How to Get What You Really Want from Testing

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

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

Updates

- This presentation is ALWAYS under construction.
- I probably will skip over some slides.
- Slides for this presentation are available on request.
- All material comes with lifetime free technical support.

A Clarification

- "How To Get What You Really Want From Testing" is a potentially misleading title.
- As a member of the Context-Driven School of Software Testing, I maintain there are no *universally* best results available from testing, just as there are no best practices…
- …and, of course, I'm not a mind reader.
- In this presentation, I'll try to help introduce ideas intended to help you choose or assign or develop testing missions that line up with *what managers and other testing clients typically need to know, in my experience*.
- Only you can really decide what's best *for you*.

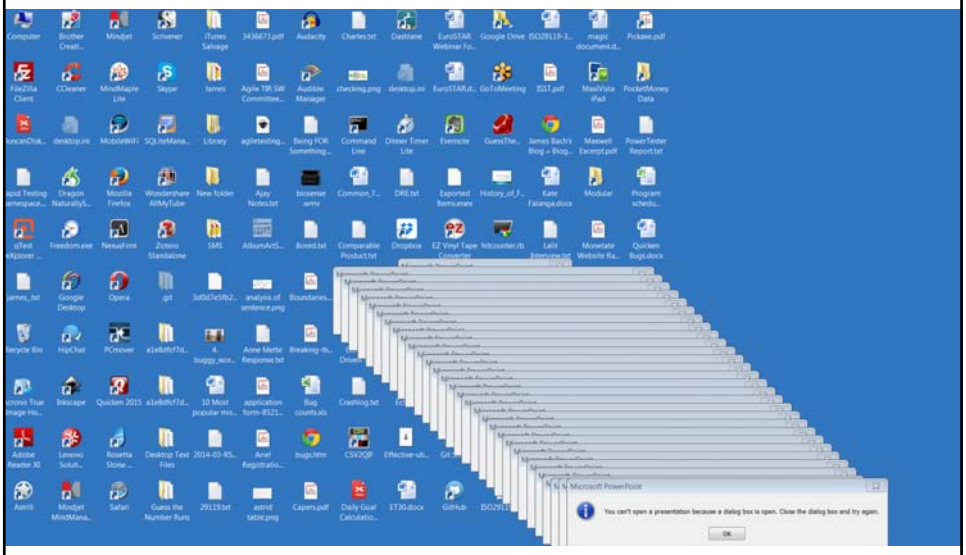
I'm Biased Towards Rapid Software Testing

Rapid Software Testing is a *mind-set*
and a *skill-set* of testing
focused on how to do testing
***more quickly*,**
***less expensively*,**
with *excellent results*.

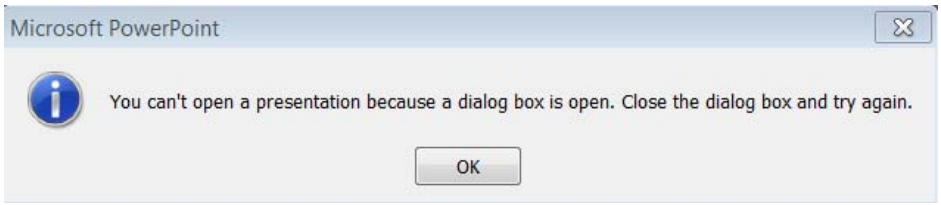
This is a general testing methodology. It adapts to any kind of project or product.

Read more about Rapid Software Testing at http://www.developsense.com

This morning…



So, what did that say?



I try to book a flight…

Flight Itinerary

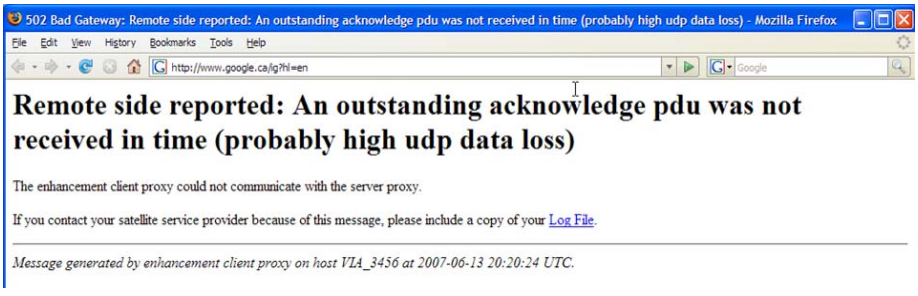
Flight	From	To	Stops	Aircraft	Fa
AC597	**Toronto, Pearson Int'l (YYZ)** Sat 16-Oct 2010 13:10 - Terminal 1	**Las Vegas, Mccarran Int'l (LAS)** Sat 16-Oct 2010 14:54 - Terminal 2	1	319	Ta
	AC597: This flight includes a stop in null.				
AC5233*	**Las Vegas, Mccarran Int'l (LAS)** Tue 19-Oct 2010 19:15 - Terminal 1	**San Francisco, San Francisco Int'l (SFO)** Tue 19-Oct 2010	0	320	Ta

aircanada.com

This makes me nervous!

aircanada.com

This makes me confused!



Error Message
Via Rail, between Montreal and Toronto, 2007

This makes me frustrated!

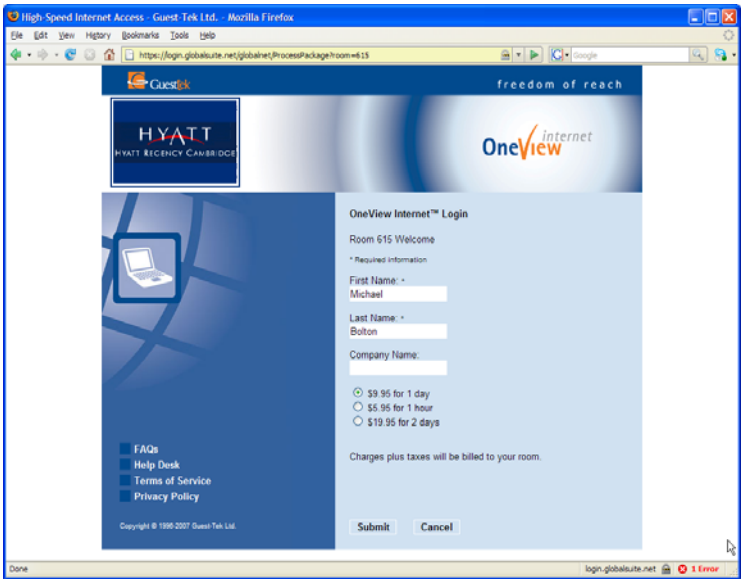
Remote side reported: An outst
received in time (probably high

The enhancement client proxy could not communicate with the server pr

If you contact your satellite service provider because of this message, pl

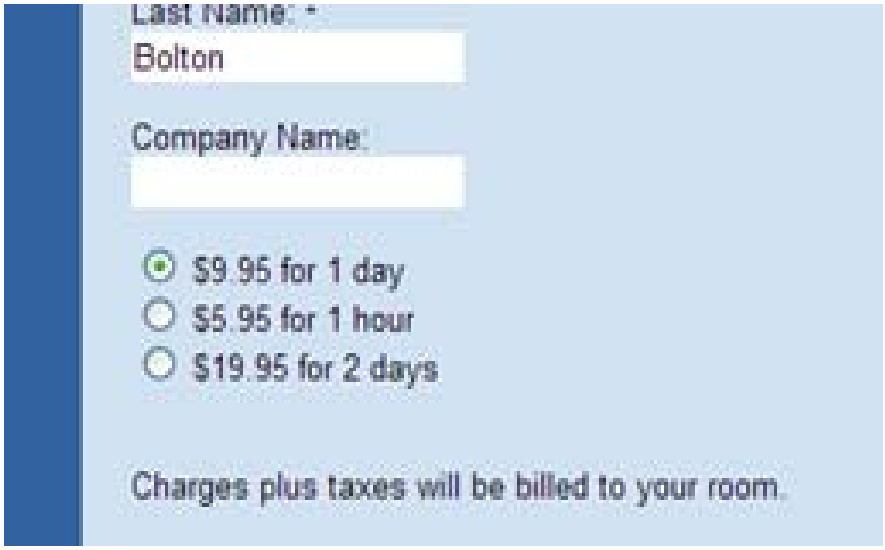
Message generated by enhancement client proxy on host VIA_3456

Error Message
Via Rail, between Montreal and Toronto, 2007



Hyatt Regency, 2008

This makes me laugh!

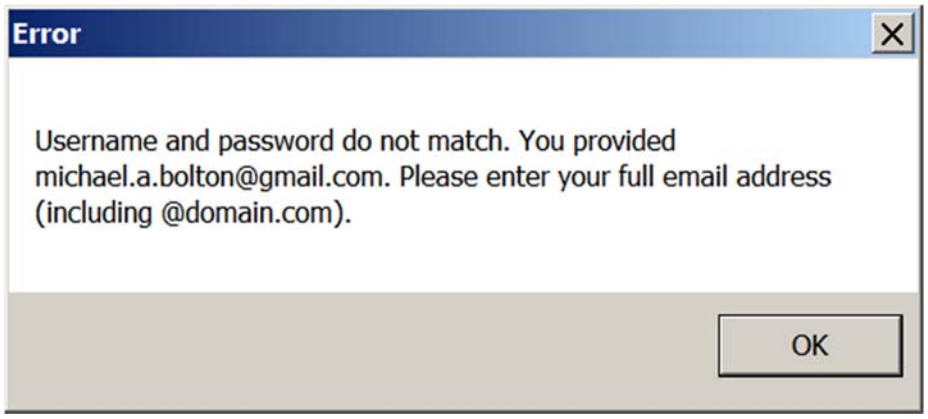


Hyatt Regency, 2008

I'm frustrated, and this message doesn't help!

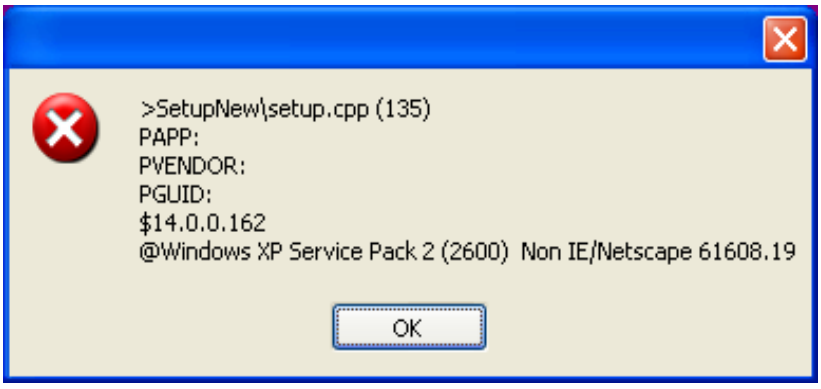
Google Chrome

This makes me confused!



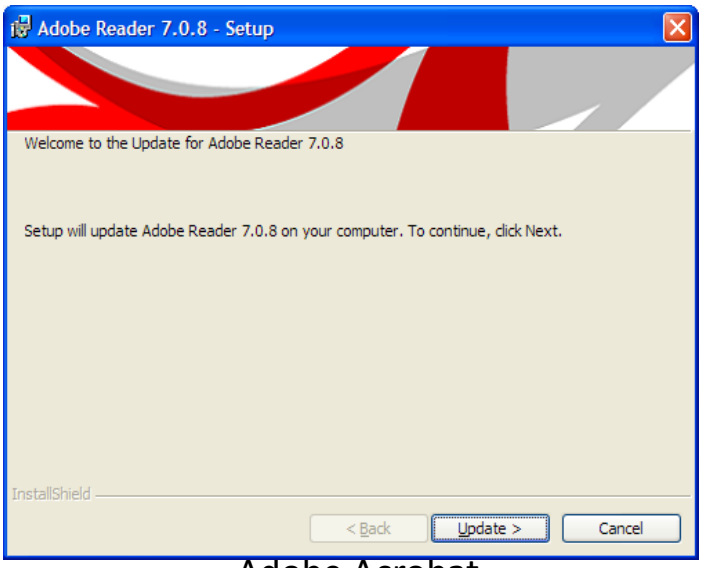
Google Calendar Update

This makes me confused!



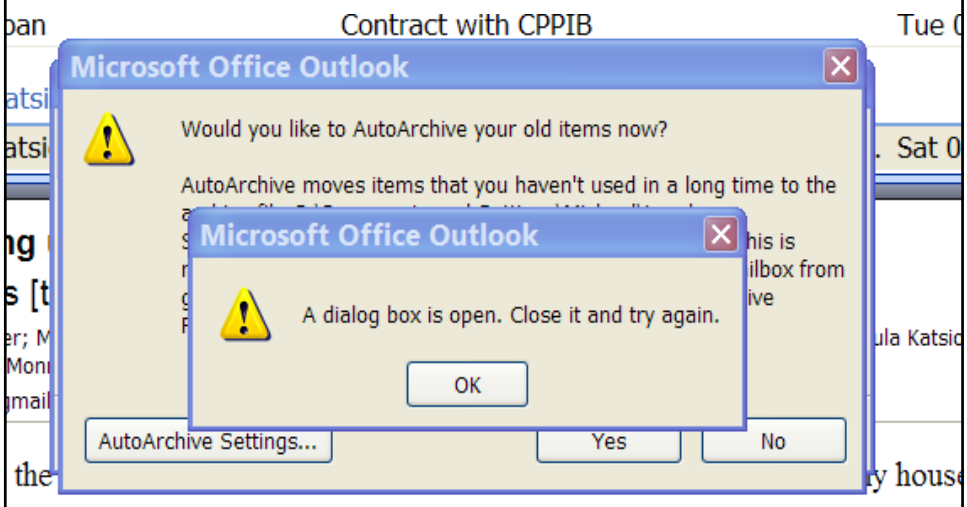
Don’t Know Who This Was

This makes me think the tester wasn't paying attention!



Adobe Acrobat

This makes me confused!



Microsoft Outlook

How Do People React to Software?

Impatience

Frustration

Amusement

Surprise

Confusion

Annoyance

What Might Feelings Tell Us?

Impatience	⇨ a threat to performance?
Frustration	⇨ a threat to capability?
Fear	⇨ a threat to security?
Surprise	⇨ a threat to reliability?
Confusion	⇨ a threat to usability? to testability?
Annoyance	⇨ a threat to charisma?
Boredom	⇨ an insignificant test?
Tiredness	⇨ time for a break?
Anxiety	⇨ a need for a particular skill?
Curiosity	⇨ a pointer to useful investigation?

Bugs!

Issues!

Some people have strange ideas about software development.

They think of THIS as "efficiency"...

image credit: istockphoto.com

But they forget that
that the product was DESIGNED.

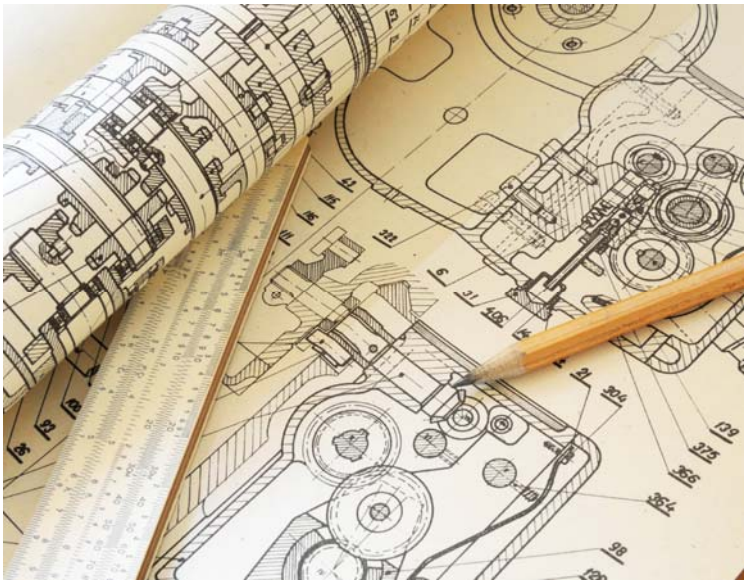


image credit: istockphoto.com

And they forget that
the FACTORY had to be built.

image credit: istockphoto.com

AND they forget that
the factory had to be DESIGNED.

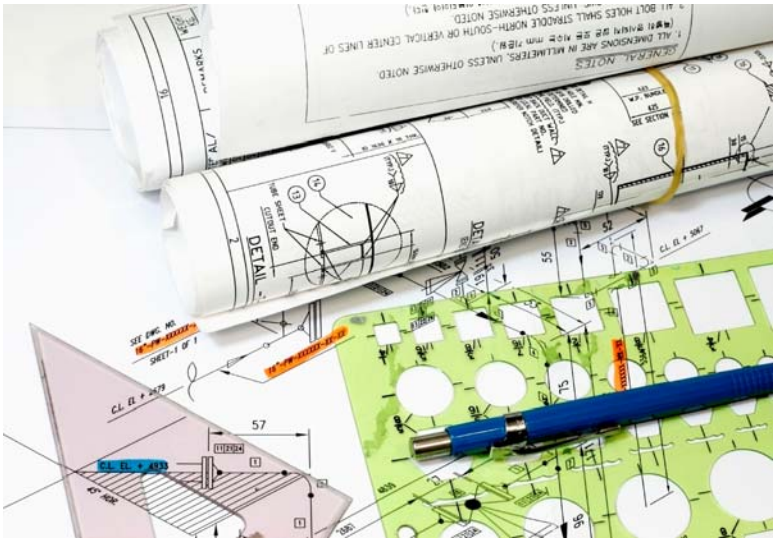


image credit: istockphoto.com

Software development is NOT factory work.

Software development is the part where we
DESIGN the thing that we're going to copy.

The people with the weird ideas
forget that designs must be DEVELOPED.

image credit: istockphoto.com

They forget that designs
must be INTERPRETED.

image credit: istockphoto.com

They forget that putting even well-designed things together is often MESSY.

image credit: istockphoto.com

They forget that even building MODELS can be messy.

image credit: istockphoto.com

All they seem to know about building models is what they can see of the finished product.

image credit: istockphoto.com

They forget what happens when you try to complete too much work too fast.

They forget what happens
when you try to drive too *quickly*
without driving *carefully*.

We develop skill and tacit knowledge in three ways.
Only one way is through explanations,
and that may be the weakest way.

image credit: istockphoto.com

We also learn from being immersed in a culture where people are doing the same kinds of things. Many process models fail to recognize this.

image credit: istockphoto.com

We learn far more, and more quickly, from our own practice, mistakes, and feedback. Many process models fail to recognize feedback loops.

image credit: istockphoto.com

People forget that even though we would prefer not to make mistakes, mistakes are normal when we're learning to do difficult things.

image credit: istockphoto.com

They forget that problems happen even when we're NOT trying to learn difficult things.

image credit: istockphoto.com

They become fixated on
numbers and charts and dashboards.

They forget that when you're driving,
it's important to look out the window too.

image credit: istockphoto.com

They stop believing that you can recognize significant, important things without numbers.

They forget that numbers are *illustrations*: extra data, not information.

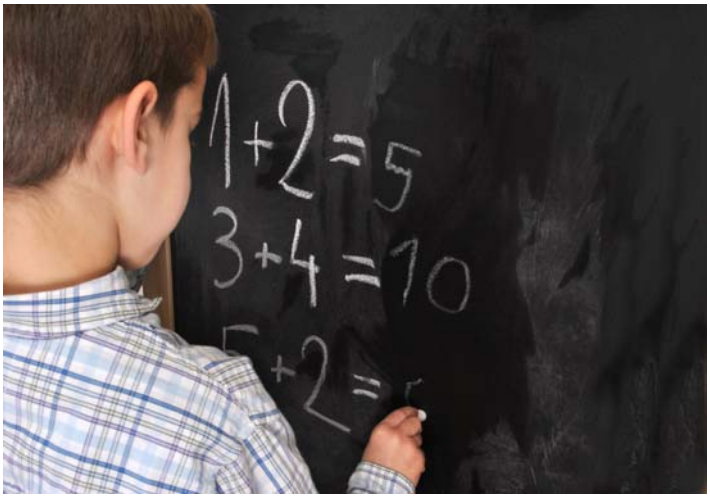


image credit: istockphoto.com

They decide that test cases are a good idea, because you can COUNT test cases.

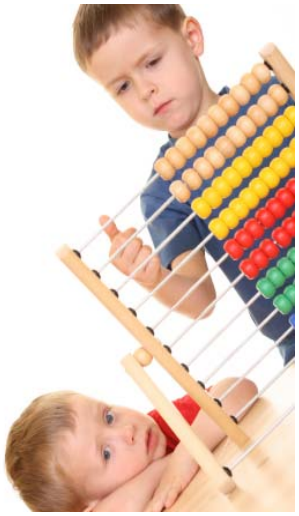


image credit: istockphoto.com

Do pilots use "piloting cases"?

image credit: istockphoto.com

Do parents use "parenting cases"?

image credit: istockphoto.com

Do scientists use "science cases"?

image credit: istockphoto.com

Do managers use "management cases"?

image credit: istockphoto.com

Then why do testers use test cases?

One Answer

- Managers like test cases because they make testing "legible"—readable to people who don't understand testing
- Testers keep talking about test cases because managers keep asking about them.
  - because testers keep talking about them
    - because managers keep asking about them
      - because testers keep talking about them
        - because managers keep asking about them
          - because testers keep talking about them
            - because managers keep asking about them
              - because testers keep talking about them
                - because managers keep asking about them

But smart people can do MUCH better than that.

How to do better?

- Consider what it means to test
- Learn the difference between checking and testing (and why that's important)
- Learn that testing is a heuristic process
- Learn how to model
  - the project
  - the product
  - what people want from the product
  - test techniques
- Learn about coverage and oracles
- *Put the tester at the centre of the testing*

Heuristics

bring useful structure to problem-solving skill.

"a **fallible method** for **solving a problem** or **making a decision**"

- a heuristic is not a **rule**
- a heuristic **can work** but **might fail**

"The engineering method is
the use of heuristics
to cause the best change
in a poorly understood situation
within the available resources."

Billy Vaughan Koen
Discussion of the Method

19-Heuristics.pdf - 2

Call this "Checking" not Testing

operating a product to check specific facts about it…

means

Observe → Evaluate → Report

Interact with the product in specific ways to collect specific observations.

Apply algorithmic decision rules to those observations.

Report any failed checks.

Three Parts to a Check

1. An *observation* linked to…
2. A *decision rule* such that…
3. they can be both be applied with algorithms—by a **program**.

That means a ***check*** can be performed

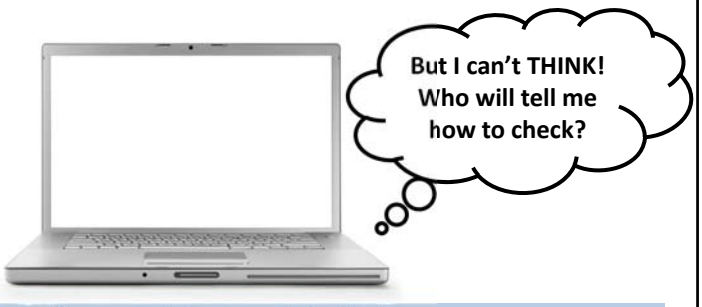
by a machine
that *can't* think
(but that is quick and precise)

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

image credit: istockphoto.com

Testing Is *More* Than Checking

- *Checking* is about confirming and verifying things we *think* are true or *hope* are true
  - But a program can have MANY problems, even when what we've *checked* seems correct
  - Checking *can* and *probably should* be done by machines



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

Harry Collins on Software Testing

"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.

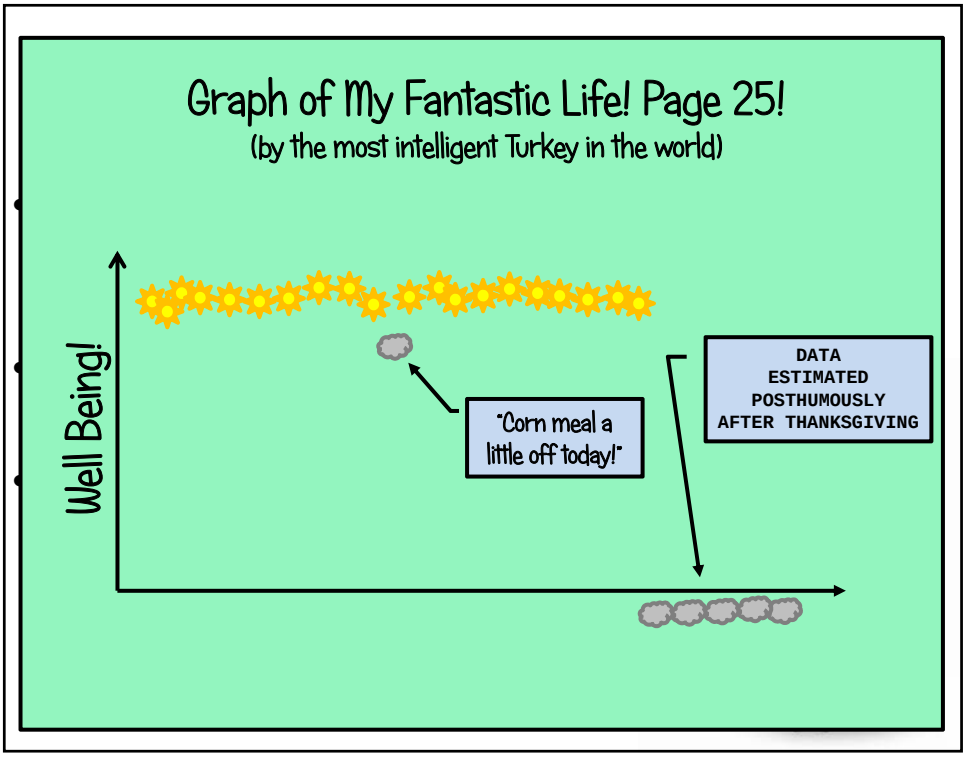
Machines are also 'social prostheses', fitting into social life where a human once fitted. It is a characteristic of medical prostheses, like replacement hearts, that they do not do exactly the same job as the thing they replace; the surrounding body compensates.

Abstract, "Machines as Social Prostheses", EuroSTAR 2013

Harry Collins on Software Testing

"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."

Abstract, "Machines as Social Prostheses", EuroSTAR 2013

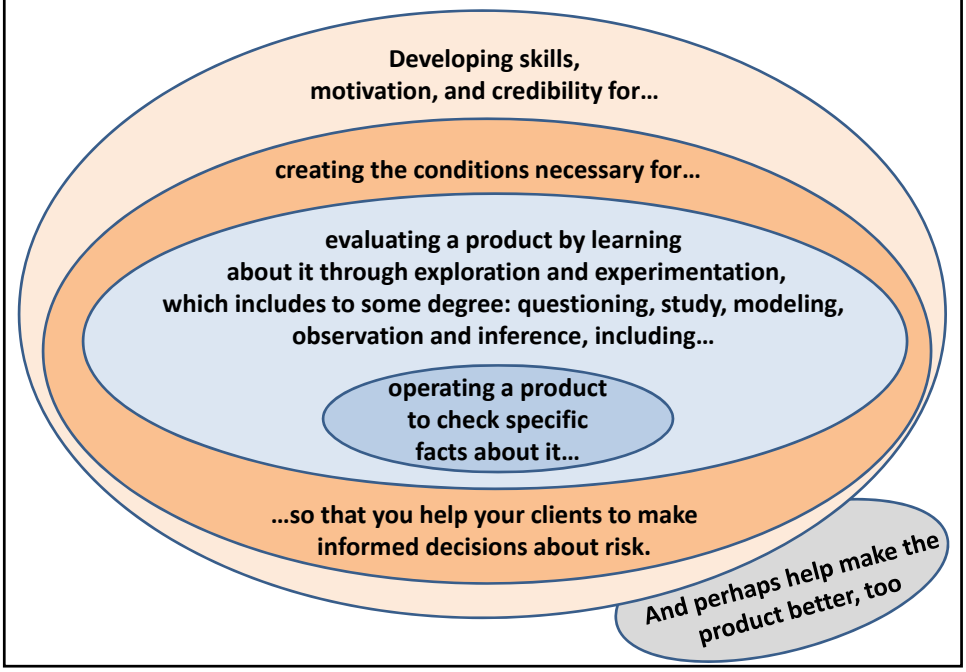


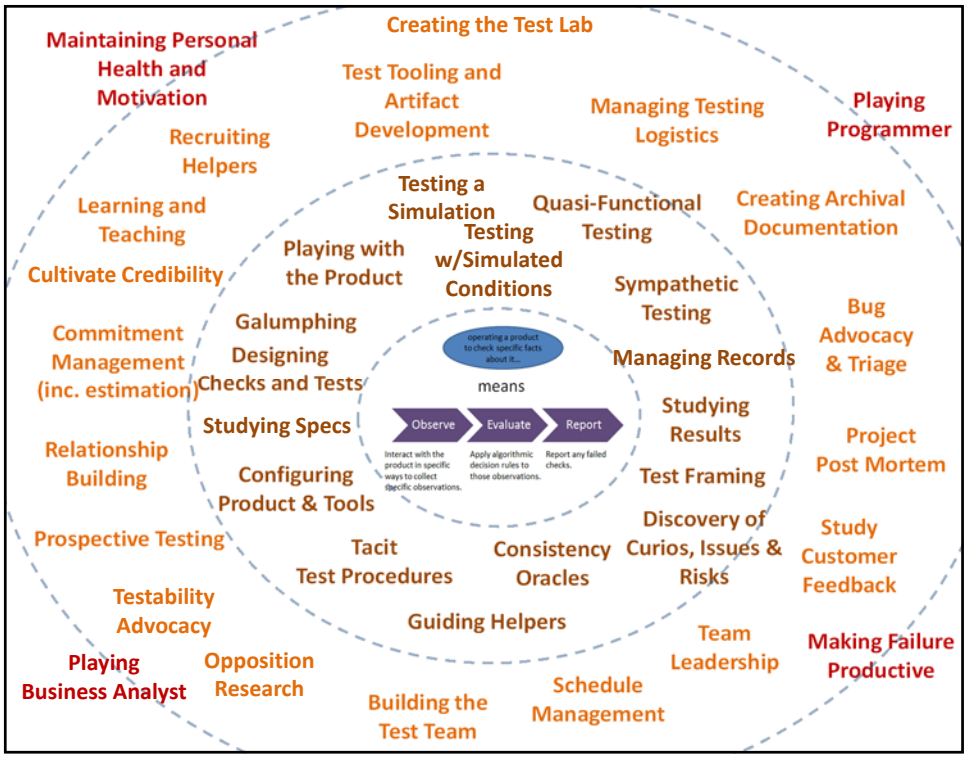
Don't Be A Turkey!

- No experience of the past can LOGICALLY be projected into the future, because we have no experience OF the future.
- This is no big deal in a world of stable, simple patterns.
- **BUT NEITHER SOFTWARE NOR PROJECTS ARE STABLE OR SIMPLE.**
- **"PASSING" TESTS CANNOT PROVE SOFTWARE GOOD.**

Based on a story told by Nassim Taleb, who stole it from Bertrand Russell, who stole it from David Hume.

Testing is…





What Is Testing?

- Excellent testing is not merely a branch of computer science
  - testing *includes* computer science, mathematics, technical domains
  - BUT… focus only on programs and functions, and you leave out questions of *value* and other relationships that include people
- To me, excellent testing is more like *anthropology*—interdisciplinary, systems-focused, investigative, storytelling

Biology

Archaeology

Language

Culture

What is testing?

"Try it and see if it works."

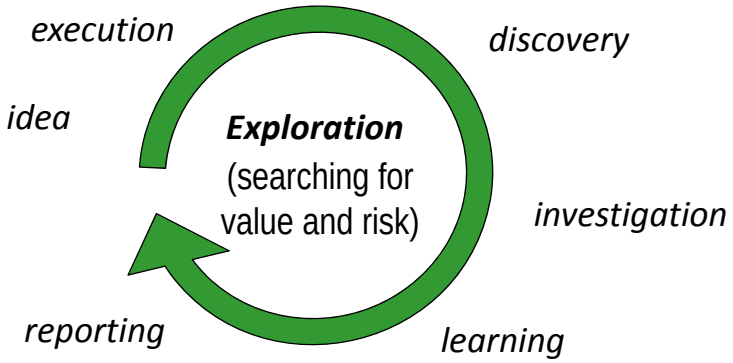
really means...

"Try it to **discover enough**,
about whether it **can work**,
and how it **might not work**,
to learn whether it will work."

63

Testing is about *Learning* and *Adaptation*

Skilled testers help to defend the value of the product by *learning* about it on behalf of our clients.



…and there are little loops between all of these things.

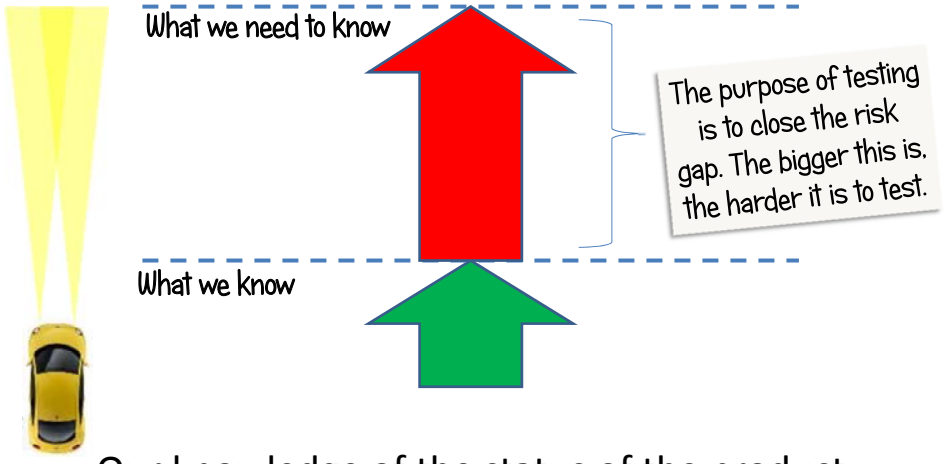
Confirmatory testing *feels* good,
but displaces discovery and learning, and hides risk.

Testers Extend Clients' Senses

Like scientific instruments, we testers extend our clients' senses to help raise awareness about product, projects and risks.

We also use our own tools to extend our clients' senses even farther.

Testers Light the Way: *The Risk Gap*



Our knowledge of the status of the product.

Important words!

- **Bug:** any problem that threatens the value of the product
- **Issue**: any problem that threatens the value of the testing, or the project, or the business
- **Coverage**: how much of the product you have tested based on *some model*
- **Oracle**: something that helps you to recognize a problem when it happens during testing.

Good testers ask "Is there a problem here?" and "Are we okay with this?"

Tell a Three-Part Testing Story

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.

A story about HOW YOU TESTED it...
...how you operated and observed it
...how you recognized problems...
...what you have and *have not* tested *yet*...
...what you won't test *at all* (unless the client objects)...

A story about how GOOD that testing was...
...the risks and costs of **t**esting or not testing...
...what made testing harder or slower...
...how testable (or not) the product is...
...what you need and what you recommend.

image credit: istockphoto.com

How [illegible] Test

- Write test cases?
- Play with the produ[illegible]
- Use the all-pairs test [illegible]
- Use equivalence class partitioning?
- Use record and playback automation?
- Read the spec?
- Report bugs? Help design the product?
- Write code for the product?
- Play ping pong in the break room?

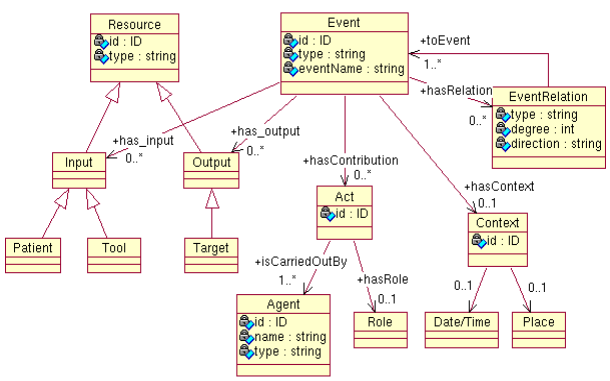
Yes, learn these things...
But there is no formula for DOING them.

Modeling

- How should I test?
- To test well, you must *learn to test*.
  For that, you need good models of testing.
- How should I test *my product*?
- To test a product you need a *diversified strategy*.
  For that, you need to model product-specific risks.
- How do I find important problems?
- To find problems, you must *cover the product*.
  For that, you need to model the product.
- How do I know when something is a problem?
- To recognize problems, you must *apply oracles*.
  For that, you need to learn about oracles.

Models

- A model is a simpler representation of a more complex idea, object, or system that helps you to understand, control, observe, or explore it.



Project Environment

MIDTESTD

- **Mission**
  - The problem we are here to solve for our customer.
- **Information**
  - Information about the product or project that is needed for testing.
- **Developer relations**
  - How you get along with the programmers.
- **Team**
  - Anyone who will perform or support testing.
- **Equipment & tools**
  - Hardware, software, or documents required to administer testing.
- **Schedule**
  - The sequence, duration, and synchronization of project events.
- **Test Items**
  - The product to be tested.
- **Deliverables**
  - The observable products of the test project.

"Ways to test…"?
General Test Techniques

FDSFSCURA

- Function testing – test each feature or function
- Domain testing—divide and conquer the data
- Stress testing—overwhelm or starve the product
- Flow testing—do one thing after another after another
- Scenario testing—test to a compelling story
- Claims testing—test based on what important people say
- User testing—involve (or systematically simulate) the users
- Risk testing—think of a problem, and then test for it
- Automatic checking—runs squintillions of programmed checks

What is Coverage?

_____ coverage is "how much testing we've done with respect to some model of _____"

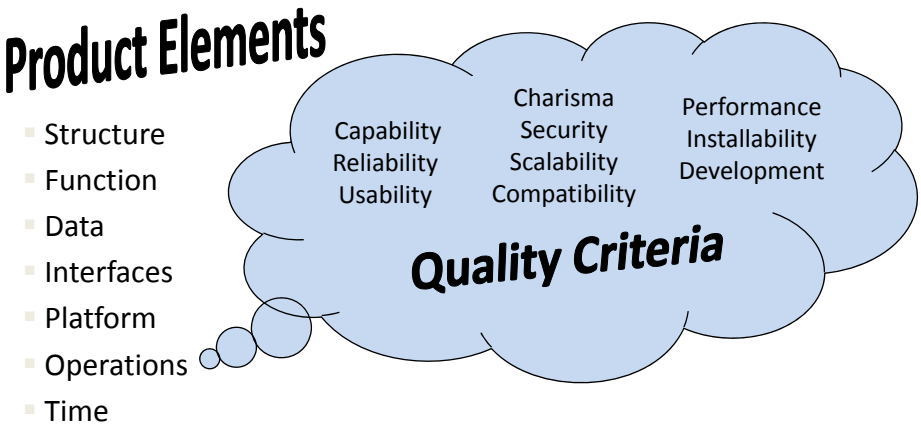
It's the extent to which we have traveled over *some map* of the product.

But what does it mean to "map" a product?
Talking about coverage means talking about

MODELS

One Way to Model Coverage: Product Elements (with Quality Criteria)

SFDIPOT – "San Francisco Depot"



Quality Criteria Where's the Value to People?

CRUCSSCPID

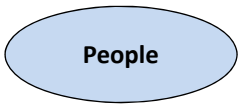
Capability	Scalability
Reliability	Compatibility
Usability	Performance
Charisma	Installability
Security	Development

Our job is to identify value, and threats to value. Many test approaches focus on capability (functionality) and underemphasize the other criteria.

General Examples of Oracles

things that suggest "problem" or "no problem"

- A process or tool by which the output is checked.
- A reference document with useful information.
- A known good example output.
- A known bad example output.
- A process or tool that helps a tester identify patterns.
- A person whose opinion matters.
- An opinion held by a person who matters.
- A disagreement among people who matter.



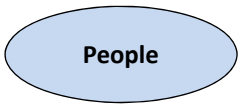
What Might Feelings Tell Us?

Impatience	⇨ a threat to performance?
Frustration	⇨ a threat to capability?
Fear	⇨ a threat to security?
Surprise	⇨ a threat to reliability?
Confusion	⇨ a threat to usability? to testability?
Annoyance	⇨ a threat to charisma?
Boredom	⇨ an insignificant test?
Tiredness	⇨ time for a break?
Anxiety	⇨ a need for a particular skill?
Curiosity	⇨ a pointer to useful investigation?

General Examples of Oracles

things that suggest "problem" or "no problem"

- A process or tool by which the output is checked.
- A reference document with useful information.
- A known good example output.
- A known bad example output.
- A process or tool that helps a tester identify patterns.
- A person whose opinion matters.
- An opinion held by a person who matters.
- A disagreement among people who matter.
- A feeling like confusion or annoyance.



Consistency ("this agrees with that")

an important theme in oracle principles

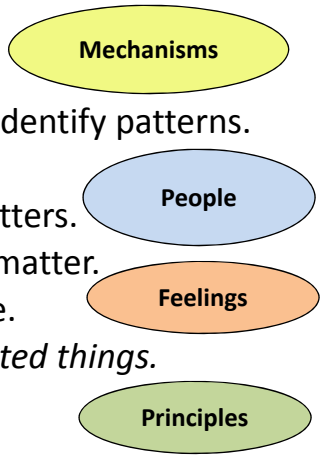
- **Familiarity:** The system ***is not consistent*** with the pattern of any familiar problem.
- **Explainability:** The system *is consistent* with our ability to describe it clearly.
- **World:** The system *is consistent* with things that we recognize in the world.
- **History:** The present version of the system *is consistent* with past versions of it.
- **Image:** The system *is consistent* with an image that the organization wants to project.
- **Comparable Products:** The system *is consistent* with comparable systems.
- **Claims:** The system *is consistent* with what important people say it's supposed to be.
- **Users' Desires** The system *is consistent* with what users want.
- **Product:** Each element of the system is *consistent* with comparable elements in the same system.
- **Purpose:** The system *is consistent* with its purposes, both explicit and implicit.
- **Standards and Statutes:** The system *is consistent* with applicable laws, or relevant implicit or explicit standards.

Consistency heuristics rely on the quality of your models of the product and its context.

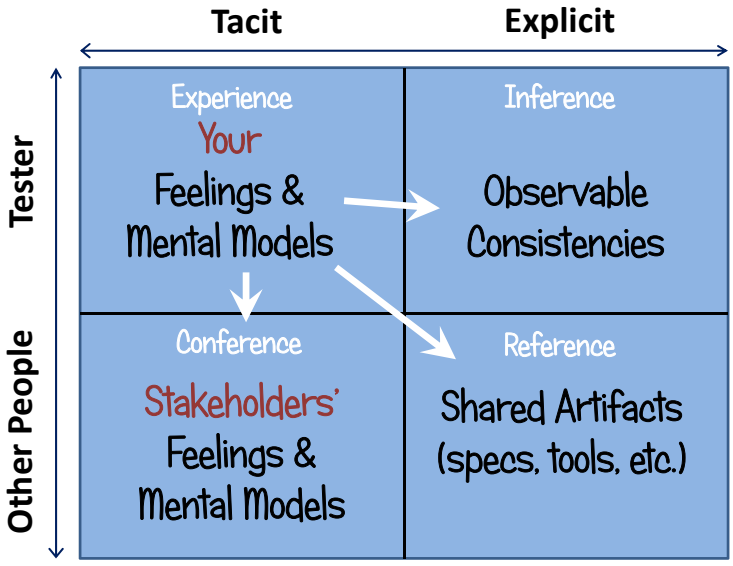
General Examples of Oracles

things that suggest "problem" or "no problem"

- A process or tool by which the output is checked.
- A reference document with useful information.
- A known good example output.
- A known bad example output.
- A process or tool that helps a tester identify patterns.
- A person whose opinion matters.
- An opinion held by a person who matters.
- A disagreement among people who matter.
- A feeling like confusion or annoyance.
- *A desirable consistency between related things.*



Oracles from the Inside Out



Oracles are Not Perfect And Testers are Not Judges

- You don't need to know FOR SURE if something is a bug; it's not your job to DECIDE if something is a bug.
- You do need to form a justified belief that it MIGHT be a threat to product value in the opinion of someone who matters.
- And you must be able to say why you think so; you must be able to cite good oracles… or else you will lose credibility.
- When testers have a good set of oracles, they become far more capable of discovering problems without depending on test cases or requirements documents.

Some Quality Criteria Are Oriented Towards Product Development

Remember Testability!

Supportability
Testability
Maintainability
Portability
Localization

Testability affords us opportunities for observing and controlling the product. Reduced testability gives bugs more time and more opportunities to hide.

Testability: Observability & Controllability

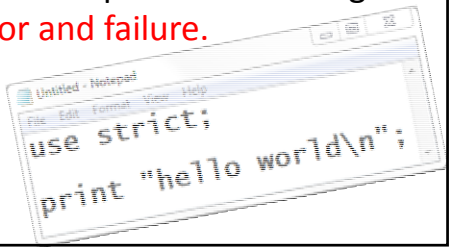
In order to test a product well, I must be able to control the execution to visit each important state that it has, see everything important, and control the variables (in the environment) that might influence it.

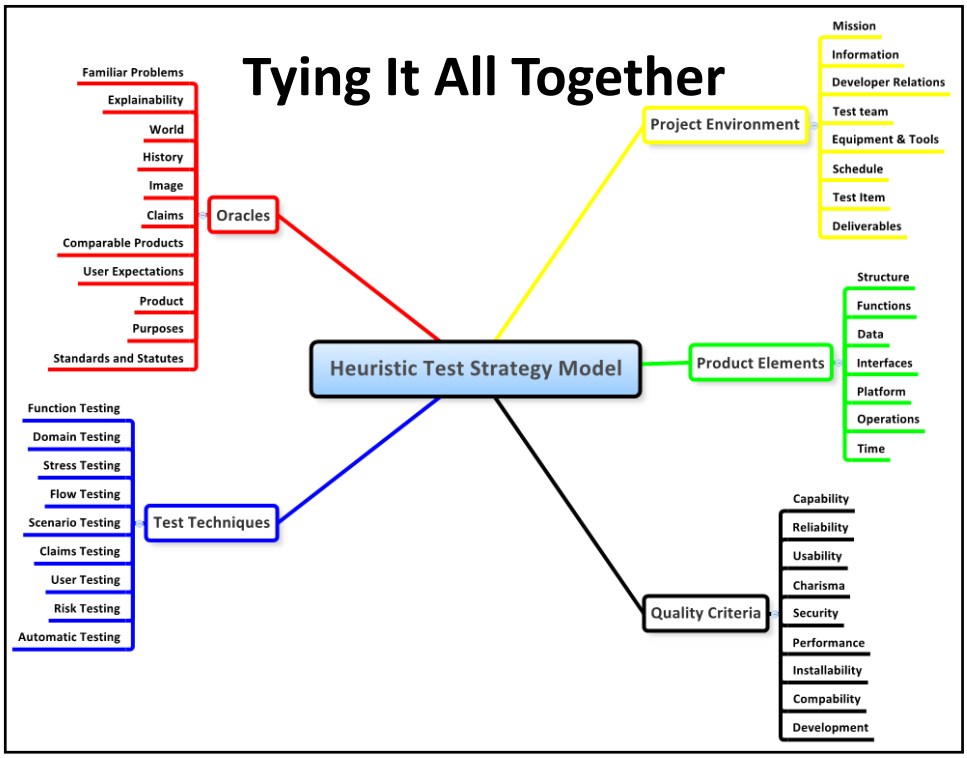
Imagine a clock under a glass shield. You are not allowed to come near it or poke or probe it...

Testability: Smallness and Simplicity

- A product is smaller, in testing terms, when there are fewer interestingly different and risky things that must be looked at.
- It is algorithmically simpler when there are fewer variations of those things to look at, and fewer conditions to control or consider when looking at them.
- This is related to your models of the product and usage as well as your theories of error and failure.

Imagine Notepad vs. Activestate Perl. Both can produce essentially infinite outputs, but one is far simpler and smaller than the other.





Core Ideas of Agile Development

- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."
  - so, keep refocusing on **increasing value**
  - and, in the testing role, refocus on *threats* to value
- "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."
  - so, develop and test in ways that **reduce the cost** of responding to constant change
  - and, in the testing role, provide relevant feedback about the product as rapidly as possible, to anticipate risks and understand the effects of change

HOW do we do “Agile Development”?

- Discover something worth building
- Build some of it
- Build it with change in mind
- Study what we’ve built
- …and loop!

Agile Development Cycle

(which has testing all through it)

This suggests a clockwise cycle, but…

Development isn't linear!

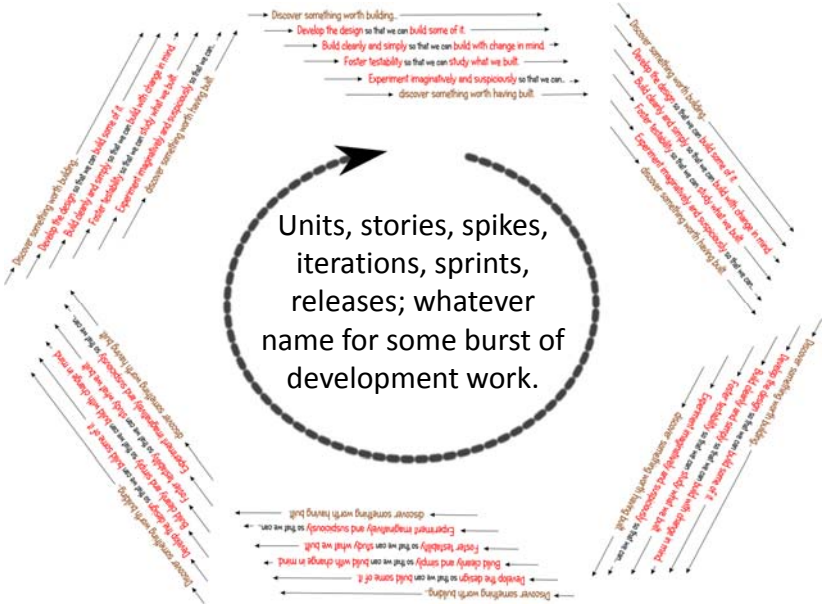
→ Discover something worth building... →
→ Develop the design so that we can build some of it. →
→ Build cleanly and simply so that we can build with change in mind. →
→ Foster testability so that we can study what we built. →
→ Experiment imaginatively and suspiciously so that we can... →
→ discover something worth having built. →

Although there is a cyclic tendency to these activities, they also overlap, combine, and support each other, in big loops, small loops, sudden turns, and epicycles.

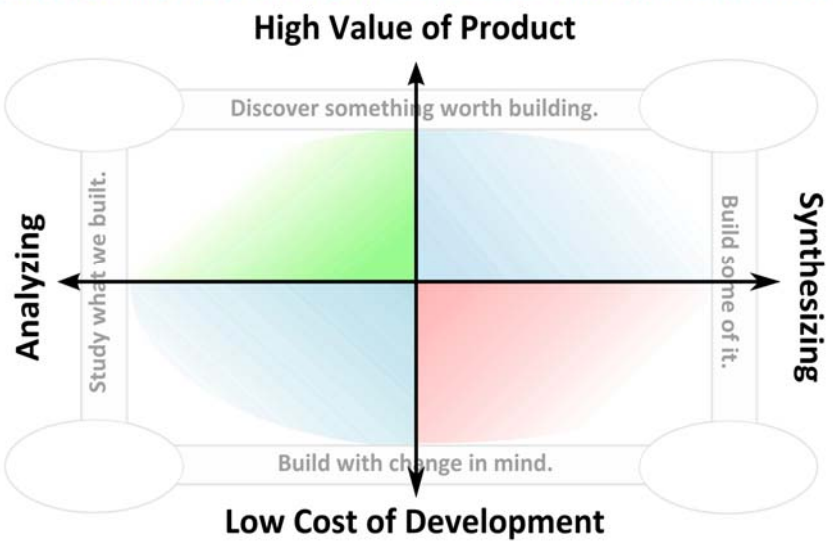
Like swirls from stirring a cup

…not like being tied to the hands

Development is a fractal!



ır highest priority is to satisfy the customer through…valuable softw



"Continuous attention to technical excellence and good design enhances agility."

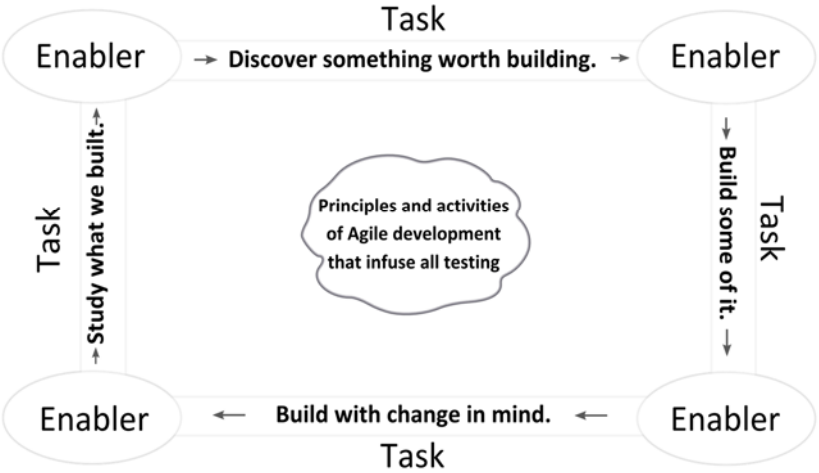
http://agilemanifesto.org/principles.html

What does it mean to do "Agile Development"?

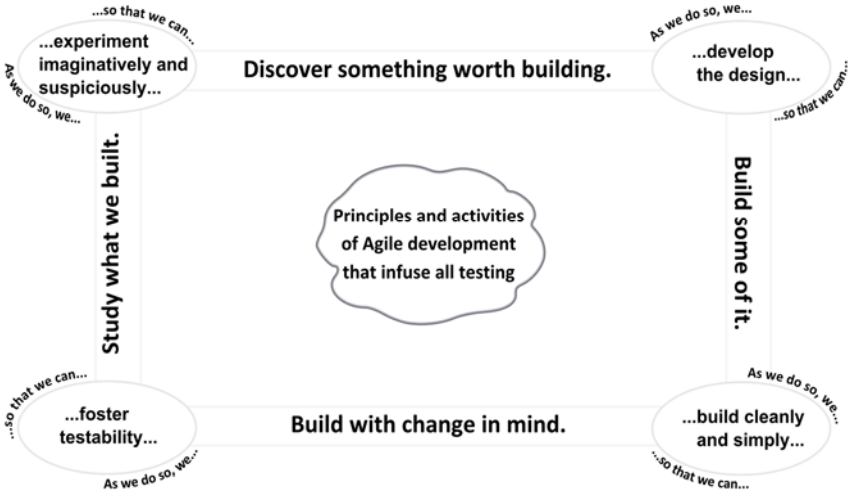
- Deliver often
  - short sprints help prevent us from getting ahead of ourselves
- Collaborate across roles
  - building a quality product is a goal we all share
- Develop craftsmanship
  - "being Agile" won't help us without study, practice and skill
- Don't be too formal!
  - individuals and interactions over processes and tools; working software over comprehensive documentation
- Apply appropriate heuristics
  - be prepared for and tolerant of occasional failures
- Develop and apply tools
  - tools are everywhere in software development and testing
- Seek a sustainable pace
  - people who are overwhelmed tend not to do good work

Agile Development
(which must include testing)

Tasks and Enablers



Tasks and Enablers

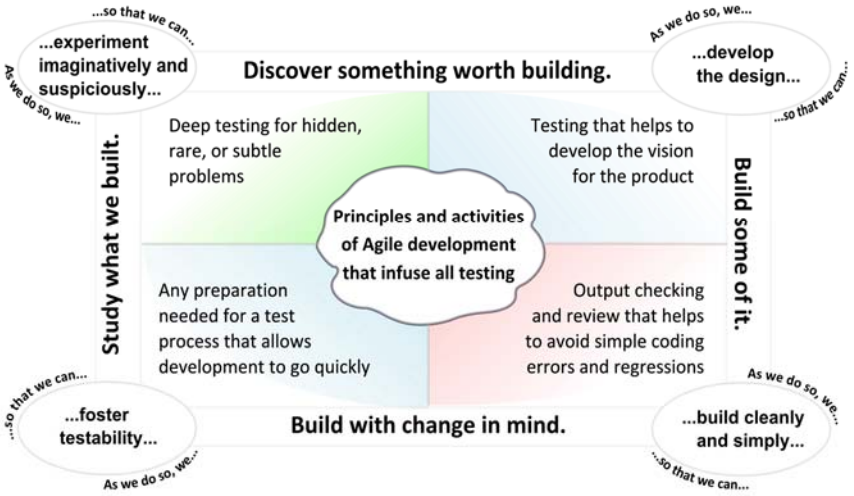


Testing in an Agile Context

- Testing to help develop the vision for the product and understanding of it
  - discussion, review, thought experiments…
- Testing to help refine the design and prevent low-level coding errors and regressions
  - TDD/BDD, unit checks, higher-level output checking…
- Preparation for testing to create a process that allows development to go more quickly
  - more testable products and projects; more tester skill*
- Deep testing for hidden, rare, or subtle problems
  - experimentation, investigation, exploratory scenarios, high volume/long sequence tool-assisted testing…

* See “Heuristics of Software Testability”, http://www.satisfice.com/tools/testable.pdf

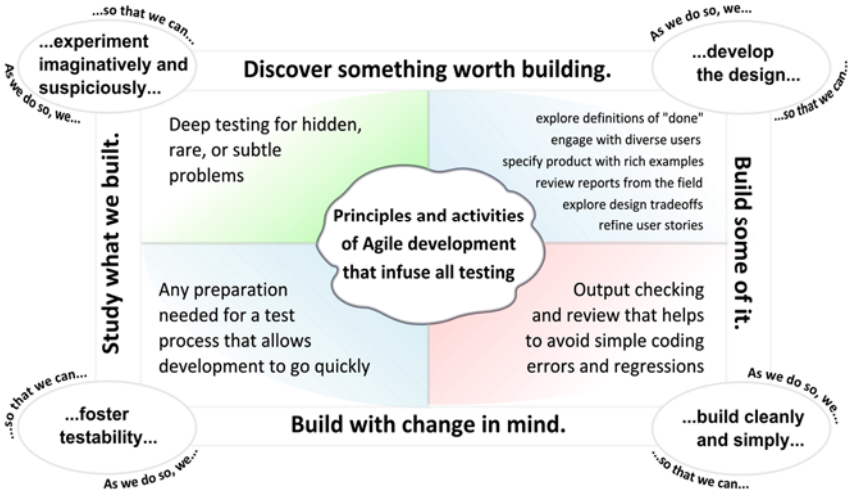
Testing in an Agile Context



Developing the Design

- Exploring definitions of done
  - we will change our definition of done when we learn more about the product and the project
- Work with diverse users
  - there are many different kinds of “user”
- Rich examples
  - examples ARE NOT tests, but examples DO help
- Review reports from the field
  - real-world problems from real users and support people
- Exploring design trade-offs
  - always think about cost, value, and risk
- Refining user stories
  - developing rich models of the product in its context

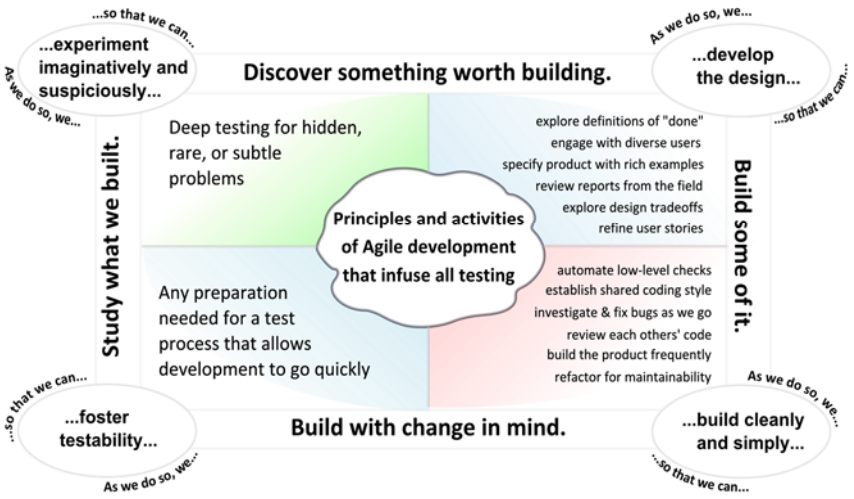
Developing the Design



Building Cleanly and Simply

- Automate low-level checks
  - producing a much more testable product
- Establishing shared coding style
  - make interpretation faster and easier
- Reviewing each other's code
  - destroy bugs before they even get into the house
- Building the product frequently
  - make constant feedback easy
- Re-factoring for maintainability
  - simplify the code because change will happen
- Investigating and fixing bugs as we go
  - destroy more bugs before they hide behind the drywall

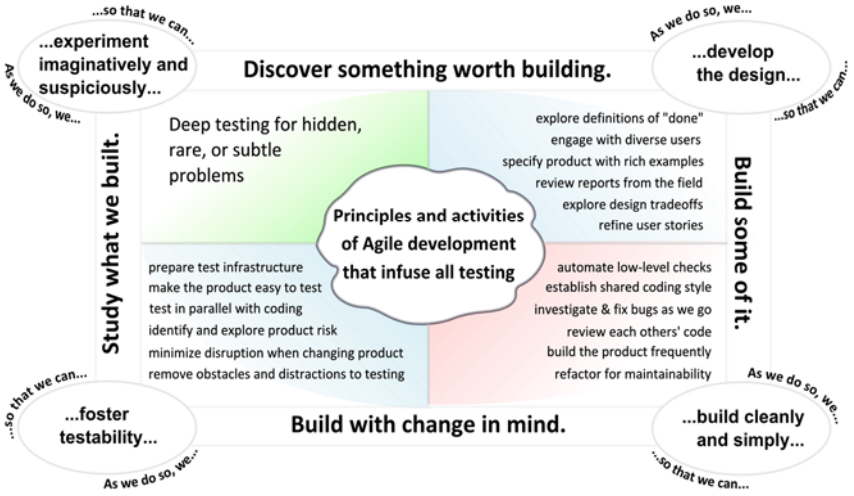
Building with Change in Mind



Fostering Testability

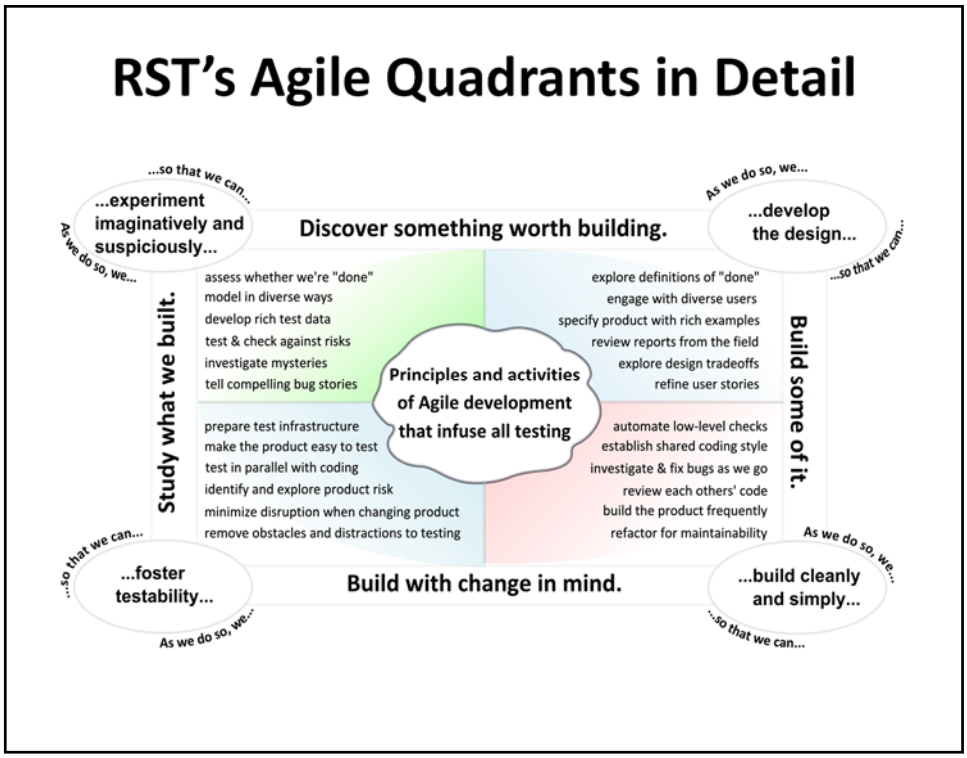
- Preparing test infrastructure
  - choose and develop tools and environments
- Making the product easy to test
  - ask developers for visibility and controllability
- Identifying and exploring product risk
  - digging up buried assumptions and problems
- Minimizing disruption when changing product
  - make testing a service, not an obstacle
- Removing obstacles and distractions to testing
  - address issues so testing can go quickly and smoothly
- Testing in parallel with coding
  - offer to test ANYTHING our clients want tested, at any time

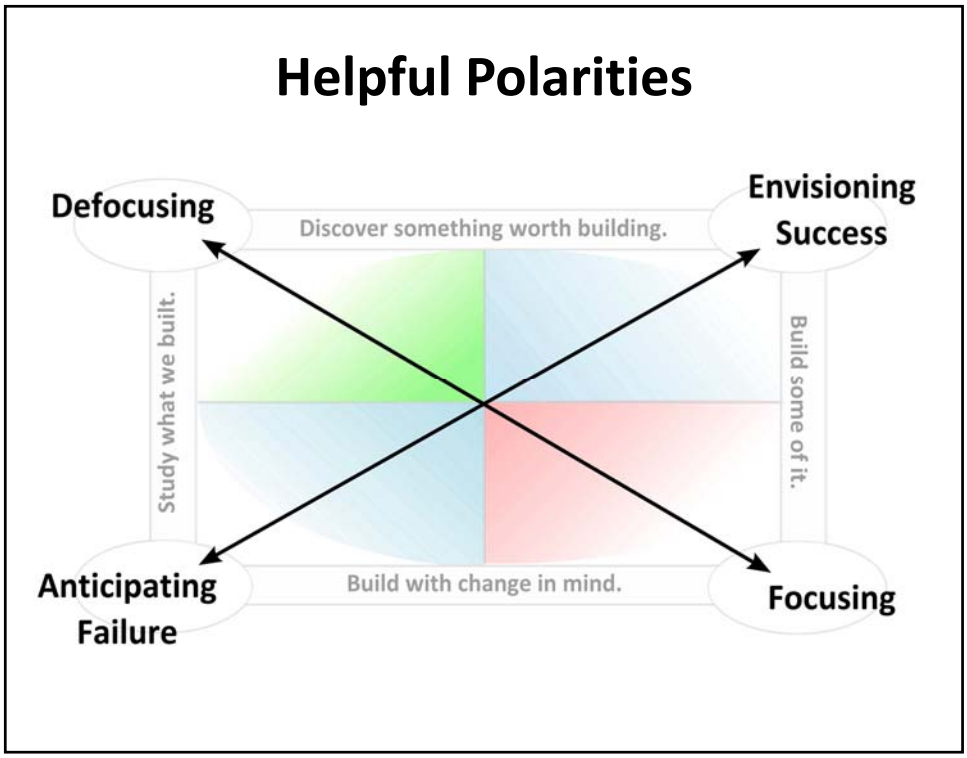
Fostering Testability



Experimenting Imaginatively and Suspiciously

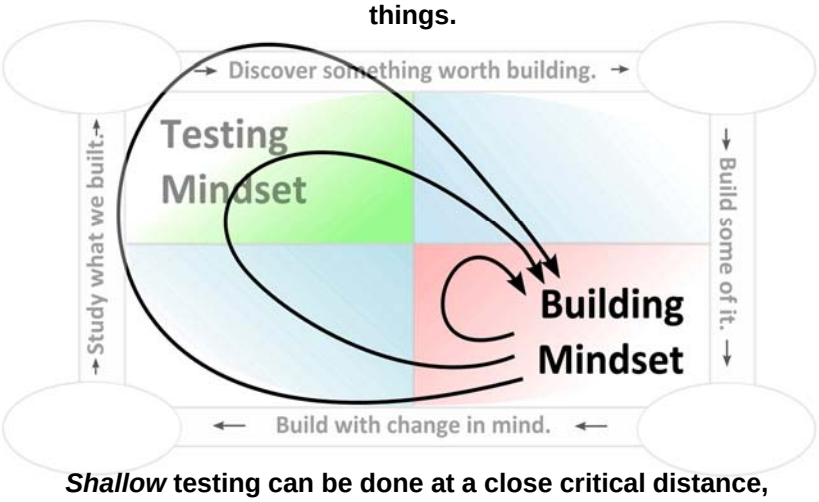
- Assessing whether we are done
  - think about may ways we *might not* be done *yet*
- Modelling in diverse ways
  - cover the product from many perspectives
- Developing rich test data
  - test for adaptability, not just repeatability
- Testing and checking against risks
  - focus testing on what matters
- Investigating mysteries
  - investigate hidden, subtle, or rare problems
- Telling compelling bug stories
  - provide context to show how bugs might threaten value





Critical Distance

"Critical Distance" refers to the differences in how we see things. Development and testing both benefit from different ways of seeing things.



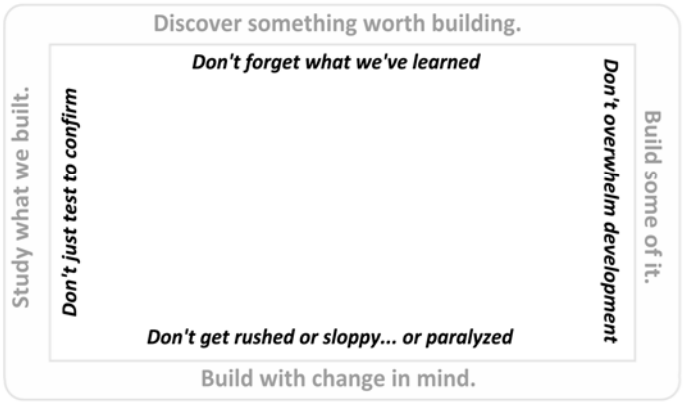
***Shallow* testing can be done at a close critical distance, Deeper or realistic or long-form testing needs or creates more distance from the builder's mindset.**

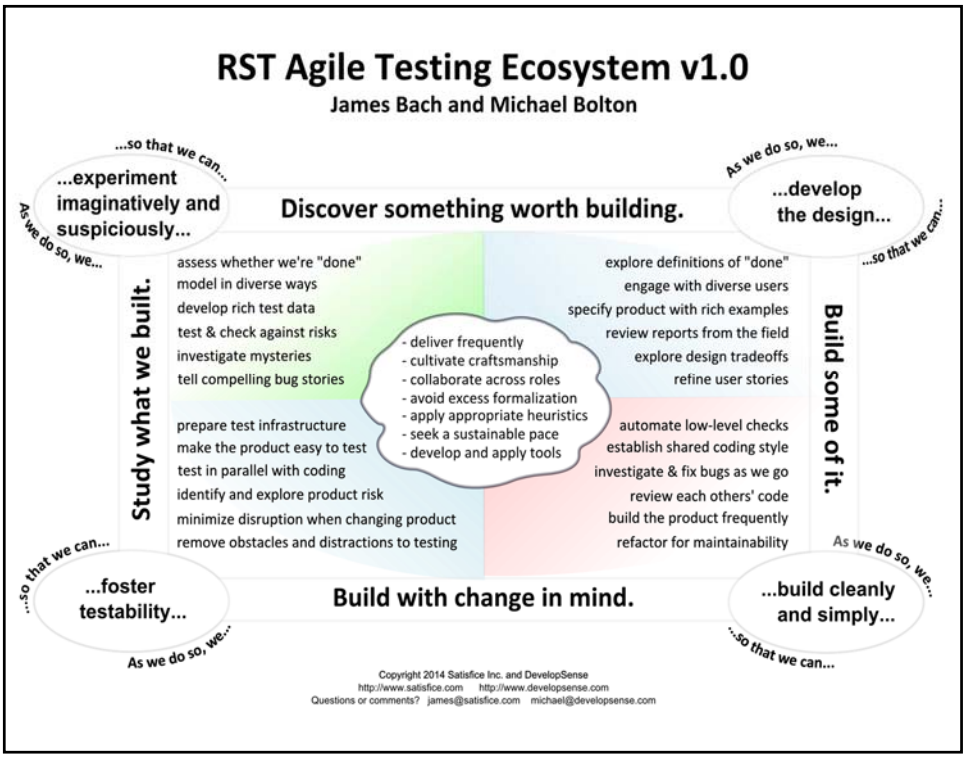
Trading Zone

Peter Galison uses the name "trading zone" for a situation in which people from different disciplines try to work together, even though they think in very different ways and use words differently.

On an Agile project, we need to set the team up to work in a mental and physical trading zone.

Some Caveats





Four Kinds of Risk Drivers

- problems in the *product*
- *unawareness* of problems in the product
- problems in the *project*
- *unawareness* of problems in the project

📖 *Risk (n.) Some **person** will suffer **annoyance, loss, or harm**, because of a **vulnerability** in a product or project that is triggered by some **threat**.*

Is Regression Your Biggest Risk?

- Before the Agile Manifesto was declared, a group of experienced test managers reported that regression problems ran from 6-15% of discovered problems
- In Agile shops, we now (supposedly) have
  - TDD
  - unit tests
  - pairing
  - configuration management
  - build and version control
  - continuous integration
- Is regression a serious risk?
- If so, can testing (whether automated or not) fix it?
- Is regression really a symptom of problems elsewhere?
- What about all the tests you *haven't* performed yet?

Regression Problems Are *Symptoms*

- If you see a consistent pattern of regression
  - the bugs you're seeing are probably not your biggest problem
  - your biggest problem might be *a favourable environment for regression*
  - is the project simply going too fast in a complex, volatile world?

Testers light the way.

This is our role.

We see things for what they are.
We make informed decisions about quality possible,
because we think critically about software.

So, how to get what you really want from testing?

Not-So-Good Questions for Testers

- Is the product done?
- Are we ready to ship?
- Is it good enough?
- How much time do you need to test?
- How many test cases are passing and failing?
- How many tests cases have you run?
- How many bugs are in the product?

Seek more than data.
Seek information.

Better Questions for Testers

- What is the product story? What can you tell me about important problems in the product?
- What risks I should be aware of?
- What have you done to obtain the product story?
- What important testing remains to be done?
- What problems are slowing testing down or making it harder to find out what we might need to know?
- What do you need to help speed things up?
- What *specific* aspects of testing are taking time?
- How do your tests link to the mission?

What might prevent the on-time, successful completion of the project?

What Interrupts Test Coverage?

- Non-testing work, setup of environments and tools, and bug investigation and reporting *take time away* from test design and execution
- Suppose testing that appropriately covers some aspect of a feature takes two minutes; let's call that a micro-session.
- Suppose also that it takes an extra eight minutes to investigate and report a bug
  - these are highly arbitrary and artificial assumptions—that is, they're *wrong*, but let's run the thought experiment anyway
- In a 90-minute session, we can run 45 micro-sessions *—as long as we don't find any bugs*

How Do We Spend Time?

(assume we're fabulous testers and would find everything our clients deem a bug)

Module	Bug reporting/investigation (time spent on tests that find bugs)	Test design and execution (time spent on tests that find no bugs)	Number of tests
A (good)	0 minutes (no bugs found)	90 minutes (45 tests)	45
B (okay)	10 minutes (1 bug, 1 test)	80 minutes (40 tests)	41
C (bad)	80 minutes (8 bugs, 8 tests)	10 minutes (5 tests)	13

Investigating and reporting bugs means….

SLOWER TESTING **or…**

REDUCED COVERAGE **…or both.**

- For Module A, our *coverage* is great—but if our clients assess us on the number of bugs we're finding, we look bad.
- For Module C, we look good because we're finding and reporting lots of *bugs*—but our *coverage* is suffering severely.
- Note that we haven't included setup time here, either.
- A system that rewards us or increases confidence based on the number of bugs we find might mislead us into believing that our product is well tested.

What Happens The Next Day?

(assume 6 minutes per bug fix verification)

	Fix verifications	Bug reporting and investigation today	Test design and execution today	New tests today	Total over two days
A	0 min	0 min (no new bugs)	90 min (45 tests)	45	90
B	6 min	10 min (1 new bug)	74 min (37 tests)	38	79
C	48 min	40 min (4 new bugs)	2 min (1 test)	5	18

Finding bugs today means….

VERIFYING FIXES LATER

…which means….

EVEN SLOWER TESTING **or…**

EVEN LESS COVERAGE **…or both.**

- …and note the optimistic assumption that all of our fixed verifications worked, and that we found no new bugs while running them. Has this ever happened for you?

122

13 Commitments for Testers to Make to Clients

1. I provide a service. You are an important client of that service. I am not satisfied unless you are satisfied.
2. I am not the gatekeeper of quality. I don't "own" quality. Shipping a good product is a goal shared by all of us.
3. I will test anything as soon as someone delivers it to me. I know that you need my test results quickly (especially for fixes and new features).
4. I will strive to test in a way that allows you to be fully productive. I will not be a bottleneck.
5. I'll make every reasonable effort to test, even if I have only partial information about the product.
6. I will learn the product quickly, and make use of that knowledge to test more cleverly.
7. I will test important things first, and try to find important problems. *(I will also report things you might consider unimportant, just in case they turn out to be important after all, but I will spend less time on those.)*

13 Commitments for Testers to Make to Clients

8. I will strive to test in the interests of everyone whose opinions matter, including you, so that you can make better decisions about the product.
9. I will write clear, concise, thoughtful, and respectful problem reports. *(I may make suggestions about design, but I will never presume to be the designer.)*
10. I will let you know how I'm testing, and invite your comments. And I will confer with you about little things you can do to make the product much easier to test.
11. I invite your special requests, such as if you need me to spot check something for you, help you document something, or run a special kind of test.
12. I will not carelessly waste your time. Or if I do, I will learn from that mistake.
13. I will not FAKE a test project.

The Themes of Rapid Testing

- Put the **tester's mind** at the center of testing.
- Learn to **deal with complexity** and ambiguity.
- Learn to **tell a compelling testing story.**
- Develop **testing skills** through practice, not just talk.
- **Use heuristics** to guide and structure your process.
- **Be a service** to the project community, not an obstacle.
- **Consider cost vs. value** in all your testing activity.
- **Diversify** your team and your tactics.
- Dynamically **manage the focus** of your work.
- Your **context should drive your choices**, both of which evolve over time.

And now, a word from our sponsor…

- We teach Rapid Software Testing, Critical Thinking for Testers, Rapid Software Testing for Programmers, and Rapid Software Testing for Managers.
- We provide consultancy services on testing and development.
- See http://www.developsense.com and http://www.satisfice.com for more information
- Contact us for questions or comments

How to Get What You Really Want from Testing

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