Testers as Their Own Worst Enemies (and how to fix that)

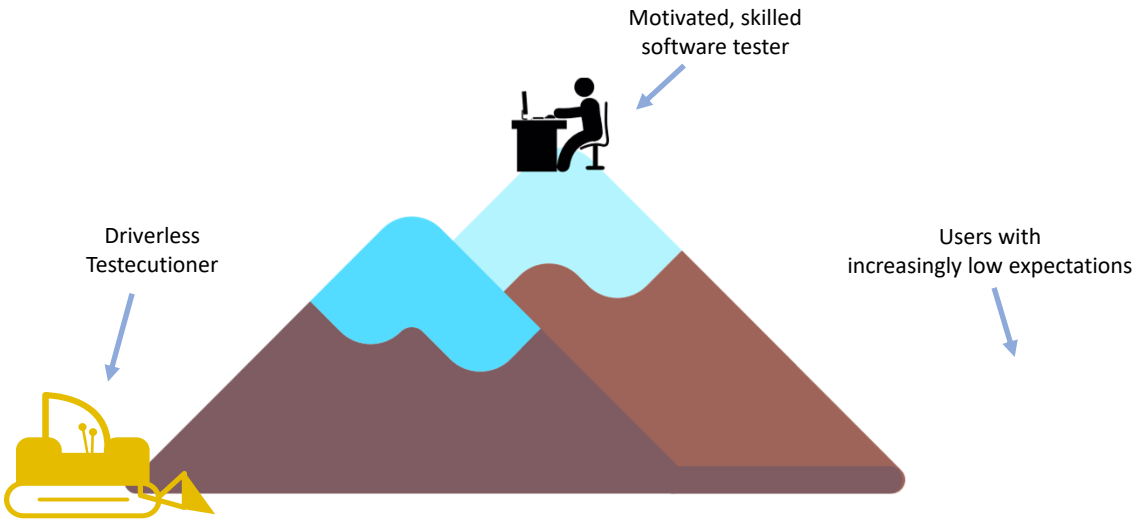
A Personal View on
Avoiding Mistakes that Testers Sometimes Make

Michael Bolton
DevelopSense
http://www.developsense.com
@michaelbolton
michael@developsense.com

with material from
James Bach
Satisfice
http://www.satisfice.com
@jamesmarcusbach
james@satisfice.com

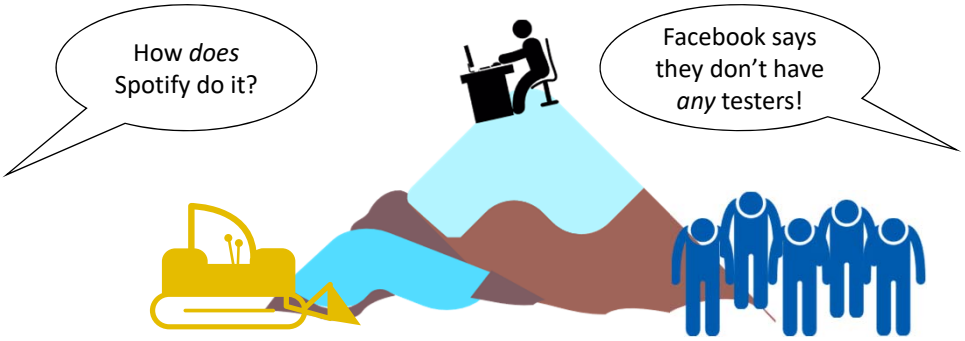
Episode IV
Not very long ago at all,
in a galaxy far too close to here…

Demand for skilled testing is eroding.

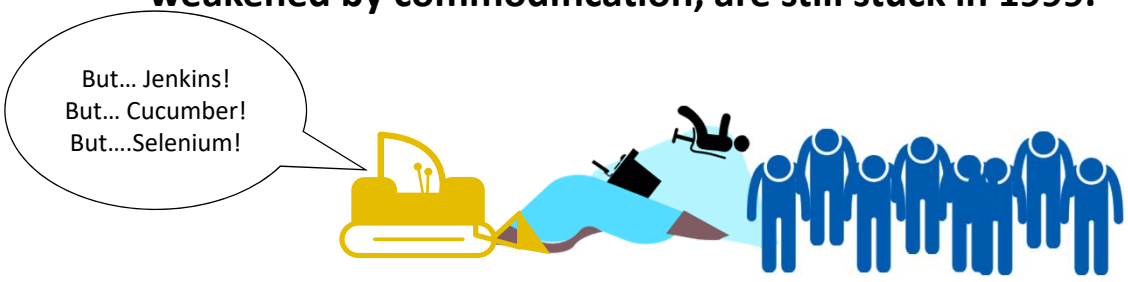


Demand for skilled testing is eroding:
shiny “automation” on one side, and low quality standards on the other.

Demand for skilled testing is eroding
shiny "automation" on one side,
and low quality standards on the other.
The software industry has become dazzled by values and practices appropriate for experimental, non-critical software.



Demand for skilled testing is eroding:
shiny "automation" on one side,
and low quality standards on the other.
The software industry has become dazzled by values and practices appropriate for experimental, non-critical software.
Meanwhile, many corners of testing culture, weakened by commodification, are still stuck in 1999.



Problem:

You can't release a product without developers,
but you *could* release one without testers.

Problem:

Developers have it easy!
They can point to a build,
or to a running program,
or even to dazzling programmed checks!
But testing work is mostly invisible and intangible.

Big Problem:

Lots of managers and developers
don't understand testing.

Problem:

If we want managers to support testing work,
we have to talk about testing and make it legible.

Problem:

If we don't *do* excellent testing
talk clearly about it,
and *make it legible*,
we risk undermining ourselves and each other.

Why I'm becoming a grumpy old guy:

Increasingly, testing is confused with "checking builds".

Our fixation on "test automation" (and now on "AI") is causing some of us to lose connection with the human, social purposes of software development and testing.

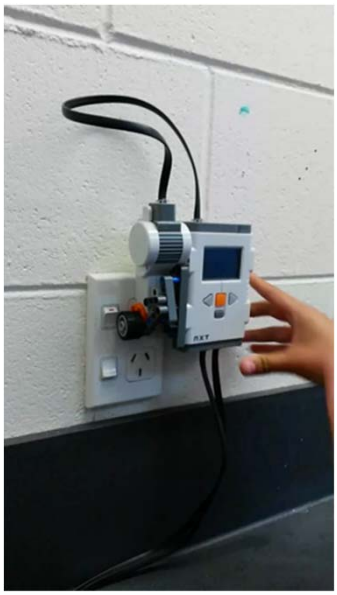
Tools are cool! **We should use them.** We should use them **a lot** to help us develop an understanding of our products.
Tools can help us to be powerful.

Push-button builds afford deep testing whenever we like. **Cool!**

But I'm seeing stuff that looks like elaborate attempts to **avoid making contact** with the software, our clients, our customers, and our mission.

Holding a machine and pressing its button so it can press a button that was made for a human to press without a machine.

GUI automation at its finest.



https://www.youtube.com/watch?v=LmaHRCPkiQs

Problem:
Delivering Bad News

~~Testers shouldn't rock the boat.~~
~~Can't we all just get along?~~

Effective testing is socially disruptive to some degree. That disruption must be acknowledged and managed.

Problem:
Delivering Bad News

~~Testers shouldn't rock the boat.~~
~~Can't we all just get along?~~

Critique of our craft is also socially disruptive.
But it's *important*.
We must anti-fragilize our ideas and ourselves.

Good news:

We can address these problems
for our colleagues and for ourselves.
We can reconnect with our clients and our products.
Doing that starts with US, right here, right now.

I suggest:

1. Focus on why our client and teams need us, and why they hire us.
2. Talk about ourselves and what we do in ways that support us.
3. Put the tester (not tools, not test cases, not artifacts, not process models) at the centre of testing.

Testing Is *Social Science*

Harry Collins
Author
Artifictional Intelligence
Tacit & Explicit Knowledge
Rethinking Expertise

"Computers and their software are two things. As collections of interacting cogs they must be 'checked' to make sure there are no missing teeth and the wheels spin together nicely.

"Machines are also 'social prostheses', fitting into social life where a human once fitted. It is a characteristic of medical prostheses, like replacement hearts, that they do not do exactly the same job as the thing they replace; the surrounding body compensates.

Abstract, "Machines as Social Prostheses", EuroSTAR 2013

Testing Is *Social Science*

Harry Collins
Author
Artifictional Intelligence
Tacit & Explicit Knowledge
Rethinking Expertise

"Contemporary computers cannot do just the same thing as humans because they do not fit into society as humans do, so the surrounding society must compensate for the way the computer fails to reproduce what it replaces.

"This means that a complex judgment is needed to test whether software fits well enough for the surrounding humans to happily 'repair' the differences between humans and machines. This is much more than a matter of deciding whether the cogs spin right."

In other words…

Is the product we've got
good enough
for people to be happy with it?

Why have testers?
Because management wants
an expert answer to this question:

Are there problems
that threaten the value of the product,
or the on-time successful completion of the project?

We don't assure quality,
***but* we *help* the people who do.**

~~Testing is about quality control, or quality assurance. Testers are quality gatekeepers.~~

Testers can't assure quality.
The difference between what we're *called* and what we *do* causes confusion for clients and pain for us.

By finding trouble now, we help people avoid trouble later.

~~Testing is about quality control, or quality assurance. Testers are quality gatekeepers.~~

Testing is about *exploring and investigating the **risk** of loss or harm to people who matter.*

We can find problems in lots of places!

~~There might be an error in the code! There might be an error in the build!~~

There can be *many problems* in the relationships between people and the product.

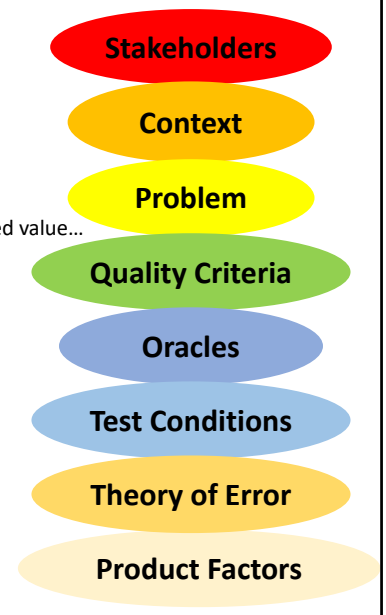
Risk is about stories and models.

~~Risk analysis is about estimating and calculating probability times impact.~~

Risk analysis is about identifying many factors that help us to develop rich, vivid stories about problems and potentialities that prompt us to take action.

Risk Story Elements

- Some PERSON(S)
  - user, customer, developer, tester, businessperson, bystander…
  - (a group, a business, a community, society at large…)
- will EXPERIENCE
  - be affected, in the context of an event or situation, at least once by …
- a PROBLEM
  - that leads to bad feelings (annoyance, frustration, confusion), loss, harm, or diminished value…
- with respect to SOMETHING DESIRABLE
  - like capability, reliability, performance…
- that CAN BE DETECTED
  - by a feeling, principle, tool, comparison to a document or picture…
- in SOME SET OF CONDITIONS
  - perhaps always, perhaps only sometimes,…
- because of a VULNERABILITY
  - a bug, a missing feature, an inconsistency…
- in the SYSTEM
  - some result, process, component, feature, environment…



We learn on behalf of others!

~~Testing is showing that the product works.~~

Testing is learning about the product; gaining experience with it; searching for problems; finding them; reporting on them.

We're problem-finders!

~~Testing is about confirming that the product works.~~

Testing is about *disconfirming*; discovering how the product *doesn't* work, or *might not* work.

We're skilled investigators!

~~Testing is about confirmation.~~

Testing is about *investigation.*

About confidence…

~~Testing is about building confidence.~~

Testing is about demolishing *unwarranted* confidence.

We're professional skeptics.

~~Testing is about reducing damaging uncertainty.~~

Testing is about *preserving appropriate skepticism.*

We use tools to help us learn.

~~Testing is all about the button-pushing, which can be done more quickly by machinery.~~

Testing is about *learning*, which can only be done by humans with intentions.
But tools can be powerful aids to testing.

Call This "Checking", Not Testing

operating a product algorithmically to check specific facts about it…

Think "spelling checker"

Or "compiler checks"

means

Observe → **Evaluate** → **Report**

Interact with the product in specific, *algorithmic* ways to collect specific observations.

Apply *algorithmic* decision rules to those observations.

Report the outputs of the evaluations *algorithmically*.

A check can be performed…

by a machine that *can't* think
(but that is quick and precise)

by a human who has been told *not* to think
(and who is slow and variable)

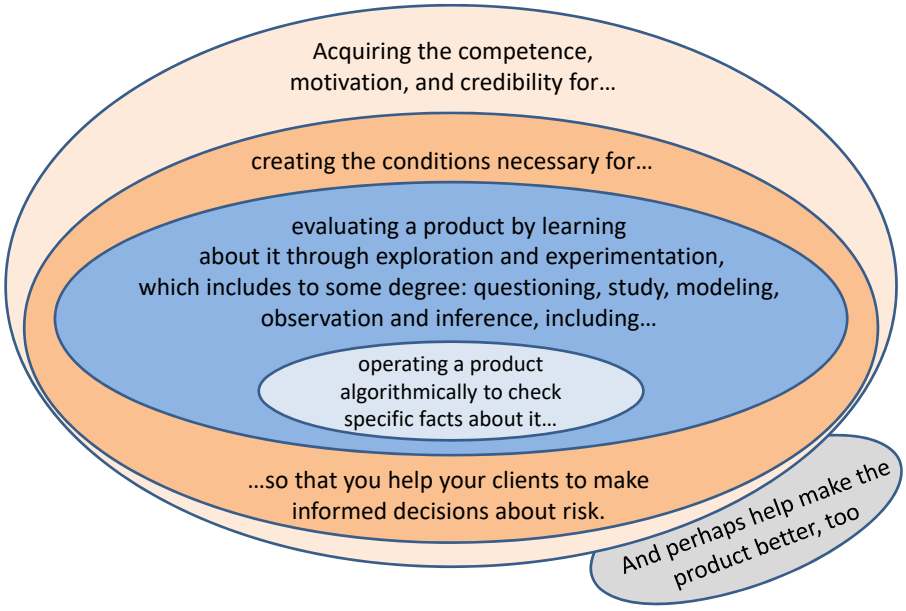
Notice that "quick" and "slow" here refer only to the speed of observable behaviours and algorithmic evaluations.
The machine is *infinitely* slow at recognizing unanticipated trouble.

Testing Is *More Than Checking*

- *Checking* is okay, but people tend to focus algorithmic checks to confirm what we know or hope to be true.
  - too bad, because checking can also be used *inside* exploratory work
- To understand our products and the risk of problems that matter to people, we must *do more* than output checking; we must *test*.

See http://www.satisfice.com/blog/archives/856

Testing is…



Why it's important to distinguish testing and checking

- Because *checking* is mechanistic. It can be made completely **explicit**, encoded, and automated. It is *inside* testing. It is a *tactic* of testing.

People don't confuse biting with eating!

Biting can be done by tools… but eating can't.

Why it's important to distinguish testing and checking

- Because *checking* is mechanistic. It can be made completely **explicit**, encoded, and automated. It is *inside* testing. It is a *tactic* of testing.
- Because *testing* involves **tacit** and **social skills** that cannot be encoded. Testing skills must be developed through socialization, practice, and increasingly challenging work, not via rote procedures.
- Because talk about efficiency and effectiveness makes for *very* different conversations when we're talking about explicit vs. tacit skills.
- Because for checking to be *truly* excellent, it must be embedded in excellent testing. Developing valuable checks requires skill!
- Programmers have resisted marginalization for years!
  (They no longer call compilers "autocoders" and programming languages are no longer called "autocodes".)

Effective checking *needs* skilled, skeptical testers.

Although *checking* can be automated, *testing* cannot.

Testing about the brains (not hands)!

Testing is neither manual nor automated.

Testing is neither manual nor automated!

Manual Doctoring

Manual Parenting

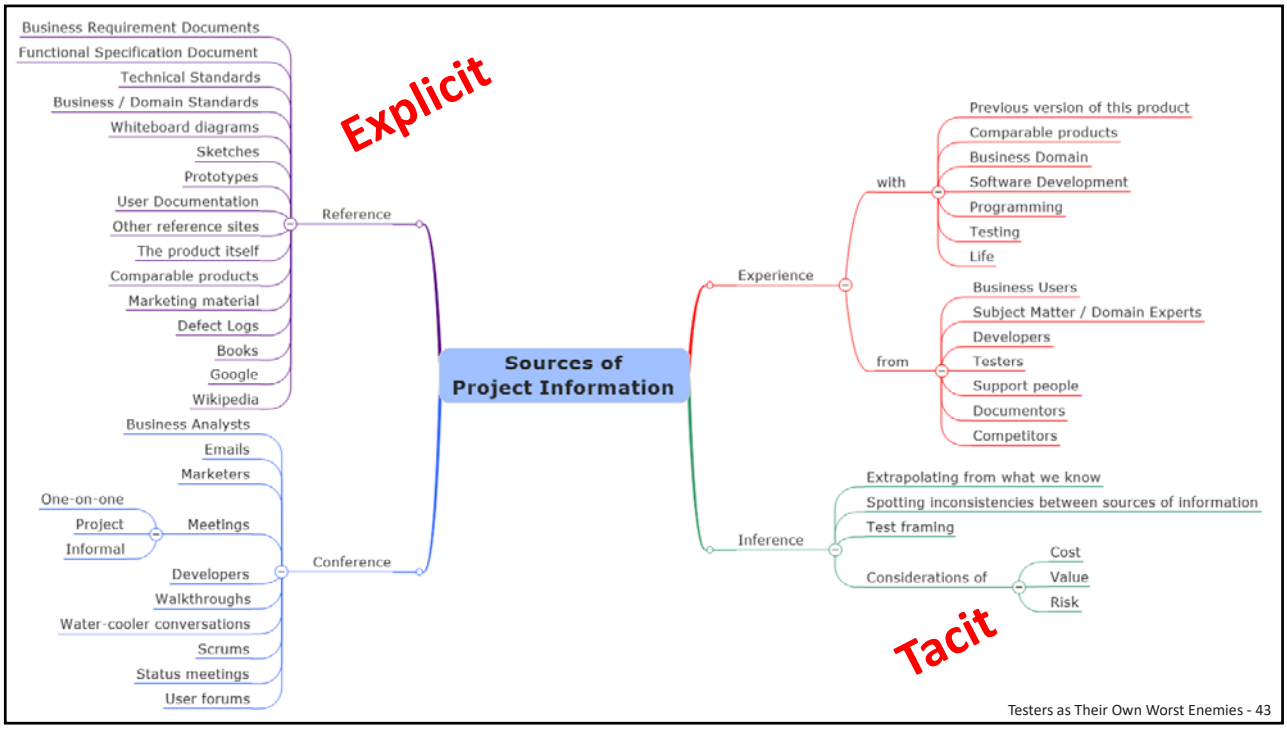
Manual Research

Manual Management

Requirements are all around us, not just in requirements documents.

~~Requirements = requirements documents!~~

Some of the requirements are described, *to some degree*, in the requirements document.



Testing is so much more than test cases!

~~"We write test cases. Passing test cases show the product is good. Failing ones show the product is bad."~~

Testing is not about correctness, or about passing or failing test cases.

Skilled testers focus on two questions.

To themselves, referring to the product:

Is there a problem here?

To the team and to management, referring to issues and obstacles:

Are you okay with this?

Reminder: Management wants an expert answer to this question:

Are there problems

that threaten the value of the product,
or the on-time successful completion of the project?

We develop powerful test strategies!

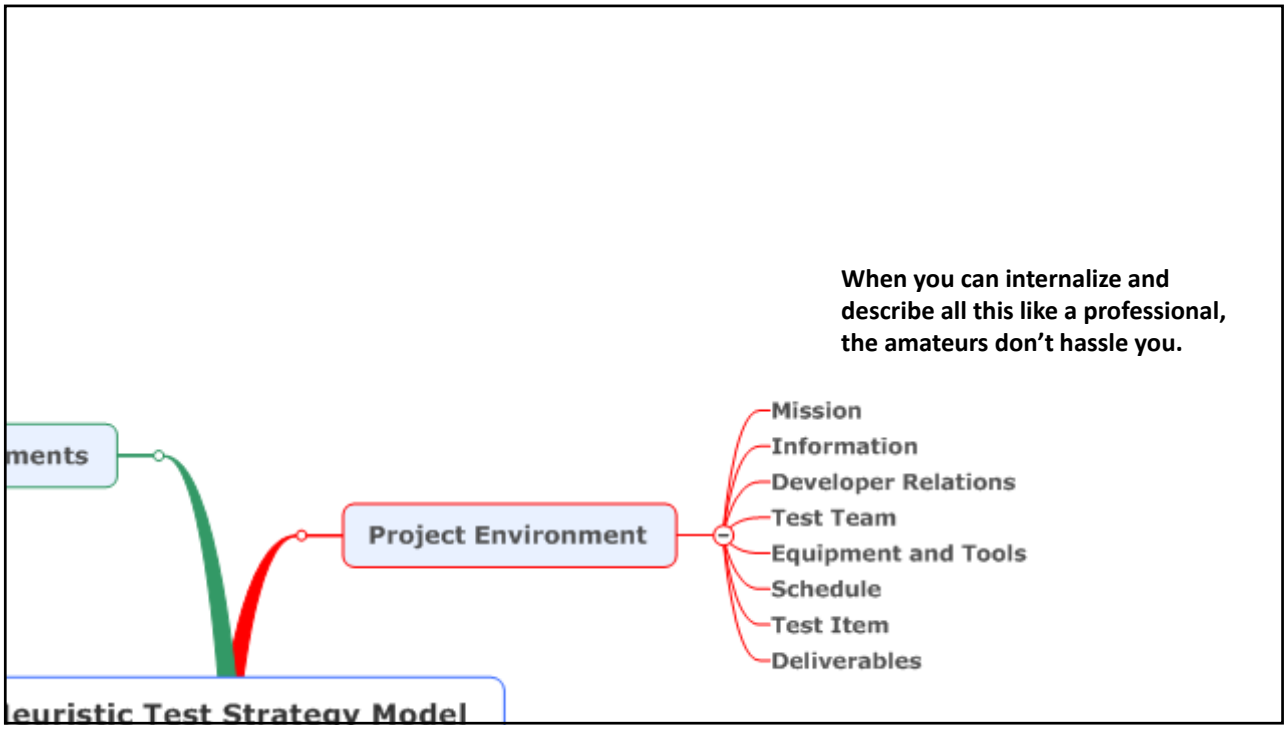
~~"We read the specs, and then we write test cases."~~

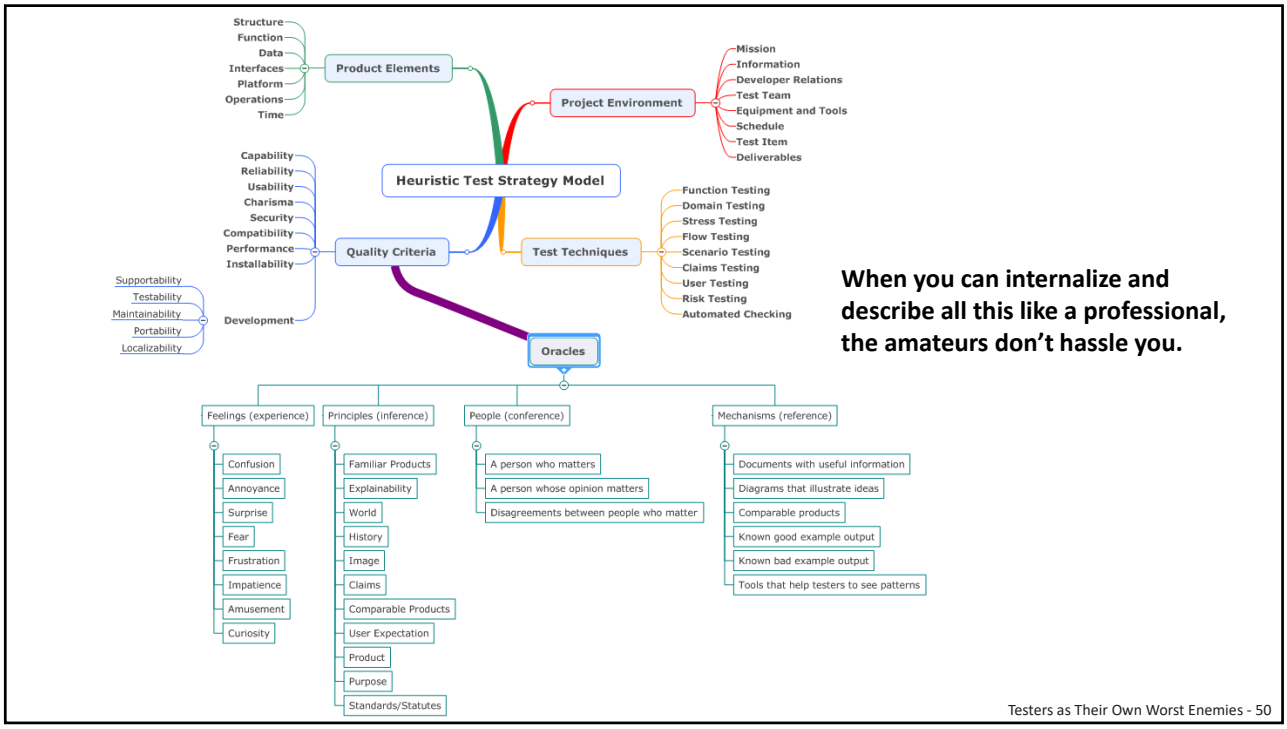
Excellent test strategy requires rich models of context, quality criteria, product factors, oracles, and test techniques.

Be a credible testing expert!

~~"My manager says we have to write test cases!"~~

Managers resort to mythodology of "best practices" and bogus metrics when they don't have credible alternatives. *Be a skilled, reliable investigator.*





We *perform* the testing!

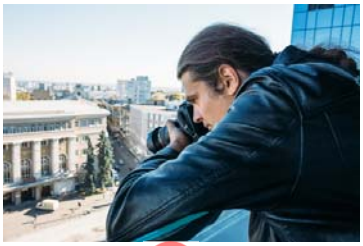
~~Testing is all about test cases.~~

A test is not an artifact, like a musical score.
A test is *a performance*, like a live concert.

Testing is not test cases!

Piloting Cases

Anthropology Cases



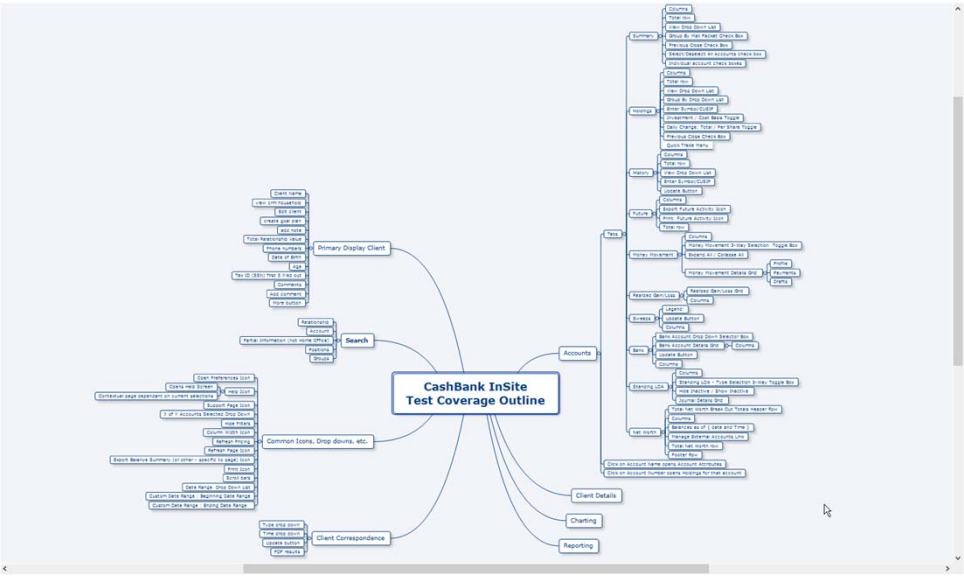
Journalism Cases

Management Cases

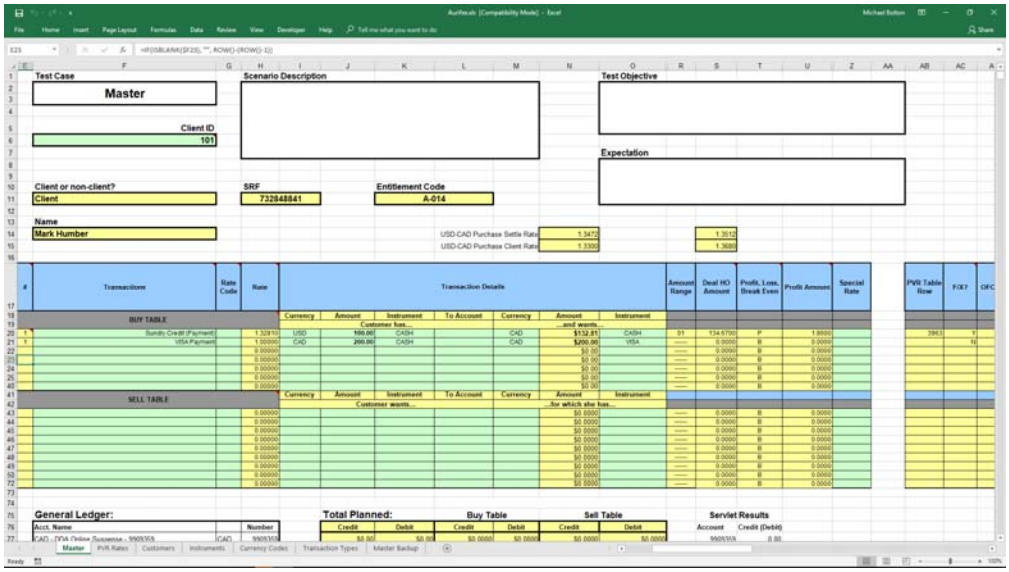
Why to resist framing testing as test cases:

- Testing is about exploration, experimentation, discovery, investigation, learning, and reporting.
- *Test cases* tend to focus on output checking, confirmation and demonstration; showing that something *works*, rather than prompting a search for problems that matter.
- When people turn testing into test cases, they start counting them.
- When people turn testing into *counting*, the information loss, dysfunction and distortion begins.

Alternatives to Test Cases: Coverage Outlines



Alternatives to Test Cases: Variable Transaction Tables



Alternatives to Test Cases: Open Procedures

3.2.2 Fields and Screens

3.2.2.1 Start the Zapper Box and the Control Box. (Vary the order and timing, retain the log files, and note any inconsistent or unexpected behaviour.)

3.2.2.2 Visually inspect the displays on each box and **VERIFY** conformance to the requirements specifications. Remain alert for the presence of any behaviour or attribute that could mislead or confuse the operator, or impair the performance or safety of the product in any material way.

3.2.2.3 With the system settings at *default* values, change the contents of every user-editable field through the range of all possible values for that field. (e.g. Use the knob to change the session duration from 1 to 300 seconds.) Visually **VERIFY** that appropriate values appear and that everything that happens on the screen appears normal and acceptable.

3.2.2.4 Repeat 3.2.2.3 with system settings changed to their *most extreme* possible values.

3.2.2.5 Select at least one field and use the on-screen keyboard, knob, and external keyboard respectively to edit that field.

Alternatives to Test Cases: More Specific Procedures

3.5.2.3 From the power meter log file, extract the data for the measured electrode. This sample should comprise the entire power session, including cooldown, as well as the stable power period with at least 50 measurements (10 seconds of stable period data).

3.5.2.4 From the Control Box log file, extract the corresponding data for the stable power period of the measured electrode.

3.5.2.5 Calculate the deviation by subtracting the Control Box's reported power for the measured electrode from the corresponding power meter reading (use interpolation to synchronize the time stamp of the power meter and Control Box logs).

3.5.2.6 Calculate the mean of the power sample X (bar) and its standard deviation (s).

3.5.2.7 Find the 99% confidence and 99% two-sided tolerance interval k for the sample. (Use Table 5 of NIST* SOP-QAD-10, or use the equation below for large samples.)

3.5.2.8 The equation for calculating the tolerance interval k is: $$k = \sqrt{\frac{(N-1)\left(1+\frac{1}{N}\right)Z^2_{(1-p)/2}}{\chi^2_{\gamma,N-1}}}$$

where $\chi^2_{\gamma,N-1}$ is the critical value of the chi-square distribution with degrees of freedom, $N-1$, that is exceeded with probability γ and $Z_{(1-p)/2}$ is the critical value of the normal distribution which is exceeded with probability $(1-p)/2$.

* See NIST Engineering Statistics Handbook.

Alternatives to Test Cases: Learning-Focused Charters

- …for Intake Sessions (Goal: negotiate mission)
  - "Interview the project manager. Ask about particular concerns or risks."
  - "Read through all new use cases, and discuss with developers."
- …for Survey Sessions (Goal: learn product)
  - "Familiarize yourself with the product by performing a UI tour. Create a Product Coverage Outline."
- …for Setup Sessions (Goal: create testing infrastructure)
  - "Develop a library of mindmaps for each major feature area. Use SFDIPOT as a checklist for coverage analysis."
  - "Identify and list all the error messages in the product."
  - "Develop a scenario playbook with SMEs and other testers."
  - "Review use cases, and for each, add several ways in which the user could accidentally or maliciously misuse the feature."

Alternatives to Test Cases: Deep Testing Charters

- …for Deep Coverage Sessions (Goal: find the right bugs)

  "Perform scenario testing based on the scenario playbook."

  "Run state-machine-based tours to achieve double-transition state coverage. Find possibilities for programmed checks."

  "Perform steeplechase boundary testing on major data items."

  "Help developers to set up automated checks for the continuous integration pipeline."

  "Generate each identified error message in the product. Look for mismanaged state and error recovery problems, confusing or unhelpful user messages, and missing error codes."

  "Develop scripts (working below the GUI) to run transactions continuously and graph results and timings. Make sure many transactions (15%? like production logs?) include invalid data that should be handled and rejected."

Alternatives to Test Cases: More Formalized Charters

PROCHAIN ENTERPRISE **SCENARIO TEST CHARTER**

UP2: "Check status and perform buffer update"

Theme	You are a project manager. You need to update your project to prepare your weekly report on project status.
Setup	- Log in with a user account set up with project manager rights. - Buffer consumption for one of the projects should ideally be in the yellow or red. - At least some of the projects should have multiple project buffers.
Activities	❑ View the Standard Projects Status Chart (or custom chart), filter on a set of projects (and turn on name labels). Start a second session in a window next to the first one (log in as the same user), and filter for the same project set. Now you have two project status charts that you can compare. ❑ Pick one project as "yours". Now, compare status history of your project to others. Explore the other project details in any way necessary to account for the *differences* in status. ❑ View all impact chains for your project, and for some of those tasks: - view task details - view task links - view task load chart ❑ If other testers are making task updates during your test session, review those changes and modify some of them, yourself. Otherwise, make at least a few updates of your own.

We tell a rich testing story.

~~The testing story is all about bugs.~~

The testing story has three parts: the status of the product; how testing has been done; and what makes testing harder and slower.

image credit: istockphoto.com

The Testing Story Is Three Braided Stories

A story about the status of the PRODUCT…
…about what it does, how it failed, and how it might fail...
…in ways that matter to your various clients.

Bugs

A story about HOW YOU TESTED…
…how you operated and observed the product…
…how you recognized problems and their significance…
…what you have testing so far *and have not tested yet*…
…what you won't test at all (unless things change).

Oracles

Coverage

A story about how GOOD that testing was, or could be…
…the risks and costs of testing or not testing…
…how testable (or not) the product is…
…what made testing harder or slower…
…what you need and recommend for faster, higher-value testing.

Issues

Focus on *describing coverage*.

~~The testing story is about how many test cases we have run.~~

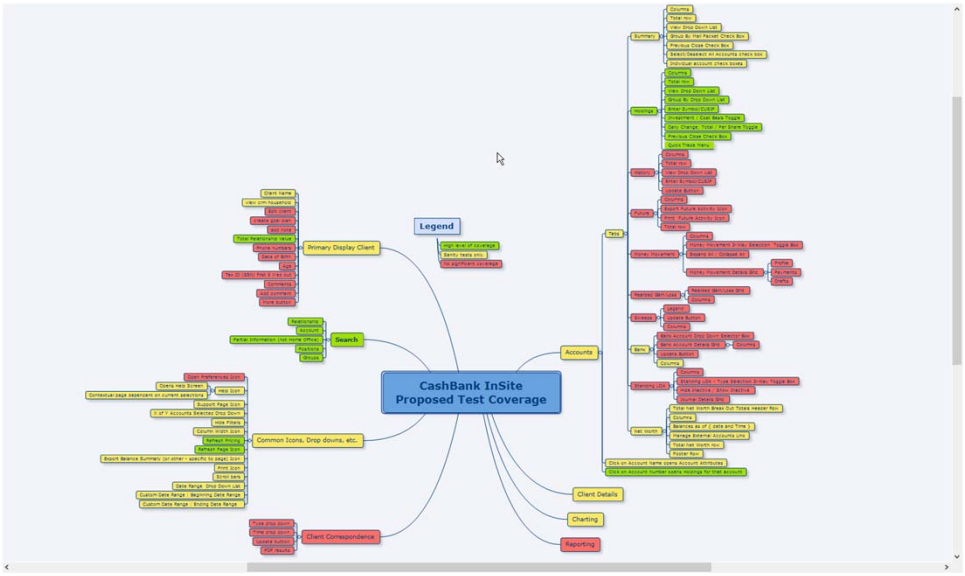
The testing story is about the *what we have covered so far* and what else *could* be covered.

Alternatives to Test-Case-Based Reporting:

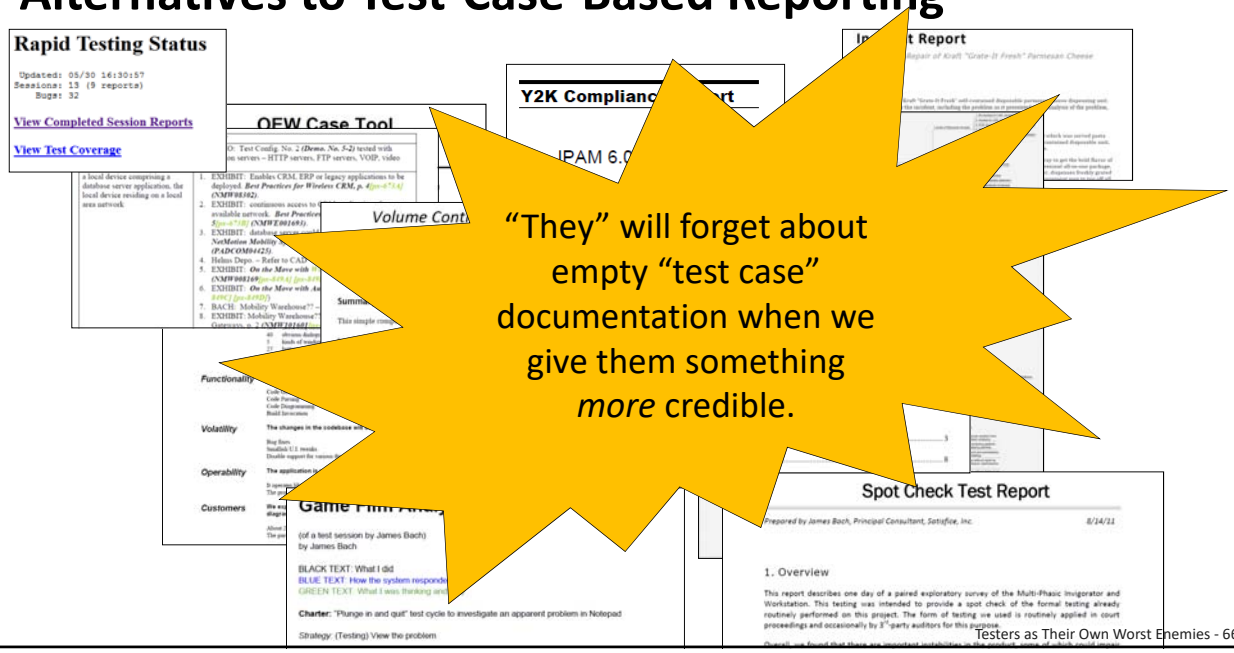
Session-Based Test Management Debriefs: PROOF!

- **P**ast
  - What happened during the session?
- **R**esults
  - What was achieved? What was covered?
- **O**bstacles
  - What got in the way or slowed things down?
- **O**utlook
  - What's next? What remains to be done?
- **F**eelings
  - How does the tester feel about all this?

Alternatives to Test-Case-Based Reporting



Alternatives to Test-Case-Based Reporting



"They" will forget about empty "test case" documentation when we give them something *more* credible.

Focus testing and checking on *risk*.

~~"But we have to run all the tests after each build!"~~

Running *all* tests after each build is probably not a well-considered, risk-focused test strategy.

Consistent Regression Problems Are *Signs of Big Trouble*

- If you see a consistent pattern of regression… or worry about it *obsessively*
  - failing checks or tests probably aren't your biggest problem
  - more likely, the issue is that you've got favourable conditions for regression to happen
  - testing cannot fix this problem; at best testing can only *detect* some regression bugs
  - the programmers are probably working too quickly to understand what's happening

THAT'S A SEVERITY-ZERO PROJECT RISK. REPORT IT.

It's OK! Learn to code!

~~I can't code!~~
~~No one will hire me!~~

You haven't learned to code *yet*.
You *could* learn to write
some useful code.
It's not that hard. Really.

…or use your social superpowers!

~~I can't code!~~
~~No one will hire me!~~

You *could* develop expertise in
analysis and experimentation…
and charm developers into
helping you when you need
tools.

We don't *need* AI, but AI needs us.

AI will replace us!

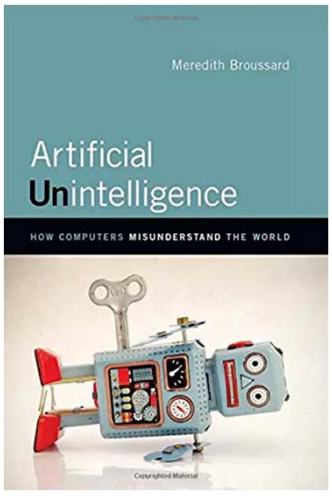
Think of AI as "algorithm improvement"; sophisticated software *that its developers admit they don't understand*. We'll need to test the daylights out of it.

We don't *need* AI, but AI needs us.

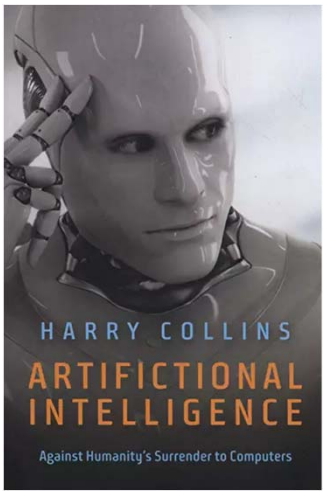
~~AI will replace us!~~

We must keep laser-focused on problems in code, training data, testing data, **and ethics,** lest AI becomes Arrogant Incompetence and Amplified Injustice.

Well-reasoned Cautions



Artificial Unintelligence
Meredith Broussard



Artifictional Intelligence
Harry Collins

We use tools for all kinds of stuff!

~~Tools are just for GUI checking, right?~~

Producing test data; obfuscating or cleaning production data for privacy reasons; generating interesting combinations of inputs; generating and representing state and flow models…

We use tools for all kinds of stuff!

~~Tools are just for GUI checking, right?~~

Setup, configuration, and environment management; submitting transactions; automated checking; creating mocks and simulators; probing the internals; monitoring at the interfaces…

We use tools for all kinds of stuff!

~~Tools are just for GUI checking, right?~~

Sorting, filtering and parsing; visualizing; internal consistency checks; applying oracles; performing statistical analysis…

We use tools for all kinds of stuff!

~~Tools are just for GUI checking, right?~~

Recording activities; documenting procedures; preparing reports; presenting reports; Mapping strategies; identifying coverage; organizing time and effort; retaining information…

We help people decide when *they're* done.

~~When will the testing be done?~~

Testing is done when managers and developers are satisfied that there is no more important development work to be done.

We help people decide when *they're* done.

~~When will the testing be done?~~

Testing is never *done*; it only *stops*. Testing stops when managers decide they can make **their** informed shipping decision.

We talk about quality.

~~How should we measure quality?~~

You can't measure quality. You can measure attributes that might have a bearing on quality

But you can report on quality, and you can discuss it.

We *discover* problems.

~~"I break the software."~~

The software was broken when we got it. Testers break *illusions* about the software.

Why it's important NOT to say "I break the software"

When you say "I break the software", you set yourself up for a potential public relations problem. Others may "repair" "I break the software."

- "The software was fine **until the testers broke it.**"
- "We could ship our wonderful product on time **if only the testers would stop breaking it**."
- "Normal customers wouldn't have problems our wonderful product; it's just that **the testers break it**."
- "There are no systemic management or development problems that have been leading to problems in the product. Nuh-uh. No way. **The testers broke it.**"

By finding, we help problem solvers.

~~Testing is about preventing problems.~~

Development solves and prevents problems. Testing is about *discovering problems* that *weren't prevented and have not yet been solved.*.

We're ready to help any time.

~~Testing is about preventing problems.~~

Early in development, testers can *help* by *anticipating risks and problems* in a way that *helps* developers to prevent them.

Why it's important NOT to say "I prevent problems"

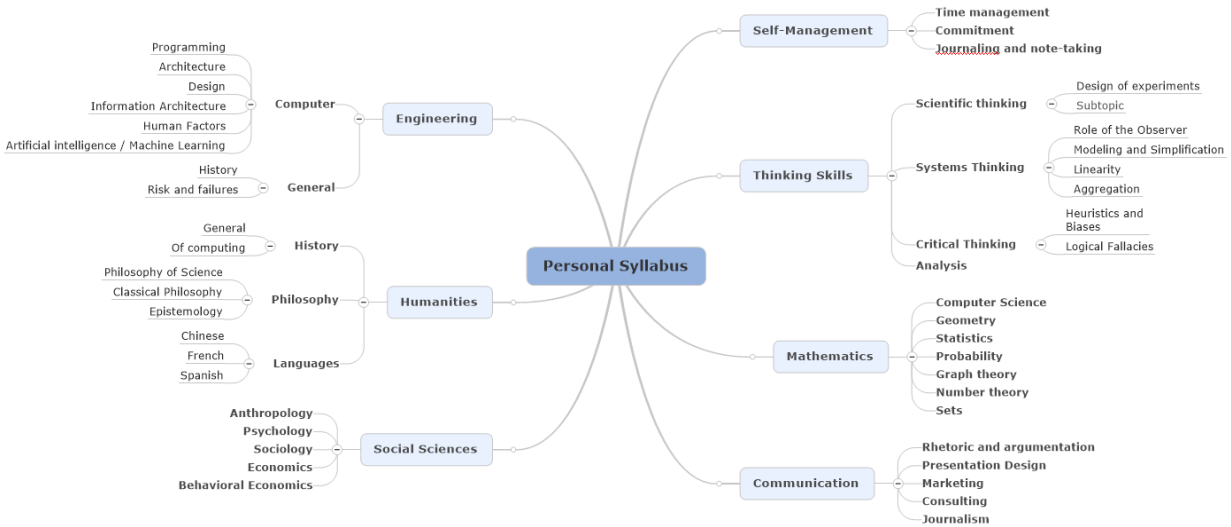
It's perfectly okay to say "I *help* people to prevent problems." We're here to help! We help!

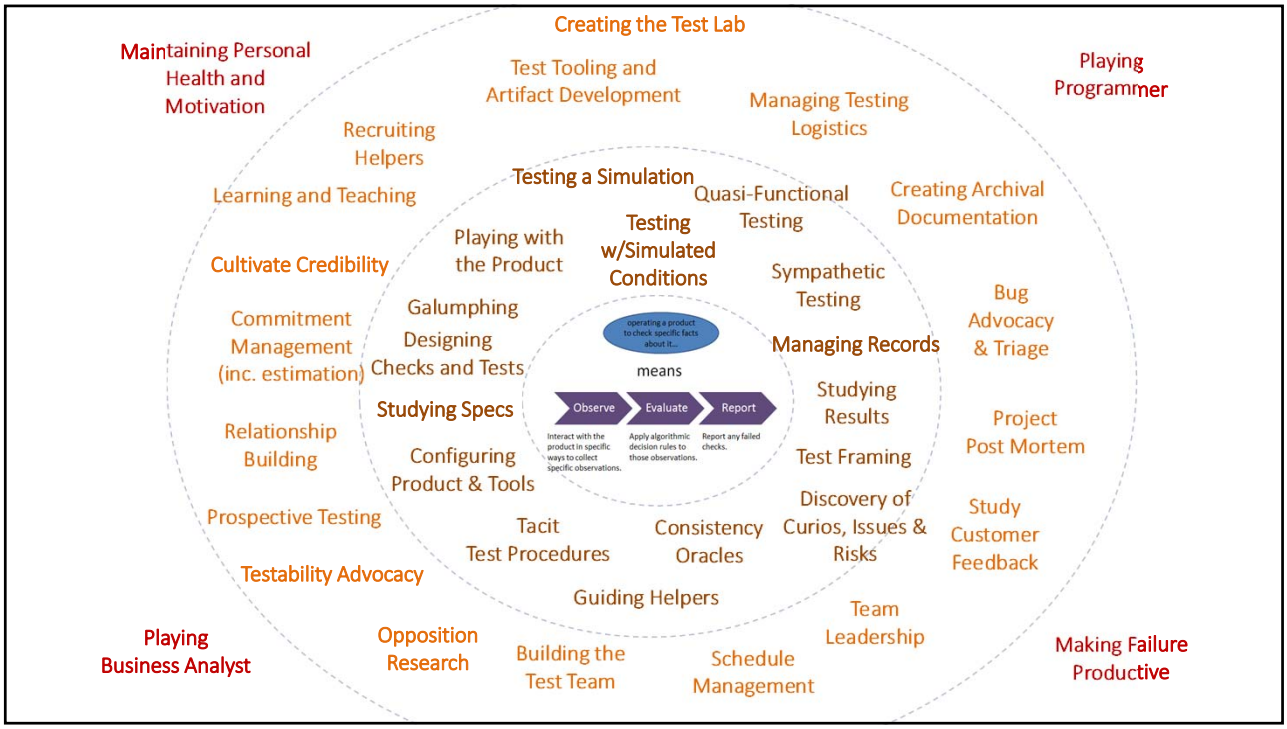
When you say "I prevent problems", there's a strong possibility you are over-reaching.

- It's important to honour the roles of the developers and the designers, and to remain humble about our own. We don't put the quality in directly. They do.
- Problems occur because of a variety of contributing factors. The same is true of the prevention of problems. It's a team, right?

One More Tiny Catch

There are lots of skills and tasks to work on.





Postscript

Speaking Imprecisely

~~"flaky tests"~~

The checks aren't flaky.
But your explanations about inconsistency might be.

Don't Fear the Certification Monster

~~I don't have a certification!~~
~~No one will hire me!~~

Put "I am not ISTQB certified, and I'm happy to explain why" on your resume.
If they reject you for *that*, you don't want to work there anyway.

Speaking precisely helps!

~~"Why is testing so expensive?"~~

"Why is *all of development* so expensive? Let's make a more testable product."

Speaking precisely helps!

~~"Why is testing taking so long?"~~

"Why is *all of development* taking so long? Let's make a more testable product."

Speaking precisely helps!

~~"Can't we just automate all the testing?"~~

"Can't we just automate *all the development*? Let's ask the developers what they think about that."

Remember the social dimensions

~~We don't need roles on Agile teams.~~

Roles help to focus skills, commitments, and relationships.

A Healthy Role Institutionalizes…

- **Competence:** Increases skill over time.
- **Focus:** Marshals energy and concentration to solve difficult problems well; economy of scale.
- **Anticipation:** Identifies future needs and potential problems before its too late.
- **Accountability:** Accepts responsibility for outcomes within scope of the role.
- **Coordination**: Minds the interface with other roles.

See “On a Role” http://www.developsense.com/blog/2015/06/on-a-role/

What is the testing role?

- *To test* is to evaluate a product by learning about it through exploration and experimentation.
- When someone is testing, that person has adopted (if only for that time) a testing role.
- *A tester’s role* is to
  - to develop one’s self as a tester
  - connect with the clients of testing
  - prepare for testing
  - perform testing
  - report the results of testing.

Talking More Clearly About Testing

Try replacing…	with…
Verify that…	Challenge the belief that…
Validate	Investigate
Confirm that…	Find problems with…
Show that it works	Discover where it *doesn't* work
Pass vs. fail…	Is there a problem here?
Test case	Test conditions and test ideas
Counting test cases	Describing coverage
Automated testing	Programmed checking
Test automation	Using tools in powerful ways
Use cases	Use cases AND *mis*use cases AND *ab*use cases AND *obt*use cases…
KPIs and KLOCs	***Learning from every bug***

Talking More Clearly About Testing

Try replacing…	with…
"The environment's down. We're stuck. We can't test."	"What can we test, review, or analyze now… and *are you OK with this situation*, dear client?"
"They didn't give us good requirements documents!"	"Let's write down what *we* know—and then they'll tell us when they think it's wrong."
"It's too hard to test this!"	"What can we do in the product and the project to things more testable?"
"We don' t have enough time to test!"	"What testing shall we do—what shall we cover—in the time we *do* have?"
"We have to…!"	"We choose to…"

Testers as Their Own Best Friends

The good news is that if we study testing,
talk to each other about testing,
practice doing testing,
practice talking about testing,

we can create a more powerful and helpful craft…

Testers as Their Own Best Friends

…and help our clients and colleagues make wonderful,
useful, beautiful, helpful, *humane* things.