Ten Tendencies That Trap Testers

An informal 10-year study of limiting behaviors seen during job interviews

Jon Bach
Manager for Corporate Intellect
Quardev, Inc.

August 13, 2007

Bio:

Jon Bach is lead consultant and corporate intellect manager for Quardev – an outsource test lab in Seattle, Washington.

He is co-inventor (with his brother James) of "Session-Based Test Management" – a technique for managing (and measuring) exploratory testing.

For 12 years, Jon has worked as a test contractor, full-time test manager, and consultant for companies such as Microsoft, Rational, Washington Mutual, and Hewlett-Packard. He has written articles for *Computer* magazine and *Better Software* (formerly *STQE Magazine*). He is also a frequent speaker and presenter at testing conferences like STAR, CAST, TCS, TISQA, QAI, and PNSQC.

At Quardev, Jon manages testing projects ranging from a few days to several months using Rapid Testing techniques (like SBTM). He is the speaker chairman for Washington Software Alliance's Quality Assurance SIG, as well as Conference President for the Association of Software Testing.

Abstract:

For the last 10 years, I have conducted over 400 job interviews for people seeking test positions. Called "auditions," these job interviews are meant to be interactive project simulations, designed to allow the tester to demonstrate their testing skill. This paper discusses the pathologies I have seen that tend to limit a candidate's ability to be effective and suggests some remedies for those tendencies. This is not a scientific study, but part of my conclusions are founded on test data (log files of mouse clicks and keystrokes) from 65 interviewees as they each used the same piece of software to demonstrate their skills.

Overview

As a software testing provider, software comes to us with its weaknesses usually hidden. The same can be said for humans. We have weaknesses, too, and most of us are very good at keeping them hidden (especially on job interviews), until, like software, our programming is tested or triggered into revealing them.

This paper is about my experience testing software testers during job interviews. It's about a process I use to get them to reveal their strengths and weaknesses to me so I can determine if they would be a good fit for the jobs we have at Quardev – an outsource, onshore test lab in Seattle. It is an informal accounting of my experiences as interviewer, or host, of over 400 individual "auditions" – roughly one-hour sessions where job candidates demonstrate their testing skill to me. Although I consider myself a scientist, this is not a scientific study because the data I collected is contextual. Other than my experience as an interviewer, this paper includes, in its foundation, data from log files kept as the candidates tested the software.

The interview process starts with having received a resume, which leads to a phone screen where the tester is asked about their work experience. Then they are given a product to test (over email) for 20 minutes, where they must write up one bug. After that, they are invited into the lab where they go through an in-person Concise In-Depth Survey (CIDS) of their work experience. The audition (project simulation) is the last stage. It consists of me standing up at the whiteboard as I introduce a sample testing project.

I write the word "Bugs" on a whiteboard and underline it. This is where they will write their problem reports. I tell them I just need a title (not the full report) and that it must be 20 words or fewer.

Under that, I write "Issues / Questions". This section is for any questions they ask me as well as any issues they'd like to raise.

An "Issue" is a concern that could affect the success of the project. It could read like a bug title: "No setup.exe in the main directory." Why not file this under the "Bugs" section? The tester may not be confident that it's a bug because they may be unsure if it is by design, so filing it as an issue preserves their credibility. The fact that there is no setup.exe could be a bug, it just depends on the candidate's level of confidence in their model of how the program is designed.

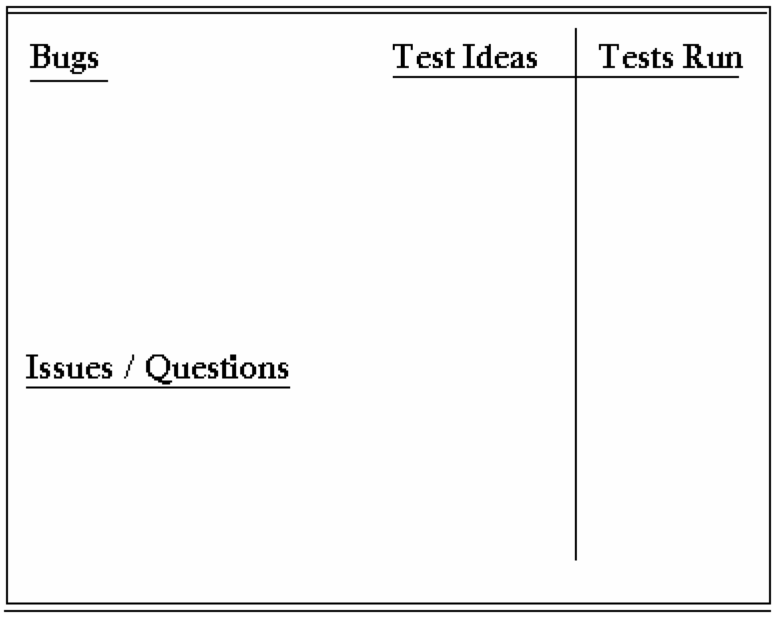
If a tester is too cautious, however, and never file bugs in the "Bugs" section during the audition, I'd be concerned. I look for a balance between caution and confidence. Being too cautious is a pathology, but so is ardent righteousness where testers declare any concern in their mind to be a bug, no matter the context.

At this point in the audition, I summarize. I remind the tester that they have three choices when they encounter what may be a problem in the software: 1) file a bug, 2) raise an issue, or 3) ask a question.

I then draw two columns next to the “Bugs” and “Issues / Questions” section: “Test Ideas” and “Tests Run”.

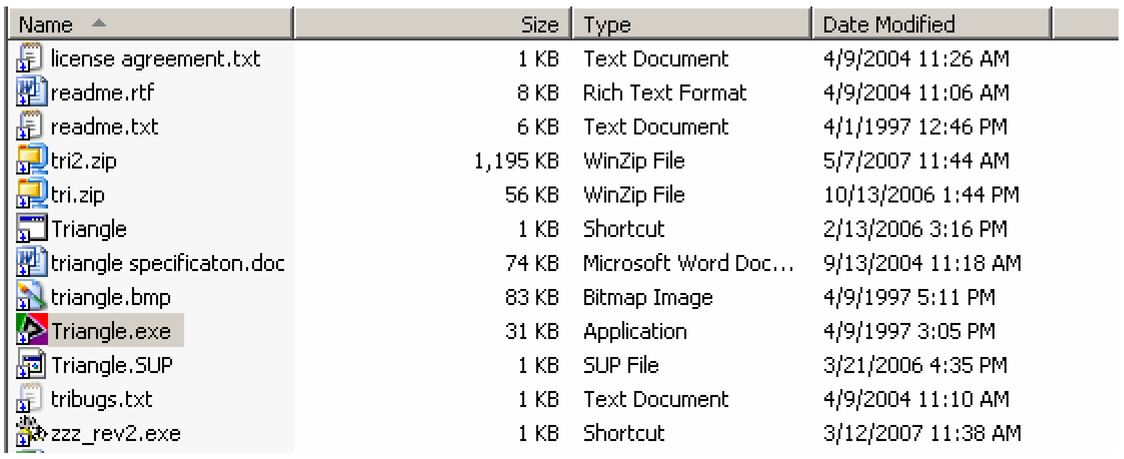
I explain that “Test Ideas” are verbal “commitments” to run tests or use techniques that take too long to actually run in the interview (like localization or scalability) whereas the “Tests Run” section is a running shorthand list of tests I see the tester perform.

At this point, the whiteboard looks like this:



As I stand at the whiteboard, prepared to be their scribe for the next hour, the candidate sits in front of a laptop running Windows XP ^{TM} . The laptop is connected to a projector so I can see what’s on their screen without having to stand over them as they test. In front of them on the laptop is an open directory containing several files necessary for the program to run. I explain that these are the contents of the “shipping CD” or “release candidate” that the client wants our lab to test.

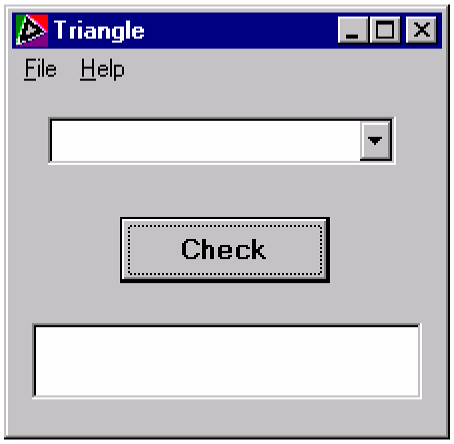
In that directory, they find the following files:



Name	Size	Type	Date Modified
license agreement.txt	1 KB	Text Document	4/9/2004 11:26 AM
readme.rtf	8 KB	Rich Text Format	4/9/2004 11:06 AM
readme.txt	6 KB	Text Document	4/1/1997 12:46 PM
tri2.zip	1,195 KB	WinZip File	5/7/2007 11:44 AM
tri.zip	56 KB	WinZip File	10/13/2006 1:44 PM
Triangle	1 KB	Shortcut	2/13/2006 3:16 PM
triangle specificaton.doc	74 KB	Microsoft Word Doc...	9/13/2004 11:18 AM
triangle.bmp	83 KB	Bitmap Image	4/9/1997 5:11 PM
Triangle.exe	31 KB	Application	4/9/1997 3:05 PM
Triangle.SUP	1 KB	SUP File	3/21/2006 4:35 PM
tribugs.txt	1 KB	Text Document	4/9/2004 11:10 AM
zzz_rev2.exe	1 KB	Shortcut	3/12/2007 11:38 AM

I explain the mission – find interesting bugs (crashes, hangs, data loss).

The program I want them to test is called the Triangle Program. It was built in Visual Basic and looks like this:



I depict this on the whiteboard and explain that the UI takes three numbers separated by commas, such that when the CHECK button is pressed, it tells you what kind of triangle it would make. There are five possible outputs – "scalene" (three unequal sides), "equilateral" (three equal sides), "isosceles" (two equal sides), "invalid", and "Not a Triangle."

I declare the audition underway and ask if the tester has any questions to get the ball rolling.

Right away, I start looking for testing skill. Things like:

- ***Questioning***: Do they ask questions to frame their testing or do they start testing right away?
- ***Resourcing***: Do they ask for mission-clarifying or oracle-building project artifacts like specs, test cases, emails, requirements?
- ***Modeling***: How will they take notice of the files in the directory? Will they open every file? Ask questions about what each one is? Design tests around them?
- ***Recording:*** Will they take notes as they test?
- ***Conjecturing***: If they don't ask me any questions, what assumptions do they have about the product and how do they act on them?

I'm looking for what I call "The 3 C's" -- caution, critical thinking, and curiosity.
(Note: a reviewer of this paper suggested "creativity" as a 4th "C", and although I look for that as well, it is usually later in the audition.)

If they decide to ask me a question, I will break into one of several characters, answering as programmer, project manager, test lead, CEO, etc. Sometimes I answer as well-meaning-but-misinformed client. Sometimes I contradict myself, as happens on real projects.

My aim is to see their skill, but to trap them in ways I have seen myself and others get trapped. Being “trapped” means a situation that constrains or limits the candidate such that a testing risk is created. Some traps are not that dire, other traps can torpedo a project. There are multiple ways out of every trap I set.

In seeing testers go through the audition, I have seen some behaviors that I will call *tendencies*. Here are the top ten tendencies I’ve seen that have trapped testers, starting with the least common:

Trap #10: Stakeholder Trust

This trap happens when testers believe that stakeholders are the keepers of all of the necessary information, and that all the information given is current and relevant.

As I play the roles of stakeholders like Programmer, Test Manager, Program Manager, CEO, and Customer, I see if the candidate questions the information they are given. In my experience, people who usually have no idea about testing theory or practice have mistaken beliefs about the role and value of testers, so I look to see if candidates push back on fallacious notions (like being told that they are solely responsible for quality).

Trap #9: Compartmental Thinking

This trap occurs when testers think only about what’s proximate or in their field-of-view. When testers do not take into account other, opposite, or orthogonal dimensions, it causes them to miss systemic bugs or to leave whole features untested.

Trap #8: Definition Faith

When testers don’t consider that terms like “regression testing”, “test case”, “function” or “feature” mean different things to different people. As a result, the tester could accidentally use these terms to convey a notion of testing completeness when testing actually hasn’t yet started.

Trap #7: Inattentional Blindness

A bit different than the Compartmental Thinking trap, Inattentional Blindness is meant to describe when the tester sees something in their field of view, but does not process the information, so in effect, does not really “see” the behavior. Derived from the concept of compartmentalization in cognitive psychology, it is “The inability to perceive features in a visual scene when the observer is not attending to them.” (*Wikipedia*)

A video produced by the University of Illinois’ Visual Cognition lab demonstrates this phenomenon. It can be viewed at http://viscog.beckman.uiuc.edu/grafs/demos/15.html

Trap #6: Dismissed Confusion

Too often, the testers see and identify suspicious behavior but rule it out because they're not confident in their reasoning. They think confusion is weakness and often believe the problem isn't the software because smarter or more veteran people than they created it.

Trap #5: Performance Paralysis

This is where a tester temporarily fears ideas. Either they have no idea of what to do or so *many* ideas that they do not choose one out of fear that another will be forsaken. The fear of choosing one idea over or failing to produce *any* idea for fear that it may be the "wrong" one, causes testers to freeze, making no choice at all.

Trap #4: Function Fanaticism

Running tests that only reveal what the program does or doesn't do through its UI, instead of what it is comprised of, what it processes, how it's used, or what it depends upon. The "fanaticism" comes when the tester goes right for opening the UI and stays locked in function tests for a majority of the audition, forsaking all other test ideas, techniques, planning, or approaches.

Trap #3: Yourself, untested

This trap is when the tester does not take the time to evaluate their work through the eyes of key stakeholders. Examples include test ideas they cannot explain to a CEO, testing knowledge they assume everyone has, imprecise notes that tell an incomplete story to a test manager, notes that are so precise that the story of testing is ambiguous, bug titles that are misleading to developers, and bug reports that have missing information that programmers notoriously ask for. In short, it's when testers do not understand (or *choose* not to understand) stakeholder expectations for information that might be valuable.

Trap #2: Bad Oracles

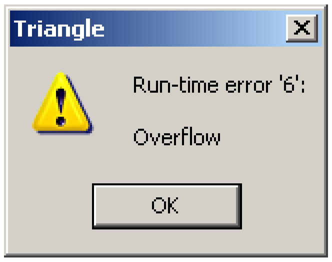
Oracles are principles or mechanisms with which to identify problems. Without oracles, there can be no testing because we won't know whether a test passed or failed. A "bug" is a difference between desired behavior and actual behavior, and oracles indicate the desired behavior. The "bad oracles" trap is when testers do not know what oracle they are using (i.e. cannot sufficiently describe it to others), or base their conclusions on faulty reasoning.

Trap #1: Premature Celebration

This means to rush to an incomplete judgment of a bug despite the prospect of easily gathering better information. ***Failures*** are symptoms of ***faults***, and testers are often content with reporting failures without looking for the underlying cause. Filing failures is fine in the audition especially because time is short, and besides, I don't need testers to investigate bugs in the audition to the degree where they can point out the problem in the code. But the trap I see the most (putting credibility on the line every time) is when testers stop their investigation too soon, writing the bug despite the fault being just a few more clicks away had the tester continued on their testing path.

This is a trap because usually the bug is a failure, not a fault, but worse, the tester does no follow-up testing or investigation to make sure they are providing a more complete picture or context of the problem.

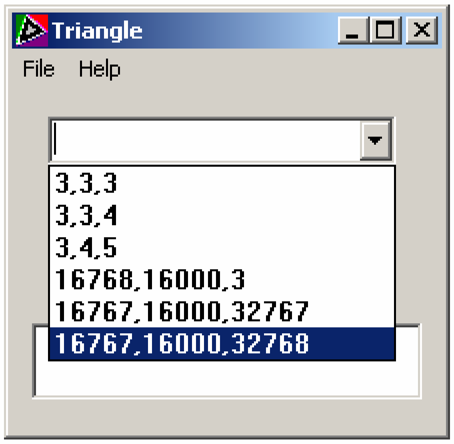
This is one bug commonly found by testers in the audition:



Run-time error '6':

Overflow

It is easily found because it lies in a pre-existing value in a dropdown list, one of only a few functions available to test in the UI:



3,3,3
3,3,4
3,4,5
16768,16000,3
16767,16000,32767
16767,16000,32768

A tester will often select this value and click the "Check" button, which causes the error. Right away, they will file this as a bug with a title like: "Run-time error '6' Overflow."

I look to see if they run follow-up tests around this. For example, if they don't know what "run-time" means, ok, but do they have ideas on whether this be a serious problem with interoperability or is this an isolated event? They see that 3,3,3 produces an equilateral triangle, and they see that 16767,16000,32768 produces a run-time error, so will they do something like a binary search to narrow down the point at which the problem starts?

Traps Summary

#	Tendency	Description
10	Stakeholder Trust	The belief that the stakeholders are the keepers of all of the necessary information for the tester to do a thorough job and that all the information they have is current and relevant.
9	Compartmental Thinking	Thinking only about what's proximate or in the tester's field-of-view.
8	Definition Faith	Not considering that subjective terms like "regression testing", "test case", "function" or "feature" mean different things to different people.
7	Inattentional Blindness	When the tester sees something in their field of view, but does not process the information correctly so in effect, does not really "see" the behavior.
6	Dismissed Confusion	The tester sees and identifies a suspicious behavior but rules it out as a bug because they're not confident in their reasoning.
5	Performance Paralysis	When a tester temporarily fears ideas.
4	Function Fanaticism	Running tests that only reveal what the program does through its UI.
3	Yourself, untested	When the tester does not take the time to evaluate their work through the eyes of key stakeholders.
2	Bad Oracles	When the tester does not know [or cannot sufficiently describe] the principle or method they used to identify a problem.
1	Premature Celebration	To rush to an incomplete judgment of a bug at the cost of better information that's readily available.

Each trap is not only avoidable, each has a way out. I look for trap-avoidance skill in the audition because it is an important martial art to have on a project. You may have seen movies where a kung fu master has felled 50 opponents, and instead of brushing off and simply walking away, the master is alert for the 51st, backing slowly out of the battle arena, scanning for that next threat from any direction.

My thoughts about remedies for these traps (some of which I have seen testers do in the audition even after getting trapped), are as follows. It is my hope that reading this list provokes reflection and inspires you to become not only better job candidates yourself, but better interviewers as well:

10) Stakeholder Trust

Remedy:

Create a line of inquiry -- a structure that organizes reading, questioning, conversation, testing, or any other information-gathering tactic. It is investigation oriented around a *specific* goal. Many lines of inquiry may be served during exploration.

9) Compartmental Thinking

Remedy:

Try the Brute Cause Analysis game. It's called "brute" cause because the game encourages testers to force reasons for causality. It's a game played with two people (but can be played alone). It starts with one person describing any failure (like "Invalid or Missing DLL: RICHTX32"). The other person describes a feature, like "Help / About" which has a link to the latest upgrade. The first person then has to tell a story about a possible link between the two even when none might exist. This exercise is meant to break testers out of unchallenged assumptions. Assumptions are not dangerous, but if unchallenged could cause us to look foolish. Compartmental thinking is a trap that can be avoided by looking at system interactions between things (like product features and error messages describing missing DLLs).

8) Definition Faith

Remedy:

What do you think when you hear the word "test"? Could mean someone wants you to evaluate or find problems? Compare and contrast with something else? Suggest improvements? Verify, corroborate, experiment? Since each of these activities may be orthogonal and have different costs, imagine some of these different definitions and check in with stakeholders to know which focus you need in order to be of value to them. Yes, it may be that they say "yes" to all of them, but if that's the case, weight the risk of trying each definition because each may have a different cost.

Then ask "test for what?" Scalability, usability, reliability, performance, robustness, correctness, compliance, regression? Each of these attributes might need you to use a different technique, which can be more or less time-consuming or expensive than others. The spirit of the "Definition Faith" remedy is to place the focus on being clear with terms so that you can manage the risk of prioritizing activities that best suit the definition of the stakeholders to which you are in service as a tester.

7) Inattentional Blindness

Remedy:

It's a Catch-22, how do you know you're not seeing something when you're not seeing it? Should we go through life paranoid, questioning everything three or four times just to be on the safe side? I've never faulted testers in the interview when they did not see something that to me was obvious, but I did see testers that either re-executed tests to get more information or consulted a separate, anchoring source (like the spec) to populate their notions of where faults were, even if it was right in front of them. These actions allowed them to see heretofore hidden things because their knowledge of the underlying environment was improved with different elements of context needed to see the whole picture.

Here's an idea: act like a pilot. Pilots are trained to consult a variety of sources so they don't rely what might be a faulty instrument. Known as "situational awareness", they scan a variety of instruments meant to tell them the same data in different ways.*

* (Fighter pilots often get so fixated on their target that they lose track of what the airplane is doing. This necessitated electronic voices in instruments -- a different sense than sight -- telling them "Pull up, pull up…" even when they could see the ground fast approaching.)

6) Dismissed Confusion

Remedy:

Your confusion is not a liability, it is a weapon for you to use. Learn to recognize your confusion as a trigger, a powerful tool that tells you that something confusing may be going on and that to remedy your uneasy gut feeling, you must ask a question or raise an issue. It is this that often provokes excellent conversations and keeps confusion from being a virus that is passed along to the customer. Knowing that you can provide this service on a project could be an easy way for you to feel empowered instead of feeling shame that you should have known better.

5) Performance Paralysis

Remedy:

Life is a journey and so is testing. It is a pursuit that has to start somewhere. When I see testers paralyzed, I try to provoke them into showing me any idea, as lame or silly as I think it might be, to see how they react to it.

One remedy is to try a P.I.Q. cycle: Plunge-In-and-Quit. Start anywhere, with any idea, follow it, but when it gets too complex or abstract, causing your brain to hurt, quit! Take a break, come back to it later with new eyes.

Testers in the auditions I've hosted have never had a lack of ideas, only a fear of picking one and describing it. When I see this happen, I will leave the room, saying I need some water asking if they want anything. When I come back, they are usually ready to show me an idea.

4) Function Fanaticism

Remedy:

In the audition, many testers run function tests (likely because time is short, so they go right for the UI to find a crash, hang, or loss of data), but it's a tendency to get themselves trapped.

When trapped like this, use a heuristic or a checklist. One I use is a mnemonic device I call SFDPO, (said as "San Francisco Depot"), which reminds me to try tests in the domains of Structure, Function, Data, Platform, Operations. Although function tests are the F in that mnemonic, it also includes tests that are designed to reveal errors in whatever the product is comprised of (S), whatever data it can process (D), whatever other software or hardware it depends upon (P), and how it is meant to be used (O).

In the "Test Ideas" column on the whiteboard, I look to see if the tester is familiar and can describe the quality factors (also known as the "ilities") like scalability, reliability, portability, maintainability, capability, usability, installability, etc.

3) Yourself, untested

Remedy:

Do you evaluate your testing techniques, strategy, approach, plan, risk assessment – any artifact of your testing before you deliver it? In the interview, I want candidates to show me they have some confidence that they're right for the job instead of assuming they did a good job with whatever they did during the interview. I look for critical thinking in their testing throughout the interview, but I also look for them to apply that critical thinking to their own work – e.g. do they ask me how they did, or what they could have done better, or do they change their mind on a bug report after further review. Some call this "humility", and I agree. It is often a virtue for a tester, but I'm saying that too much (or too little of it) of it can jeopardize your credibility.

2) Bad Oracles

The most common bad oracle I have seen when testers test the Triangle Program is the belief that any three numbers makes a triangle. The tester types in any three numbers and they expect the program to give a result of either equilateral, isosceles, or scalene. I do not expect them to be trigonometry experts, but I do expect them to have a notion of impossible triangles. For example, the numbers 1, 1, 10 can never be three lengths for the sides of a triangle because the sides will never connect, but if they don't realize this, I ask them to come to the whiteboard and draw it for me. Most testers quickly realize it is impossible and so I give them the oracle: "for any triangle to be legitimate, the sum of the two smallest sides must be greater than the third." From that point on, I watch how their testing changes.

A remedy for the notion of bad oracles is to get other stakeholders' ideas on whether or not a bug is a bug. Raising issues is a good way to do this, which is why I have a section on the whiteboard for it in the audition. It's a way to MIP a bug (mention-in-passing) as if you're standing in the hallway informally talking to the Program Manager to say "by the way, I saw this weird thing…", giving you the opportunity to hear them talk about how the design should work without risking your credibility.

Very few testers notice in the audition that there is a second revision of the Triangle Program in the directory – a version that actually draws the triangle. This could serve as an oracle to get a bearing on what triangles are legitimate and what triangles are invalid.

1) Premature Celebration

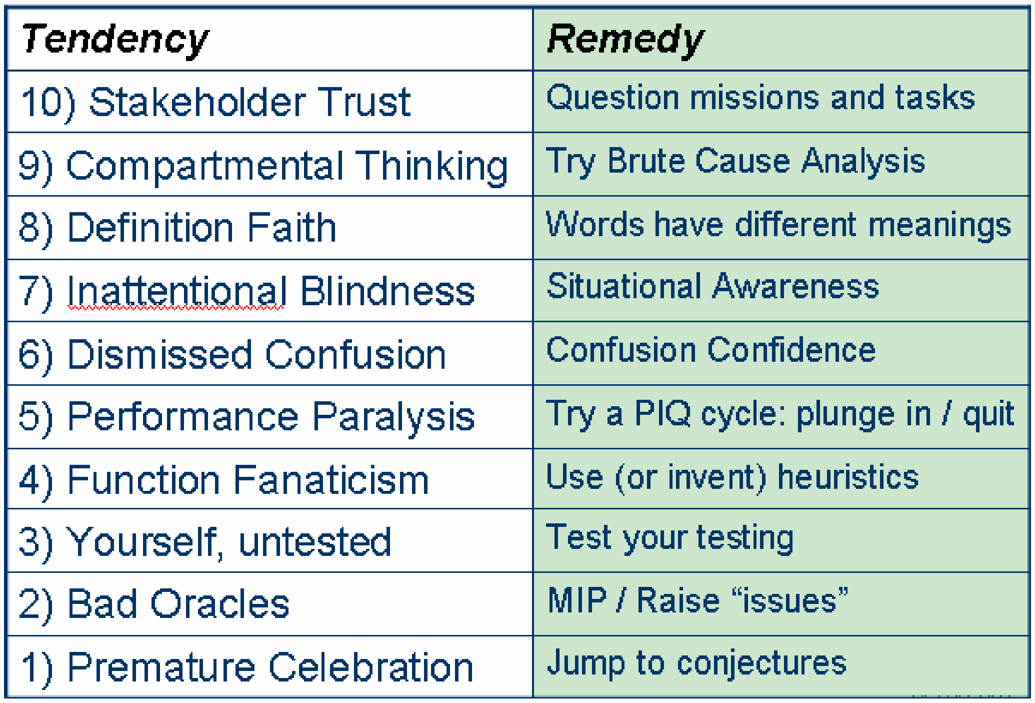
Remedy:

It wasn't hard to find the run-time I showed in the traps section above, and yes, it's fine for a tester to feel validated that they've done their job to find important problems quickly, but what if they started the party a bit later? How do they know it's not just the tip of the iceberg and they could get the platform to bluescreen with a little more prodding? At this point, I'm looking to see if testers jump to *conjectures*, not conclusions.

There is a heuristic that I've been practicing to help me avoid this trap more consistently. It is called "Rumble Strip." It takes its name from the grooves in the shoulder of a highway meant to make your car tires noisy if you go over them – an audible alert that you are going off into the shoulder and that bad things might happen if you continue on that path. It is meant to get us back on the road. But what if as testers, we used the rumble strip as a guide to make bad things even worse – to find those better problems just a mouse click or two away?

This is where a hacker mentality comes in handy. In other words, yes, you found a hang or a crash, but before you report it, can you exploit the state of the machine to find something even worse? In other words, can you continue to give the program input or cause it to reveal a memory leak? Think of any bug you find like an invitation to a larger party with VIPs to which you are always invited. It begins with the discovery of a bug, continues with newly formed conjectures and follow-up tests, and may end with the celebration of the year when you find an even more severe bug than what you started with.

Summary:



Tendency	*Remedy*
10) Stakeholder Trust	Question missions and tasks
9) Compartmental Thinking	Try Brute Cause Analysis
8) Definition Faith	Words have different meanings
7) Inattentional Blindness	Situational Awareness
6) Dismissed Confusion	Confusion Confidence
5) Performance Paralysis	Try a PIQ cycle: plunge in / quit
4) Function Fanaticism	Use (or invent) heuristics
3) Yourself, untested	Test your testing
2) Bad Oracles	MIP / Raise "issues"
1) Premature Celebration	Jump to conjectures

Caveats

Data: It's important to note that this paper is non-scientific. Although the Triangle Program has a hidden logging feature that captures their mouse clicks and keystrokes, those tests do not show the full range of a candidate's testing ability. The traps I have seen are not limited to the audition, but are also present in the phone screen exercises and the written exercise portion of the interview process.

Bias: One of the flaws in my research is my bias toward extrovert testers – testers who can openly communicate their thinking in real time and tend to enjoy (and get fueled by) interaction. Introverts, on the other hand, tend to need solitude to refuel. This was pointed out to me last year, so I amended the audition to include time for me to leave the room and let testers consider a problem before demonstrating a technique. It also encouraged me to incorporate a written portion of the interview well before the audition.

Results: The audition is used in conjunction with a phone screen, a written interview, and a one-hour survey of their work experience. When they get to the final audition phase, they are ready for a test of their skill and we use the audition as a project simulator. People who do well in the audition usually become hired because it is the toughest part of the interview process. Those that make it through have caused me to have confidence that they can work on any project at the lab, no matter the nature, mission, or context because they will have shown questioning ability, technical ability, test idea creation balanced with test execution in a short amount of time.