventions were redesigned and we observed a marked improvement. A further clue was given by Kahneman [2], who reported on a successful and unsuccessful de-biasing intervention with a group of psychology students, saying:

“To teach students any psychology they did not already know, you must surprise them.”

Hence, interventions were designed to engage participants in a way that surprised them and encouraged them to re-evaluate themselves.

The following format was found to be successful:

- Initial survey and feedback regarding an individual’s current vulnerability to the bias
- Detailed explanation of the bias, suspected causes, impact and possible countermeasures
- Repeated cycle of practice, feedback and improvement
- Measurement of the bias reduction actually achieved

Bias blind spot

Why is reducing our bias blind spot important? Because before we can accept that any cognitive bias affects us, we must first accept that we are every bit as vulnerable to cognitive bias as the next person [8]. Tackling the bias blind spot is also important because it is at the root of many misunderstandings between individuals and groups, resulting in error and friction on IT projects.

Training started with a survey [9]. This was used to gauge participants' initial level of bias blind spot and also deliver later feedback. Participants were then taken through a de-biasing training session, based on experiments performed by Nisbett, Morewedge, Pronin, Scopelliti [1,7,8,9].

De-biasing training was combined with a programme of demonstrations and interactive lab sessions aimed at illustrating the extent to which we are all susceptible to visual and cognitive illusions, as well as highlighting just how tenuous our grip on reality actually is.

The training intervention was followed by a survey that was used to gauge participants' post-intervention vulnerability to this bias. Vulnerability to the bias blind spot was reduced by approximately 20% – 50%.

Once participants had accepted their vulnerability to bias blind spot, they were subsequently receptive to other de-biasing training interventions.

Inattentional blindness

Inattentional blindness occurs when we fail to recognise something that is unexpected, although it is in plain sight. Failure to spot something in plain sight is of obvious relevance to testing.

Examples can be found throughout history, indicating both how commonplace and also how important it is. For penicillin, Alexander Fleming is unlikely to have been the first person to have experienced petri dish cultures ruined by fungi. Dozens of trained researchers, whose very profession centred on making discoveries, would have looked at dishes of ruined cultures, but never once saw what was in front of them. Fleming, however, was the first to notice and understand the effect.

So why do we miss something that appears so obvious after it is called to our attention? Research shows that what we see depends very much upon what we are expecting to see [10,11,12]. This is obviously bad news if we are hoping to find a bug that reveals itself through an unexpected event. (It is also bad news if we are riding a bicycle and we encounter a car driver who is expecting to encounter and avoid only car-like objects.)

We tend to miss an event when we are (a) engaged in an attentionally demanding task, (b) not expecting a new stimulus, and (c) actively searching for a specific item to achieve a goal. We will even miss an event if the task we were engaged with is completed and we no longer have any task to attend to [10]. Unfortunately, these conditions are precisely those under which testers normally operate.

So how can we reduce our vulnerability to inattentional blindness?

Previous research results [10,11,12] suggested that greater impact would result from modifying the test environment ergonomics, rather than undergoing a pure training intervention. This was achieved by ensuring that the observer was not engaged (either currently or recently) in an attentionally demanding task at a time when an unexpected event was possible.

The following modifications to environment ergonomics were found to reduce inattentional blindness:

- Dual or multi-screen configurations (to enable side-by-side comparison of actual and expected results)
- Reduced mental load, through use of checklists and pre-calculation of expected results
- Pair testing, with each tester having a specific allocated role
- Multiple test runs, to allow a 'general observation' run
- Automated run-through, with the tester purely observing

In addition to modifying the environment, participants were taken though a training intervention to reduce inattentional blindness.

Confirmation bias

When testers suffer from confirmation bias, they restrict their investigations to confirming the expected behaviour of the system. This results in defects being missed.

It may be claimed that this bias has been addressed through negative testing. However, negative testing is an entirely conscious process, and does not address the problem of a thought being rejected by our subconscious before we are aware of it. Defects that pass through multiple test stages and into live systems are a testament to our vulnerability to this bias.

The training interventions were modelled on the work of Wason [13], plus our own material. This was followed by a practical session that was used to gauge participants' post-intervention vulnerability to this bias.

Hindsight bias

Sometimes known as the 'I-knew-it-all-along' effect. Hindsight bias is when we see past events as being predictable at the time those events occurred, even though there was little objective evidence to support any such prediction. Hindsight bias seriously restricts our ability to judge or learn from past events.

The hindsight intervention was modelled on Fischhoff's classic experiment [14], plus our own material.

False memory (confabulation)

False memory, or confabulation, is a memory disturbance where we create a fabricated, distorted or misinterpreted memory, but without any conscious intention to deceive. An inaccurate recollection of events is an obvious disadvantage to accurate test recording.

This bias was one of the first we recognised to affect testing, and was stumbled upon as follows. During a debugging training session, a student reported observing an effect that the instructor knew could not possibly have been observed. The instructor attempted to correct the error by getting other class members to verify that they had not observed the effect. However, to his astonishment, several class members claimed to have made this impossible observation. This event was repeated in subsequent training sessions, and the effect was found to be surprisingly reproducible.

The de-biasing intervention began by taking participants through a series of false memory list scenarios, similar to those created by Deese-Roediger-McDermott [15]. This was followed by a reconstruction of automobile destruction scenarios, similar to those created by Loftus [16]. Intervention was completed with cycles of scenarios similar to the debugging session described earlier.

Field trial

Although the results of these interventions were encouraging, we recognise that they relate to reducing cognitive biases, whereas our ultimate goal is to improve performance on testing projects. Therefore, a field trial is currently in process, focused on investigating the effect of cognitive de-biasing on leakage of defects to live environments. Initial results are encouraging and support the assertion that reducing cognitive biases leads to defect reduction.

Conclusion and outlook

Software defects have been linked to specific cognitive biases. Some of these biases have been reduced using a combination of training and modifications to test environment ergonomics. An on-going field trial to reduce defects through de-biasing interventions shows positive initial results.

Further conclusions are possible. However, study of the connection between software defects and cognitive biases is still in its infancy, so rather than finish with some fixed conclusions, perhaps it is better to finish with some open questions:

- Can the software industry use advances already available from cognitive sciences to approach the problem of software defects from an entirely different starting point?
- How might the software industry look if it adopts such an approach?
- Under what circumstances and during which software stages are we most vulnerable?
- In addition to software development, what other areas of IT could benefit?
- A software defect begins life in our minds. When will that become the first place we look to correct it?

And finally:

- Can addressing the fundamentals of how we think help a world that is becoming ever more connected through the Internet of Things?

References

[1] Nisbett R. E., Wilson T. D. Telling more than we know: verbal reports on mental processes. Psychological Review, 8, 231 – 259 1977

[2] Kahneman D. Thinking fast and slow

[3] Gigerenzer G. Risk savvy.

[4] Fischhoff B. (1982). De-biasing. In D. Kahneman, P. Slovic & A. Tversky (Eds) Judgement under uncertainty: Heuristics and biases. Cambridge University press.

[5] Heuer R.J. Psychology of Intelligence Analysis. Center for the Study of intelligence. CIA. 1999

[6] Moore D.T. Critical Thinking and Intelligence Analysis. National Defence Intelligence College, 2007

[7] Morewedge C.K., Yoon H., Scopelliti I., Symboreski C.W., Korris J.H., and Kassam K.S. Debiasing Decisions: Improved Decision Making with a Single Training Intervention. Policy Insights from the Behavioural and Brain Sciences Journal 2015, Vol 2(1) 129 - 140

[8] Pronin E., Lin D. Y., Ross R. The Bias Blind Spot: Perceptions of Bias in Self versus Others. PSBP Vol 28, No 3. March 2002, 369 - 381

[9] Scopelliti I., Morewedge C.K., McCormick E, Min H. L, Lebrecht S, Kassam K.S. Bias Blind Spot: Structure, Measurement, and Consequences. Management Sciences Vol 61, No 10, 2015, pages 2468 - 2486

[10] Mack A, and Rock I. Inattentional Blindness. An Overview. Cambridge MA Press

[11] Most, S. B., Scholl B. J., Simons D. J., Clifford E. R. (2005). What You See Is What You Get: Sustained Inattentional Blindness and the Capture of Awareness Psychological Review 2005 Vol. 112 (1): 217–242

[12] Simons D. J, Chabris C. F., Gorillas in our midst: sustained inattentional blindness for dynamic events Perception, 1999 [Also, see their book 'The invisible Gorilla']

[13] Wason P. C. On the failure to eliminate hypotheses in a conceptual task. The Quarterly Journal of Experimental Psychology 1960,12:3, pages 129 – 140

[14] Fischhoff B. Hindsight <> foresight: the effect of outcome knowledge on judgement under uncertainty. Journal of Experimental Psychology: Human Perception and Performance 1975, Volume 1, pages 288 – 299.

[15] (Deese J.), Roediger III H. L., McDermott K. B. Creating false memories: remembering words not presented in lists. Journal of experimental psychology: learning, memory, and cognition. 1995, volume 21, no 4

[16] Loftus E. F., Palmer J. C. Reconstruction of automobile destruction: an example of the interaction between language and memory. Journal of verbal learning and behaviour, volume 13, issue 5, 1974

SQS Software Quality Systems AG
Phone: +49 2203 9154-0
Fax: +49 2203 9154-55
info@sqs.com | www.sqs.com