A Rapid Introduction to Rapid Software Testing

James Bach, Satisfice, Inc.
james@satisfice.com
www.satisfice.com
+1 (360) 440-1435

Michael Bolton, DevelopSense
michael@developsense.com
www.developsense.com
+1 (416) 656-5160

Acknowledgements

- **James Bach, Michael Bolton, Huib Schoots, Paul Holland,** and **Griffin Jones** co-create and teach this class.
- Some of this material was developed in collaboration with **Cem Kaner**.
- **Jon Bach** has been a long-term collaborator in developing ET management and training methods.
- **Lenore Bach** (wife of James) maintains the emotional and home environment that motivated James and enabled him to create this.
- **Mary Alton**, Michael's wife does the same for him, while also contributing support for our Web sites *and* trying to lead us towards better visual design in our materials. If what you see doesn't look good to you, it's probably because Mary wasn't involved.
- Many of the ideas in this presentation were also inspired by or augmented by other colleagues including Doug Hoffman, Bret Pettichord, Brian Marick, Dave Gelperin, Elisabeth Hendrickson, Jerry Weinberg, Noel Nyman, and Mary Alton.
- Some of our exercises were introduced to us by Payson Hall, Ross Collard, James Lyndsay, Dave Smith, Earl Everett, Brian Marick, and Joe McMahon.
- Many ideas were improved by students who took earlier versions of the class going back to 1995.

Assumptions About You

- You test software, or *any other complex human creation.*
- You have at least *some* control over the design of your tests and *some* time to create new tests.
- You are worried that your test process is spending too much time and resources on things that aren't important.
- **You test under uncertainty and time pressure.**
- **Your major goal is to find important problems quickly.**
- **You want to get *very good* at (software) testing.**

Test Something!

http://adam.goucher.ca/parkcalc/index.php

Thank you, Adam Goucher (who hosts this)

Matt Heusser (who discovered it)

and the anonymous intern (probably) (who coded it)

What Just Happened?

"We're making a product!"

"We need you to start testing it right now!"

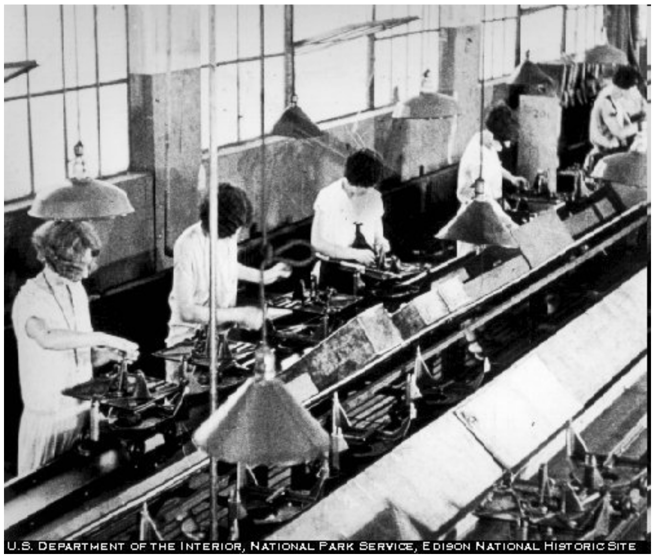
What do you do?

Testing in two easy steps!

1. Prepare test cases.
2. Execute test cases.

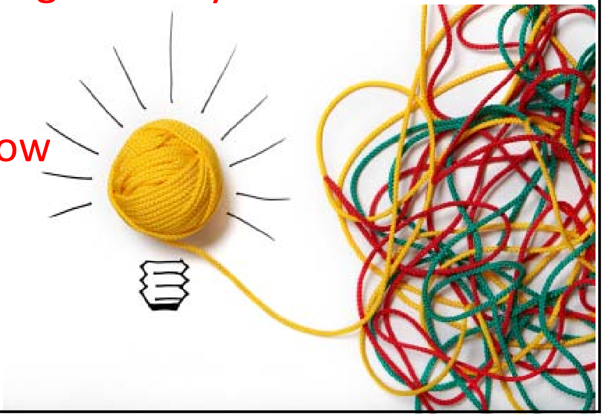
Maybe it's more like this…

1. Read the specification.
2. Identify specific items to be checked.
3. Prepare test cases.
4. Execute test cases.


U.S. Department of the Interior, National Park Service, Edison National Historic Site

Or maybe it's more like this…

1. Read the spec.
2. OMG there is no spec!
3. Oh wait, there is a spec! I'll just read it.
4. OMG the spec is old and confusing and maybe WRONG…
5. Maybe I should ask someone…
6. OMG Nobody seems to know how this thing is supposed to work!
7. Wait… is there something…. *anything* I can test?

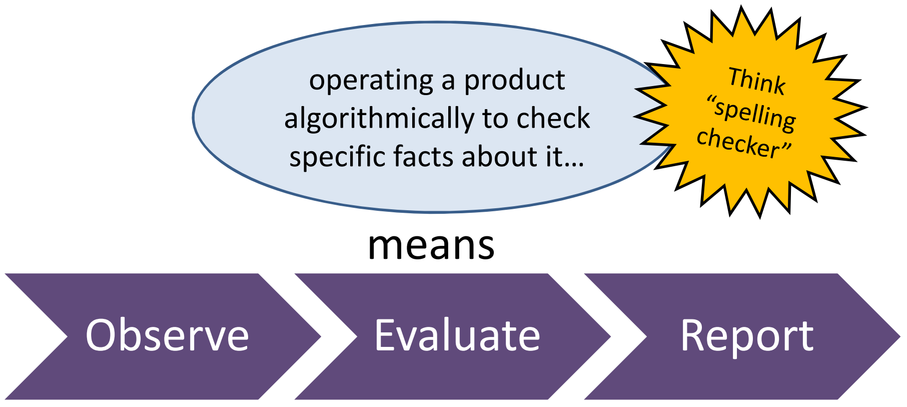


Yes! You CAN test…

- …the product
- …a mockup of the product
- …some document describing the product
- …a diagram that models the product
- …some feature of a work in progress
- …a product *like* this product
- …somebody's ideas about the product

Testing is the process of evaluating a product by learning about it through exploration and experimentation.

Call this "Checking" not Testing


operating a product algorithmically to check specific facts about it...
Think "spelling checker"
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)

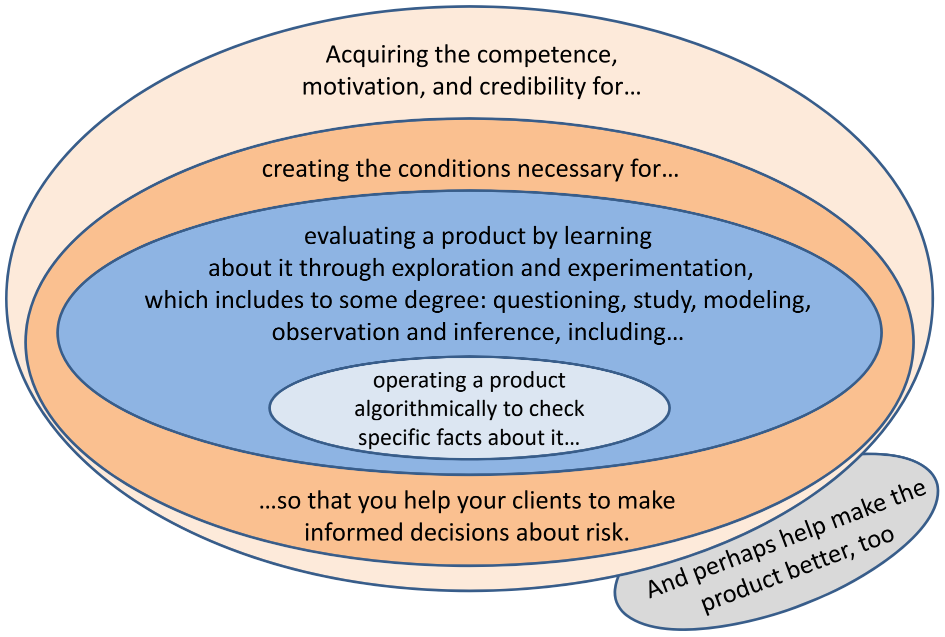
Notice that "quick" and "slow" refer only to the speed of observable behaviours and algorithmic evaluations.
The machine is *infinitely* slow at recognizing unanticipated trouble.

Testing Is *More Than Checking*

- *Checking* is okay, but it is mostly focused on confirming what we know or hope to be true.
- To escape the problems with verification, we must do more than checking; we must *test*.
- And… checking is always embedded in testing!

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

Testing is…



Acquiring the competence, motivation, and credibility for…

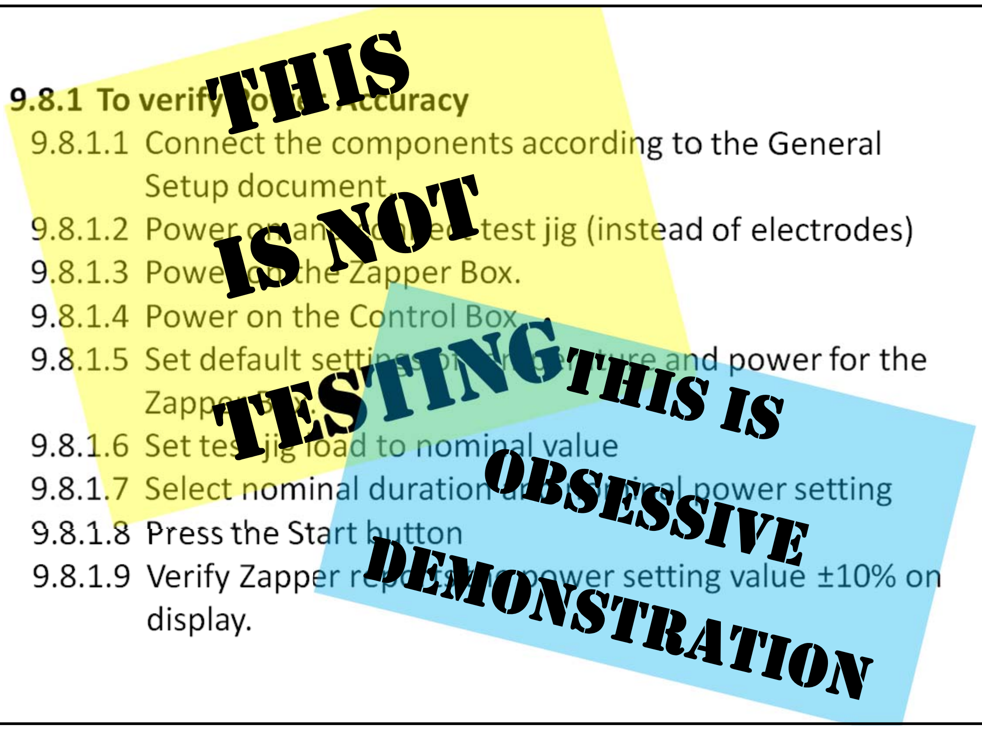
creating the conditions necessary for…

evaluating a product by learning about it through exploration and experimentation, which includes to some degree: questioning, study, modeling, observation and inference, including…

operating a product algorithmically to check specific facts about it…

…so that you help your clients to make informed decisions about risk.

And perhaps help make the product better, too


THIS IS NOT TESTING
THIS IS OBSESSIVE DEMONSTRATION

9.8.1 To verify Power Accuracy

9.8.1.1 Connect the components according to the General Setup document.
9.8.1.2 Power on and connect test jig (instead of electrodes)
9.8.1.3 Power on the Zapper Box.
9.8.1.4 Power on the Control Box.
9.8.1.5 Set default settings of temperature and power for the Zapper Box.
9.8.1.6 Set test jig load to nominal value
9.8.1.7 Select nominal duration and nominal power setting
9.8.1.8 Press the Start button
9.8.1.9 Verify Zapper reports the power setting value ±10% on display.

Uninteresting Testing Questions

- Does this test case pass or fail?
- How many test cases do we have?
- What's our pass/fail ratio?
- How long do you need to test?
- When will the testing be done?

Two *Fundamental* Testing Questions

Is there a problem here?

Are we okay with this?

If you don’t answer these questions, people won’t trust you.
That’s when they start asking silly questions.

What do managers and developers *really* want from testers?

An answer to this question:
Are there problems
that threaten
the on-time successful
completion of the project?

We test to obtain answers to that question.

Rapid Testing

Rapid 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.

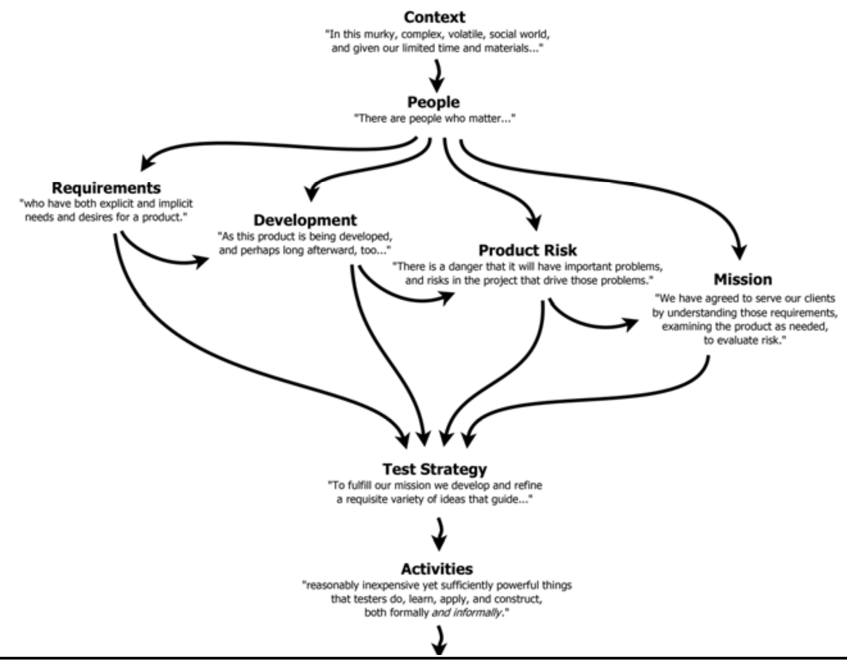
What is Rapid Software Testing?: Concise Answers

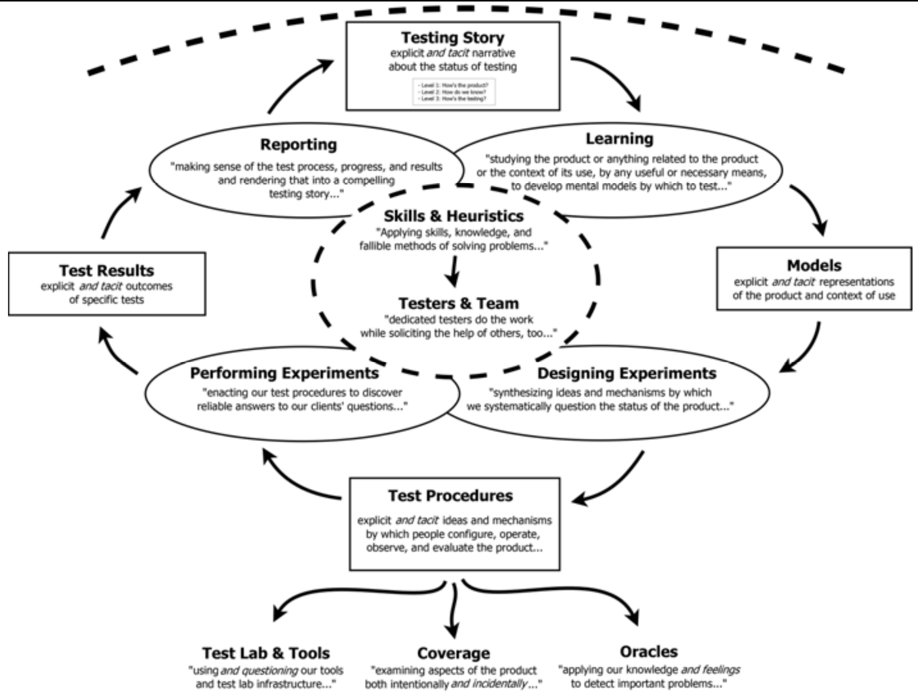
Rapid Software Testing is a methodology focused on how people learn and self-organize under pressure:

1. Learning to cope with complexity and uncertainty.
2. Learning how to design tests.
3. Learning how other people can help you test.
4. Learning how to use tools to amplify testing.
5. Learning what the product can be.
6. Learning what the bugs are in the product.
7. Learning to construct and share your testing story.

And how to deliver the fruits of that learning both effectively and ethically.

A Rapid Testing Framework


Context
"In this murky, complex, volatile, social world, and given our limited time and materials..."
People
"There are people who matter..."
Requirements
"who have both explicit and implicit needs and desires for a product."
Development
"As this product is being developed, and perhaps long afterward, too..."
Product Risk
"There is a danger that it will have important problems, and risks in the project that drive those problems."
Mission
"We have agreed to serve our clients by understanding those requirements, examining the product as needed, to evaluate risk."
Test Strategy
"To fulfill our mission we develop and refine a requisite variety of ideas that guide..."
Activities
"reasonably inexpensive yet sufficiently powerful things that testers do, learn, apply, and construct, both formally *and informally*."


Testing Story
explicit *and tacit* narrative about the status of testing
- Level 1: How's the product?
- Level 2: How do we know?
- Level 3: How's the testing?
Reporting
"making sense of the test process, progress, and results and rendering that into a compelling testing story..."
Learning
"studying the product or anything related to the product or the context of its use, by any useful or necessary means, to develop mental models by which to test..."
Skills & Heuristics
"Applying skills, knowledge, and fallible methods of solving problems..."
Testers & Team
"dedicated testers do the work while soliciting the help of others, too..."
Test Results
explicit *and tacit* outcomes of specific tests
Models
explicit *and tacit* representations of the product and context of use
Performing Experiments
"enacting our test procedures to discover reliable answers to our clients' questions..."
Designing Experiments
"synthesizing ideas and mechanisms by which we systematically question the status of the product..."
Test Procedures
explicit *and tacit* ideas and mechanisms by which people configure, operate, observe, and evaluate the product...
Test Lab & Tools
"using *and questioning* our tools and test lab infrastructure..."
Coverage
"examining aspects of the product both intentionally *and incidentally*..."
Oracles
"applying our knowledge *and feelings* to detect important problems..."

Rapid Testing Building Blocks

Rapid Software Testing uses a social and systems science approach
informed and inspired by Jerry Weinberg, Herbert Simon, and Harry Collins

- **Context.** We listen and respond to the world around us.
- **Role and Self-Image.** Taking responsibility for your work.
- **Mission and Motivation.** Knowing what you are here to do.
- **Ethics and Integrity.** Rejecting waste and deception.
- **Diversity.** You need variety to cover complex products.
- **Relationships.** Working with ever-changing connections.
- **Models.** Respecting both tacit and explicit knowledge.
- **Skills.** Developing your abilities on the job.
- **Heuristics.** Fallible ideas and tools that solve problems.
- **Exploration.** Everything evolves; answers come over time.
- **Product Risk.** Danger of a bad bug hiding in the product.
- **Tests.** Not test cases… Actual tests!

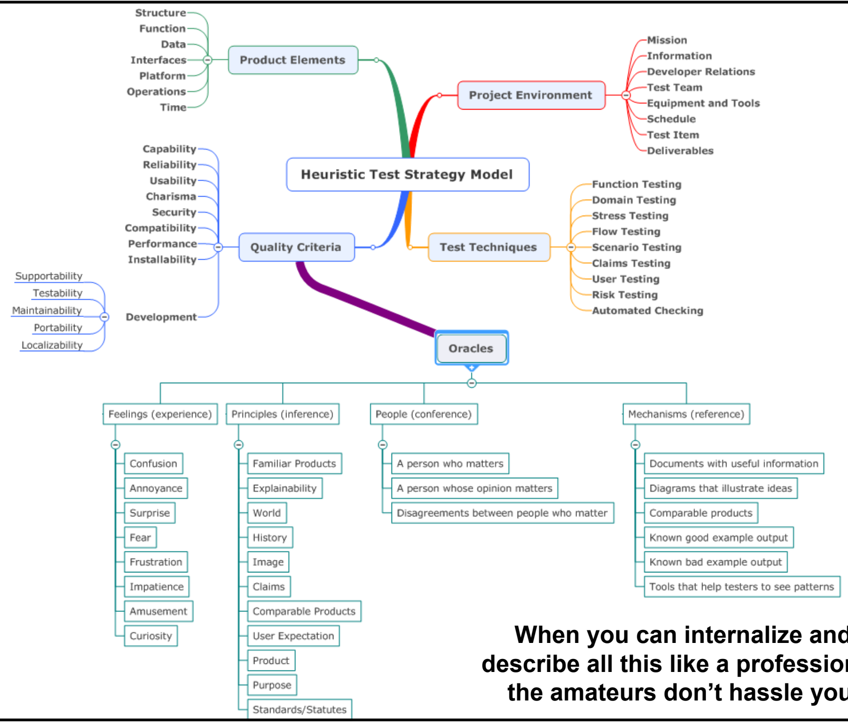
What about…? Quick Answers!

- **Reporting.** Testers must learn to report and explain.
- **Speech.** Precise!
- **Documentation.** Concise! (Conversation is good.)
- **Management.** We focus on activities, not artifacts.
- **Metrics.** Never count test cases; maybe count time.
- **Automation.** **We use tools**. Tools are important. Tools can help with many things, including checking. But *testing* can't be automated.

All of these points are consistent with the Agile Manifesto and Agile principles.

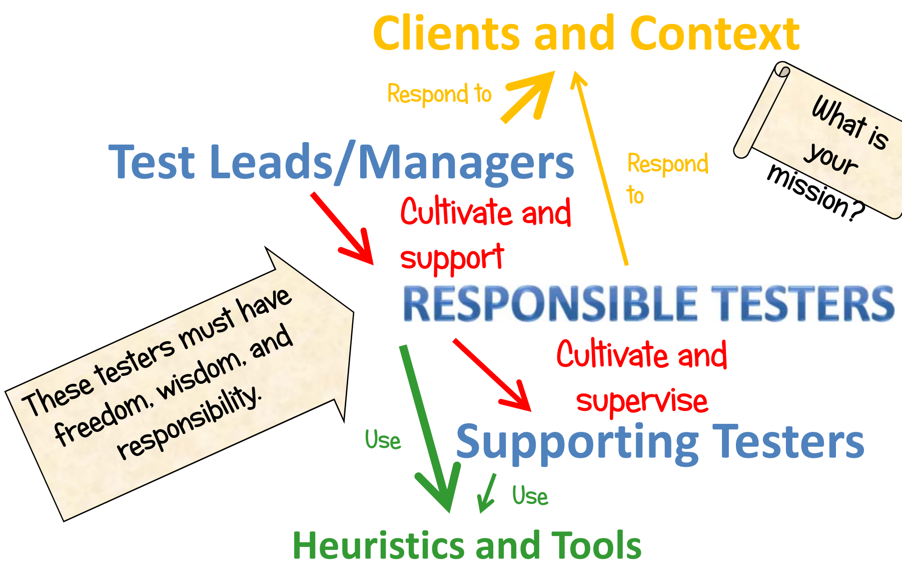
Testing begins and ends with models.

- **A model is an idea, activity, or object…**
  such as an idea in your mind, a diagram, a list of words, a spreadsheet, a person, a toy, an equation, a demonstration, or a program…
- **…that represents another idea, activity, or object.**
  such as something complex that you need to work with or study.
- **…A GOOD model is one that helps you understand or manipulate the thing that it represents.**
  - A map helps navigate across a terrain.
  - "2 + 2 = 4" is a model for figuring out how many apples are in a basket when we add two apples to a basket that already has two apples in it.
  - Atmospheric models help predict where hurricanes will go.
  - A fashion model helps people to understand how clothing would look on actual humans (okay, really skinny humans).
  - Your beliefs about what you test are a model of what you test.



When you can internalize and describe all this like a professional, the amateurs don't hassle you.

Model: Three Testing Roles



Mission

Testing Is *Serving the Clients*

If you don’t have an understanding and an agreement on the mission of your testing, then doing it “rapidly” would be pointless.

Know your mission.

↘

Begin sympathetically…

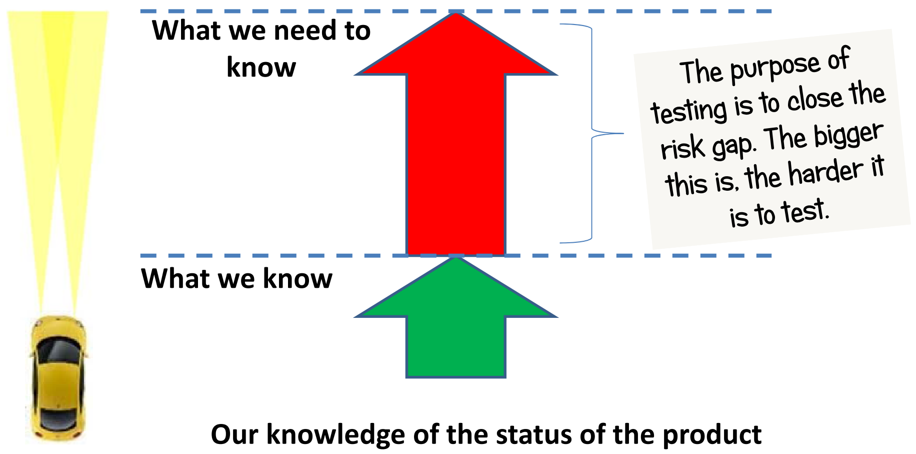
↘

…Then chase the risk!

Risk

Develop risk-focused strategy and skills of risk analysis.

The Risk Gap


What we need to know
What we know
The purpose of testing is to close the risk gap. The bigger this is, the harder it is to test.

Our knowledge of the status of the product

Typical Trouble with Risk Analysis

- Testers afraid of anything that has the word “analysis” in it, and looking for easy answers.
- Instead of product risks, focusing instead on *project* risks.
- Having no systematic method of risk analysis.
- Conceiving of big huge categories, or tiny specific bugs, but nothing in between.

See “Risk Analysis Heuristics (for Digital Products)”, attached to this report.

A Product Risk Story

"A victim will suffer a problem because of a vulnerability in the product, triggered by some threat."

Unpacking the Product Risk Story

"Some person(s) will experience a problem with respect to something desirable that can be detected in some set of conditions because of a vulnerability in the system."

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…

Stakeholders

Context

Problem

Quality Criteria

Oracles

Test Conditions

Theory of Error

Product Factors

An Example Table of Risk Elements

Where is the problem?	Can we see it?	When would we see it?	How often does it happen?	How regularly?	What is affected?	Quality criterion?	Who is affected?	How do they feel?
Structure	Obvious	Immediately	Frequently	Consistent	Structure	Capability	Society	Impatient
Function	Obscure	Later	Rarely	Intermittent	Function	Reliability	Public	Confused
Data	Invisible	Never			Data	Usability	Consumer	Annoyed
Interfaces	Can others?				Interfaces	Charisma	Network Admin	Surprised
Platform					Upstream Platform	Security	External Developer	Disappointed
Operations					Downstream Platform	Scalability	Internal Developer	Angry
Time					Operations	Coexistence	Tester	Afraid
					Time	Inter-operability	Tech writer	Suspicious
						Performance	Malicious user	Uninformed
						Installability	Incompetent	Paralyzed
						Configurability	Bystander	
						Supportability		
						Testability		
						Maintainability		
						Portability		
						Localizability		

Some Geometric Risk Analysis Heuristics

- **Cardinality:** Can there be 0, 1, or more than one object?
- **Boundaries:** Is there a limit? More than one? Are different limits consistent?
- **Extrapolation:** If we can go THIS far, can we go FARTHER?
- **Interpolation:** If two things exist in different places, does something exist between them?
- **Intersections:** Do components collide? Can one contaminate another?
- **Surface Integrity:** Does behavior change correctly as input changes in any given dimension?
- **Symmetry/Asymmetry:** If a behavior exists for A, does a corresponding behavior exist for B?
- **Pattern Completion:** Is a pattern apparent that has not yet been completed, or is obscured (all customary parts of a shape)?
- **Negation:** Whatever is there might become its opposite.

Analyzing Risk

- Consider creating and maintaining a *product* risk list and a *project* risk list.
- Brainstorm a list of risks, and rank them in order of significance
- Then compare this list to coverage and quality criteria areas in the HTSM.
- Identify tasks associated with investigating and managing risks.
- Do some testing *not* focused on specific risks, in order to discover unrecognized risks.

See RST Appendices, http://www.satisfice.com/rst-appendices.pdf.
Examples include “Install Risk Catalog”, “OWL Quality Plan”, “Test Plan”, “OEW Case Tool”.

Strategy

What is a test strategy?

Test strategy is the set of ideas that guide your choice of tests.

A **set of ideas** does not necessarily mean a document. The test strategy may be entirely in your head. Or it may be in several heads, and emerge through discussion, over time.

It may be documented partially on a whiteboard or Post-Its or in a mindmap. Or it could be in a formal document all dressed like Cinderella at the royal ball.

What is a test strategy?

Test strategy is the set of ideas that guide your choice of tests.

To **guide** is to influence but not necessarily to determine. Testing is shaped by many factors in addition to strategy, including opportunities, skills, mistakes, time pressures, limitations of tools, testability, and unconscious biases.

What is a test strategy?

Test strategy is the set of ideas that guide your choice of tests.

I mean **choice** in the most expansive sense of the word, not simply the selection of existing test cases.

Choice of tests includes choices of **what tests to design** and **how to design them** and **all decisions** made during test design. It includes choices made during **test execution**, too, including **how to perform tests** and **what mix of tests** to perform in response to which perceived risks.

Why have a test strategy?

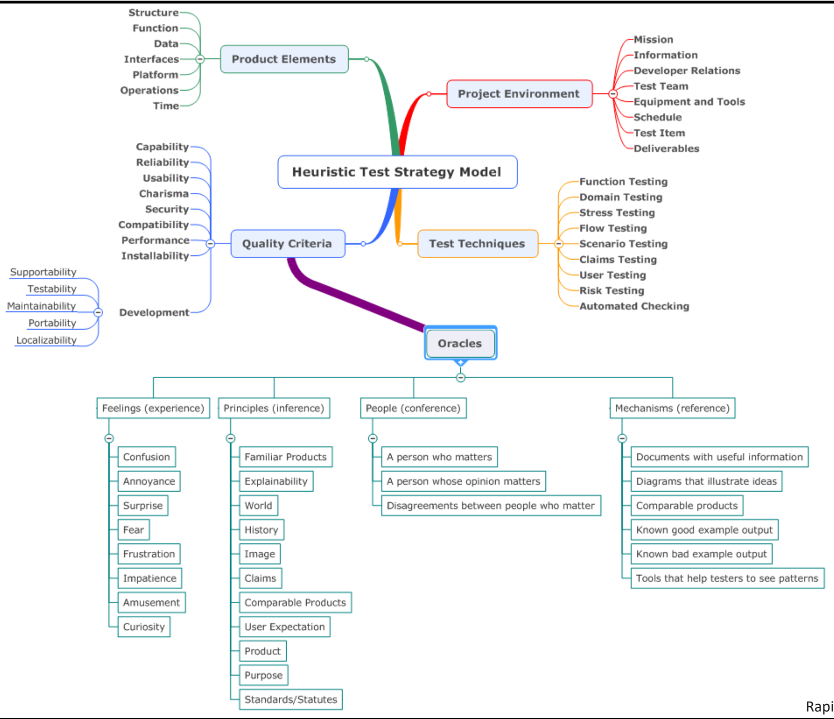
If you are testing—even if you are only *thinking* about testing—you *already* have a test strategy, so that's not a meaningful question.

Here are some better questions:

- Why *worry* about your test strategy?
- Why have an *explicit* test strategy?
- Why *explain* it? Why *document* it? Why talk about it?

Possible answers:

- to help steer your strategy, focus it, or adapt it
- to share it with others
- to gain credibility and accountability
  …for less time, less effort, or higher value.
- *Because tools and test scripts don't talk.*



Oracles

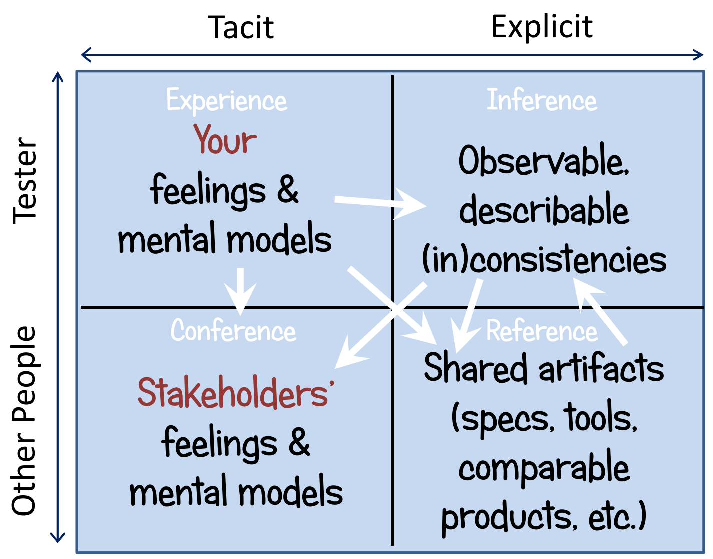
How Do We Recognize Problems?

An oracle is…

a means of

recognizing a problem.

Oracles from the Inside Out


Tacit
Explicit
Tester
Other People
Experience
Your feelings & mental models
Inference
Observable, describable (in)consistencies
Conference
Stakeholders' feelings & mental models
Reference
Shared artifacts (specs, tools, comparable products, etc.)

Consistency ("this agrees with that")

an important theme in oracle principles

- **Acceptability:** The product *is consistent* with how good it can reasonably be (not just good; good *enough*).
- **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:** 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.

User desires can be expressed as Quality Criteria

CRUCSSCPID

Capability	Scalability
Reliability	Compatibility
Usability	Performance
Charisma	Installability
Security	Development (next slide…)

Many test approaches focus on testing for capability (functionality) and underemphasize the other criteria. Yet any inconsistency may represent diminished value for some person who matters.

Some "Users"—the Development Organization—Have Special Quality Criteria

Remember Testability!

Supportability
Testability
Maintainability
Portability
Localizability

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

General Examples of Oracles

things that suggest "problem" or "no problem"

- A comparable product or algorithm.
- A process or tool that checks output.
- A process or tool that helps a tester identify patterns.
- A reference document with useful information.
- Known good or bad example output.
- A person whose opinion matters.
- Opinions held by a person who matters.
- A disagreement among people who matter.
- A feeling like confusion or annoyance.
- *A desirable consistency between related things.*

Mechanisms

Artifacts

People

Feelings ⟺ ***Principles***

All Oracles Are Heuristic

An oracle doesn't tell you that there IS a problem.
An oracle tells you that you *might be seeing a problem.*

An oracle can alert you to a possible problem,
but an oracle *cannot* tell you that there is *no* problem.

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

Rely solely on documented, anticipated sources of oracles,
and your testing will likely be slower and weaker.

Train your mind to recognize *patterns* of oracles
and your testing will likely be faster
and your ability to spot problems will be sharper.

Coverage

What IS Coverage?

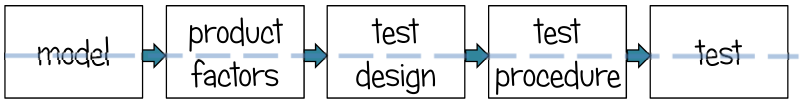
_______ coverage is "how thoroughly we have examined the product with respect to some model of ______".

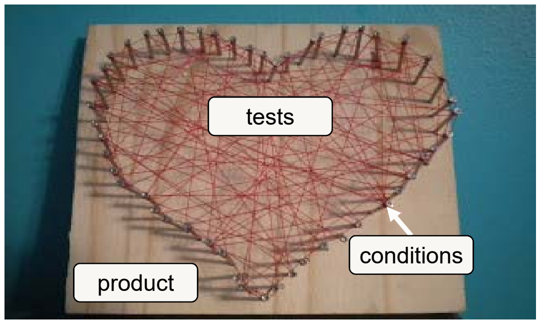
Interesting kinds of coverage

- Product coverage
  - *What aspects of the product did you look at?*
  - *(Code coverage is only one aspect of the product)*
- Risk coverage
  - *What risks have you tested for?*
- Requirements coverage
  - *What requirements have you tested for?*

Product Factors

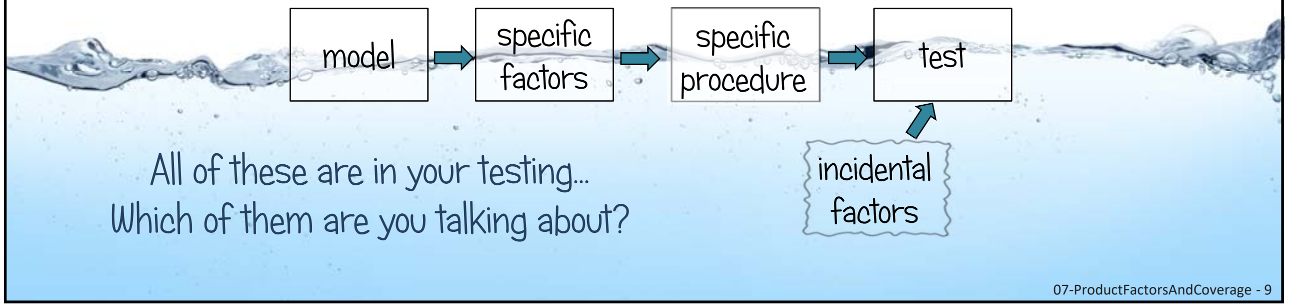
A product factor is anything about the product or its context that could be examined in a test.



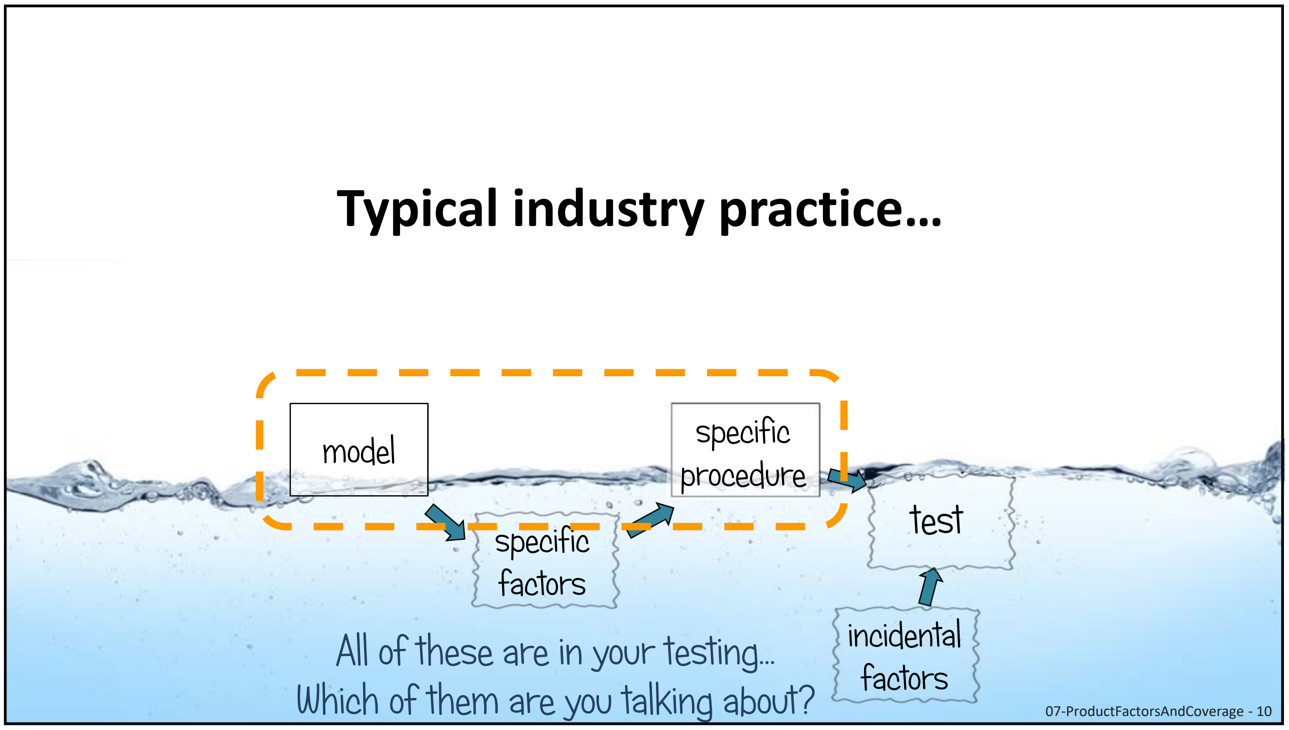


Product Factors

A product factor is anything about the product or its context that could be examined in a test.



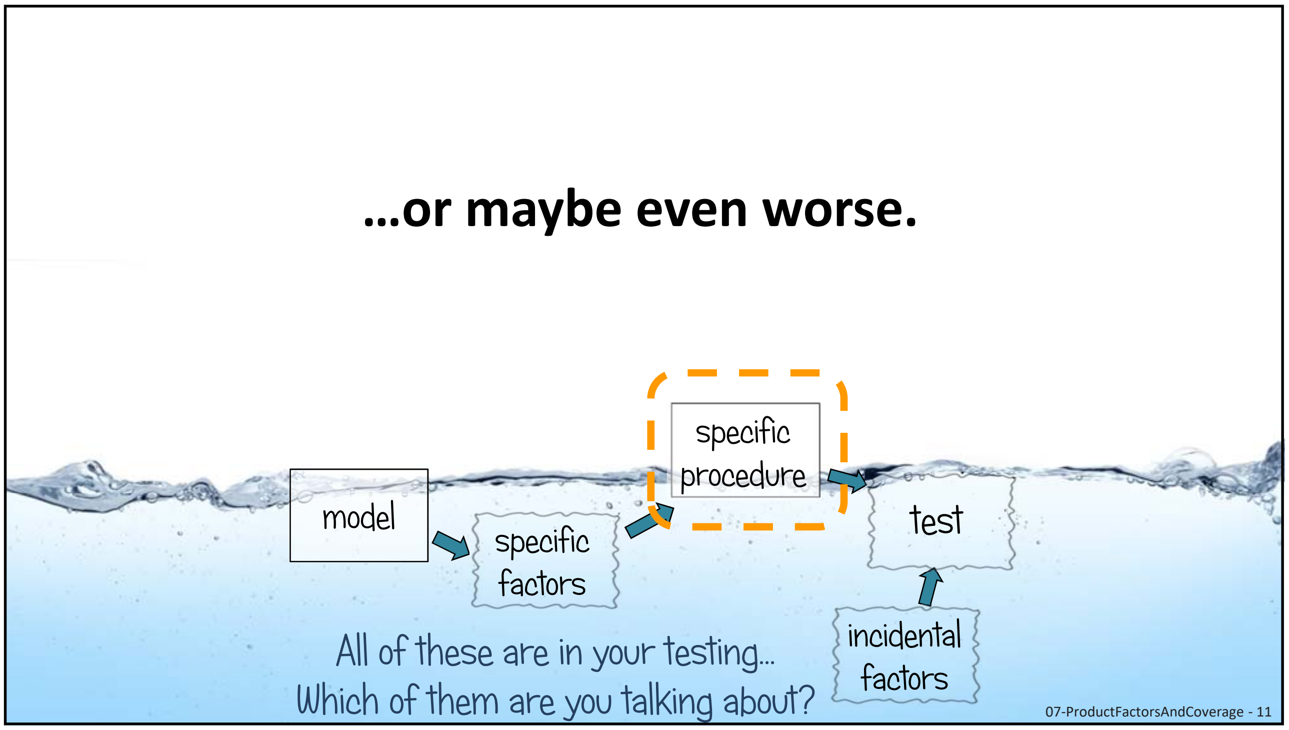
All of these are in your testing...
Which of them are you talking about?



Typical industry practice…

All of these are in your testing…
Which of them are you talking about?

07-ProductFactorsAndCoverage - 10

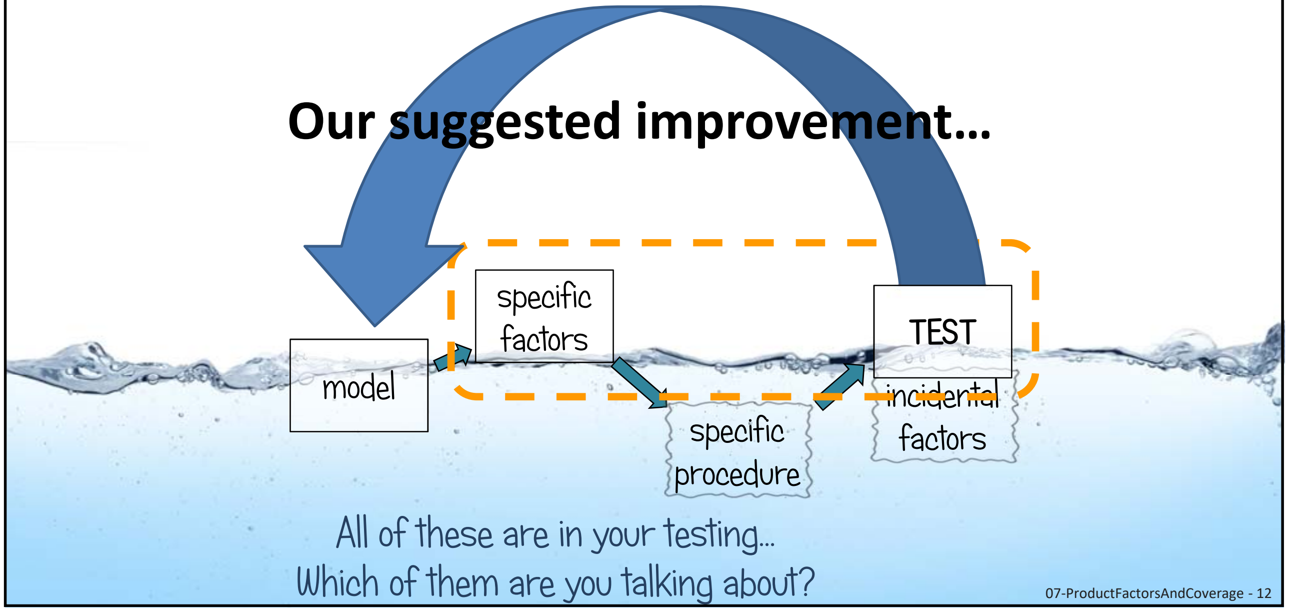


…or maybe even worse.

All of these are in your testing…
Which of them are you talking about?

07-ProductFactorsAndCoverage - 11

Our suggested improvement…



All of these are in your testing…
Which of them are you talking about?

What IS Coverage?

_______ coverage is "how thoroughly we have examined the product 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

There are as many kinds of test coverage as there are ways to model the system.

- Structure
- Function
- Data
- Interfaces
- Platform
- Operations
- Time
- Technical Risk
- Business Risk
- …

…and each kind of coverage can be obtained *intentionally*, *incidentally*, or *accidentally*.

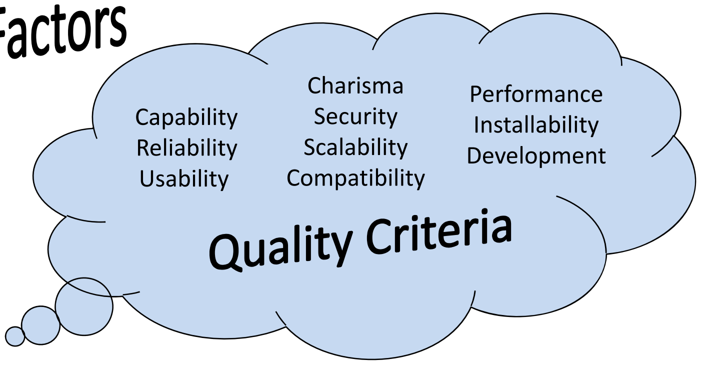
See "Got You Covered", "Cover or Discover",
and "A Map By Any Other Name"
http://www.developsense.com/publications.html

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

SFDIPOT – "San Francisco Depot"

Product Factors

- Structure
- Function
- Data
- Interfaces
- Platform
- Operations
- Time



Product Coverage Outline

- A **product coverage outline** is an artifact (a map, or list, or table…) that identifies the dimensions or elements of a product that might be relevant to testing it. (Sketches or diagrams can help too.)
- **"dimensions of the product"** means attributes of the product that you can describe. They may represent differences between two products, or within the same product over time.
- **"relevant to testing"** means that there is probably some value to some client, with respect to some testing mission, of observing, studying, manipulating or understanding a particular dimension.

- The Product Elements section of the Heuristic Test Strategy Model provides a point of departure for creating a coverage outline.
- Creating a coverage outline requires and exercises the skill of *factoring—identifying dimensions of interest in a product*

Coverage

Level 0	**We don't really know anything about this area.** We're aware that this area exists, but it's a black box to us, so far.
Level 1	**We're just getting to know this area.** We've done basic reconnaissance; surveyed it; we've done smoke and sanity testing. We may have some artifacts that represent our models, which will helps us to talk about them and go deeper.
Level 2	**We've learned a good deal about this area.** We've looked at the core and the critical aspects of it. We've done some significant tests focused on the most important quality criteria, and we're collecting and diversifying our ideas on how to cover it deeply.
Level 3	**We have a comprehensive understanding of this area.** We've looked deeply into it from a number of perspectives, and applied a lot of different test techniques. We've done harsh, complex, and challenging tests on a wide variety of quality criteria. If there were a problem or unrecognized feature in this area that we didn't know about, it would be a big surprise.

See http://developsense.com/articles/2008-09-GotYouCovered.pdf, http://developsense.com/articles/2008-10-CoverOrDiscover.pdf, http://developsense.com/articles/2008-11-AMapByAnyOtherName.pdf

Activities

One Big Problem in Testing

Formality Bloat

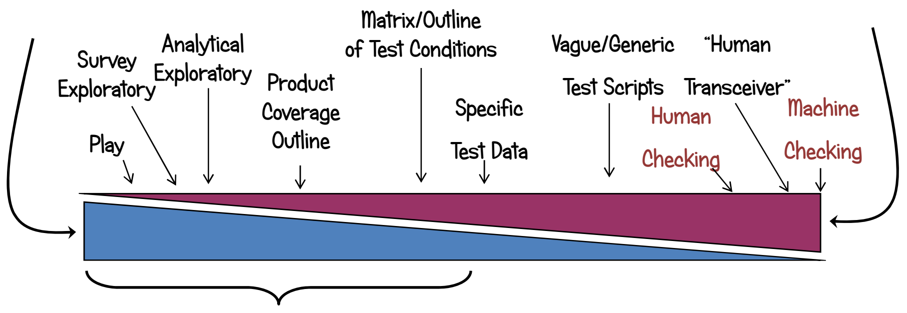
- Much of the time, your testing doesn't need to be very formal*
- Even when your testing *does* need to be formal, you'll need to do substantial amounts of informal testing in order figure out how to do *excellent* formal testing.
  - Who says? The FDA. See http://www.satisfice.com/blog/archives/602
- Even in a highly regulated environment, you do *formal* testing primarily for the auditors. You do informal testing to make sure you don't lose money, blow things up, or kill people.

* Formal testing means testing that must be done to verify a specific fact, or that must be done in a specific way.

The Formality Continuum

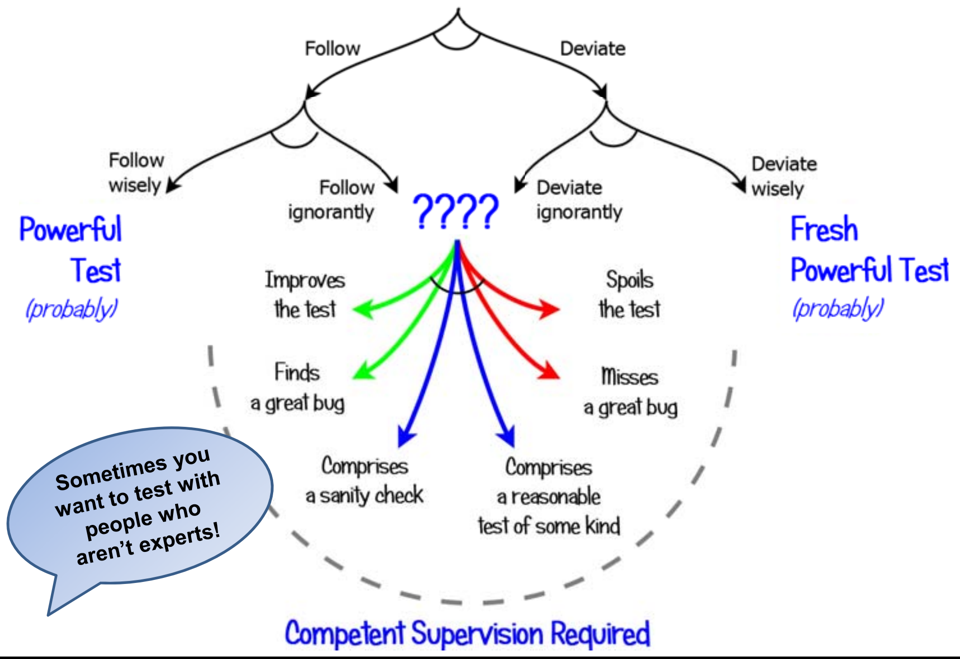
INFORMAL
Not done in any specific way, nor to verify specific facts.

FORMAL
Done in a specific way, or to verify specific facts.

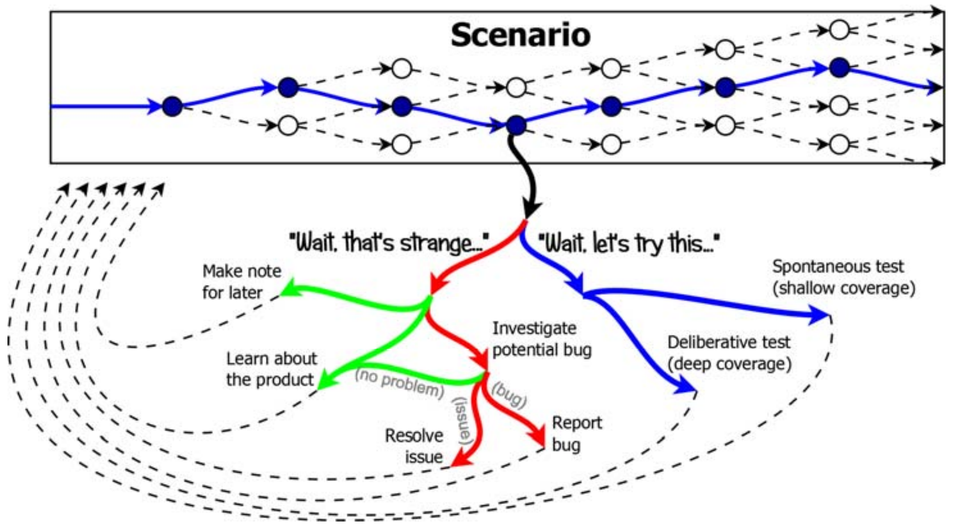


All testing is exploratory; all testing is scripted. These days, when we refer to "exploratory testing", we generally do it to emphasize the exploratory nature of activities on the *informal* side of this continuum.

Testing with a Formal Procedure



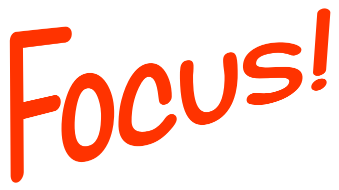
Testing with a Less Formal Procedure



Scenario
"Wait, that's strange..."
"Wait, let's try this..."
Make note for later
Learn about the product
(no problem)
Investigate potential bug
(bug)
(issue)
Resolve issue
Report bug
Spontaneous test (shallow coverage)
Deliberative test (deep coverage)

Rapid Intro to Rapid Testing - 69

To test a *very simple* product meticulously, *part* of a complex product meticulously, or to maximize test *integrity*…



1. Start the test from a *known* (clean) state.
2. Prefer *simple, deterministic* actions.
3. Trace test steps to a *specified model*.
4. Follow *established and consistent* lab procedures.
5. Make *specific* predictions, observations and records.
6. Make it *easy to reproduce* (tools may help).

Rapid Intro to Rapid Testing - 70

General Focusing Heuristics

- use test-first approach or unit testing for better *code* coverage
- work from prepared test coverage outlines and risk lists
- use diagrams, state models, and the like, and cover them
- apply specific test techniques to address particular coverage areas
- make careful observations and match to explicit oracles

Follow your procedures.

To do this *more rapidly*, make *preparation* and *artifacts* fast and frugal:
leverage existing materials and avoid repeating yourself.
Emphasize doing; relax planning. You'll make discoveries along the way!

To find *unexpected problems*,
***elusive problems* that occur in sustained field use,**
or more problems *quickly* in a complex product…

De-Focus!

1. Start from *different states* (not necessarily clean).
2. Prefer *complex, challenging* actions.
3. Generate tests from a *variety* of models.
4. *Question your lab procedures and tools.*
5. Try to *see everything* with open expectations.
6. Make the test *hard to pass*, instead of easy to reproduce.

PowerPoint bug: an italic first letter of the line makes the number italic too!

General Defocusing Heuristics

- diversify your models; intentional coverage in one area can lead to unintentional coverage in other areas—this is a Good Thing
- diversify your test techniques
- be alert to problems other than the ones that you're actively looking for
- welcome and embrace productive distraction
- do some testing that is *not* oriented towards a specific risk
- use high-volume, randomized automated checks

Question and vary your procedures.

Cost as a Simplifying Factor
Try quick tests as well as careful tests

A *quick test* is a cheap test that has some value but requires little preparation, knowledge, or time to perform.

- Happy Path
- Product Tours
  - *Sample Data*
  - *Variables*
  - *Files*
  - *Complexity*
  - *Menus & Windows*
  - *Keyboard & Mouse*

- Interruptions
- Undermining
- Adjustments
- Dog Piling
- Continuous Use
- Feature Interactions
- Click on Help

Cost as a Simplifying Factor
Try quick tests as well as careful tests

A *quick test* is a cheap test that has some value but requires little preparation, knowledge, or time to perform.

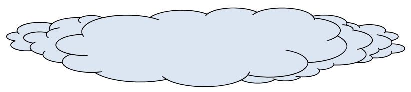
- Input Constraint Attack
- Click Frenzy
- Shoe Test
- Blink Test
- Error Message Hangover
- Resource Starvation
- Multiple Instances
- Crazy Configs
- Cheap Tools

Managing Exploratory Work

A Key Problem for Test Managers

Engineering is an exploratory process that relies on skill, knowledge, and motivation. Lots of important and deep work happens without pre-existing instructions...

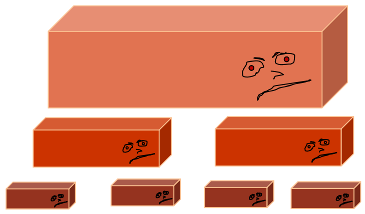
Like a...



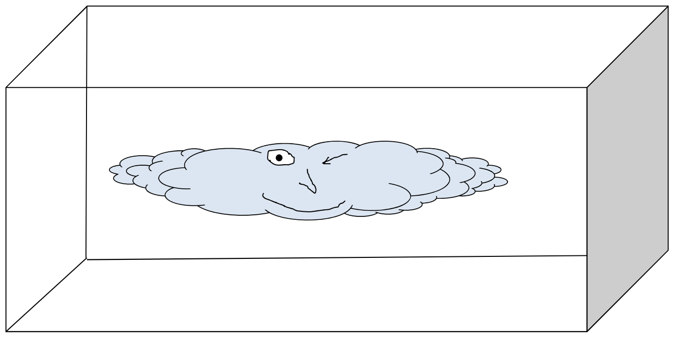
...mysterious cloud!

But managers often think in terms of discrete tasks and outcomes...

Like...



...bricks?



Solution: Put the cloud into a fake brick.

Three Forms of Test Management

- **People-based:** Account for the people who test.

  "Jerry tests the back-end. Michael tests the front-end."

- **Artifact-based:** Account for tangible work products.

  "Here's the 217 test cases we created."

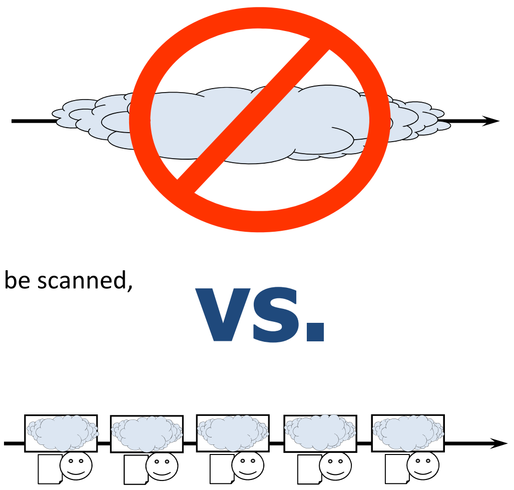
- **Activity-based:** Account for the things that testers do.

  "Here are the test activities that comprise our strategy. We did 17 test sessions this week, so far. Mostly, scenario testing."

Two kinds of activity-based management: *thread* or *session*

Accountability for Exploratory Work: Session-Based Test Management

- Time Box
  - Typically 90-minutes (+/- 45)
- Charter
  - A clear, concise mission for a test session
- Reviewable Results
  - a session sheet—a test report that can be scanned, parsed and compiled by a tool
- Debriefing
  - a conversation between tester and manager or test lead



See http://www.satisfice.com/sbtm.

Time Box:

Focused test effort of fixed duration

A **normal** session is 90 minutes long.

A **real** session may be somewhat longer or shorter.

A **normal** session is uninterrupted.

A **real** session may be somewhat interrupted.

Real sessions are "normed" for the purposes of reporting metrics.

This is so that our clients don't get confused by the numbers.

Start with *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)

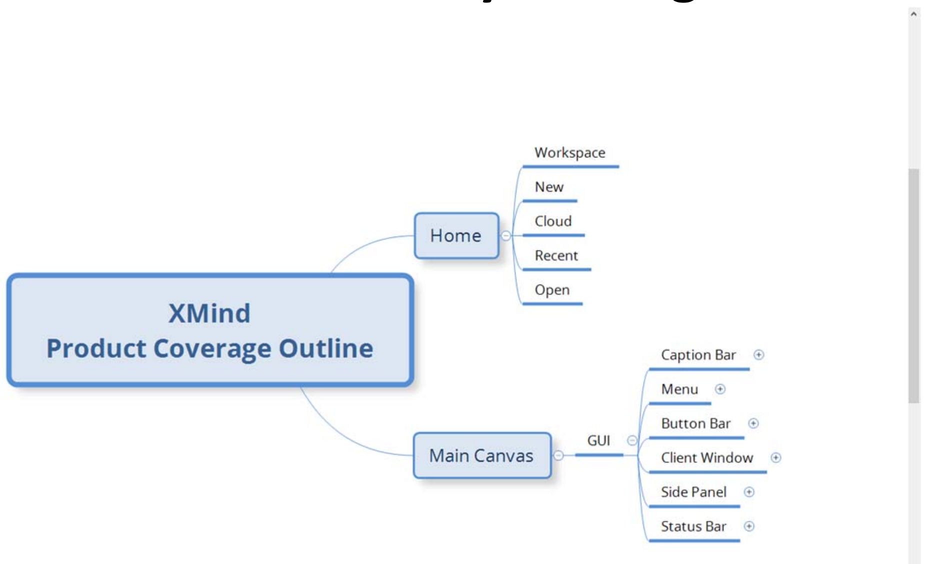
  "Develop a library of mindmaps for each major feature area. Use SFDIPOT as a checklist for coverage analysis."

  "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."

Learn the Product by Touring

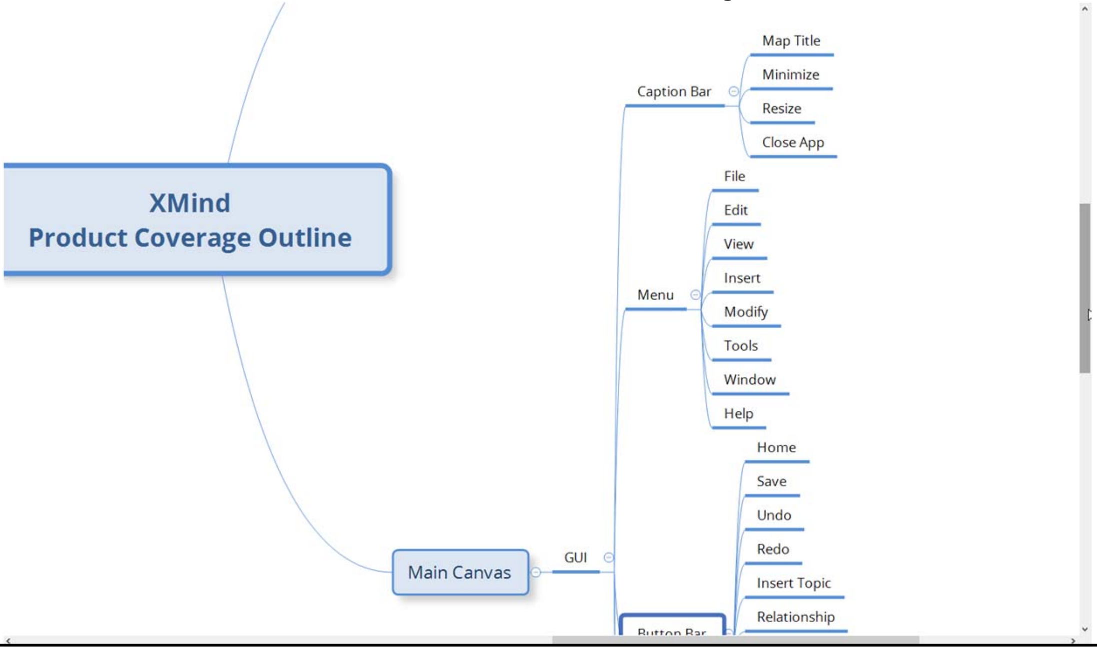


Touring the Product: Mike Kelly's FCC CUTS VIDS

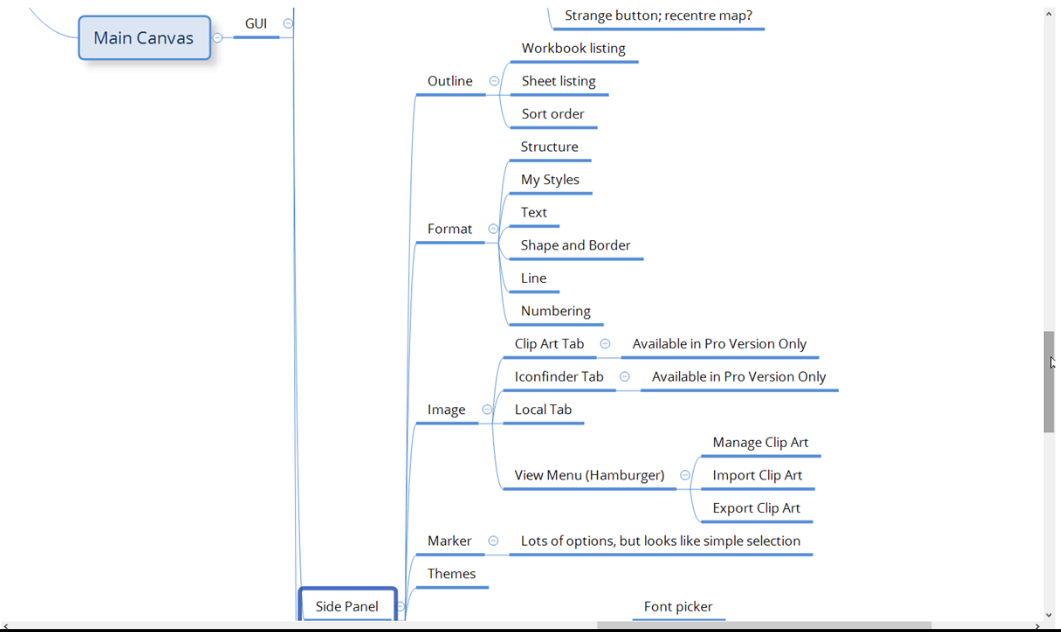
- Feature tour
- Complexity tour
- Claims tour
- Configuration tour
- User tour
- Testability tour
- Scenario tour
- Variability tour
- Interoperability tour
- Data tour
- Structure tour

Create your own lists!

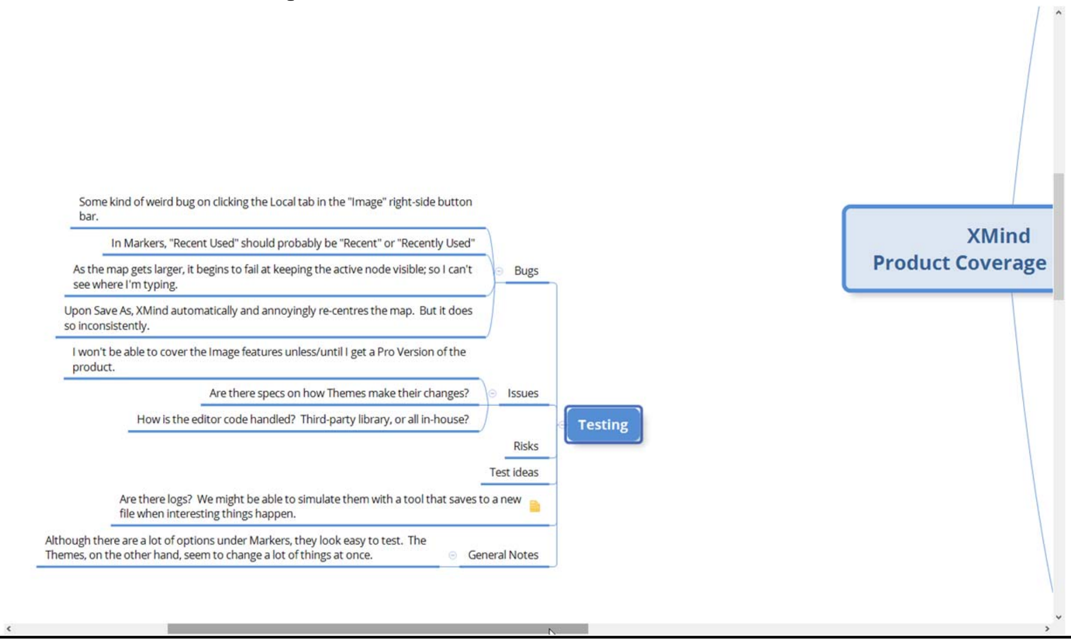
Iterate and Go Deeper



Keep Going Deeper Still



Keep Test Notes as You Go



Some kind of weird bug on clicking the Local tab in the "Image" right-side button bar.

In Markers, "Recent Used" should probably be "Recent" or "Recently Used"

As the map gets larger, it begins to fail at keeping the active node visible; so I can't see where I'm typing.

Upon Save As, XMind automatically and annoyingly re-centres the map. But it does so inconsistently.

I won't be able to cover the Image features unless/until I get a Pro Version of the product.

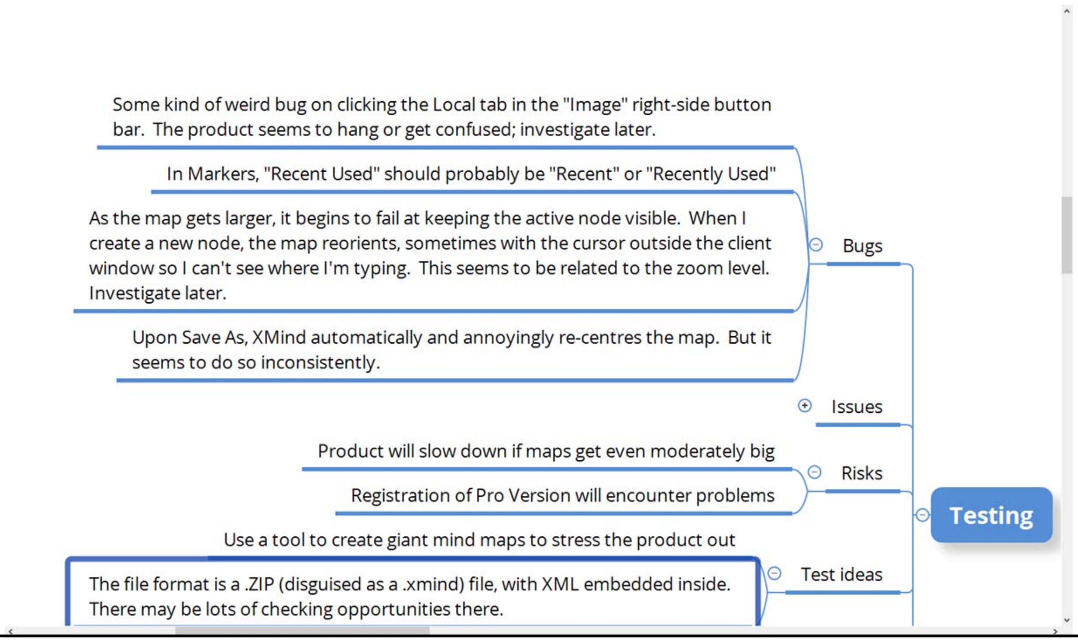
Are there specs on how Themes make their changes?

How is the editor code handled? Third-party library, or all in-house?

Are there logs? We might be able to simulate them with a tool that saves to a new file when interesting things happen.

Although there are a lot of options under Markers, they look easy to test. The Themes, on the other hand, seem to change a lot of things at once.

Discover Bugs While Learning!



Some kind of weird bug on clicking the Local tab in the "Image" right-side button bar. The product seems to hang or get confused; investigate later.

In Markers, "Recent Used" should probably be "Recent" or "Recently Used"

As the map gets larger, it begins to fail at keeping the active node visible. When I create a new node, the map reorients, sometimes with the cursor outside the client window so I can't see where I'm typing. This seems to be related to the zoom level. Investigate later.

Upon Save As, XMind automatically and annoyingly re-centres the map. But it seems to do so inconsistently.

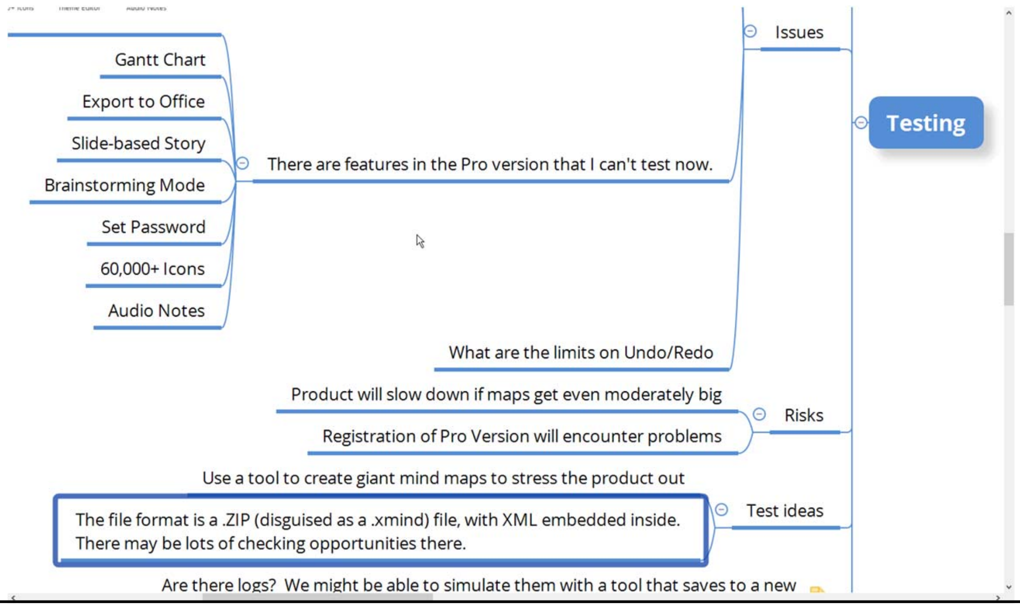
Product will slow down if maps get even moderately big

Registration of Pro Version will encounter problems

Use a tool to create giant mind maps to stress the product out

The file format is a .ZIP (disguised as a .xmind) file, with XML embedded inside. There may be lots of checking opportunities there.

Note Issues While Learning!


Testing
Issues
There are features in the Pro version that I can't test now.
Gantt Chart
Export to Office
Slide-based Story
Brainstorming Mode
Set Password
60,000+ Icons
Audio Notes
What are the limits on Undo/Redo
Risks
Product will slow down if maps get even moderately big
Registration of Pro Version will encounter problems
Test ideas
Use a tool to create giant mind maps to stress the product out
The file format is a .ZIP (disguised as a .xmind) file, with XML embedded inside. There may be lots of checking opportunities there.
Are there logs? We might be able to simulate them with a tool that saves to a new

Feed *Learning* into…

- Analysis Sessions (Goal: get deep coverage ideas)

  "Identify primary components and interactions with external applications."

  "Survey the OWASP Top 10 Security Risks page."

  "Perform comparative analysis on four major competitors."

  "Brainstorm a risk list for botched conversion of legacy data."

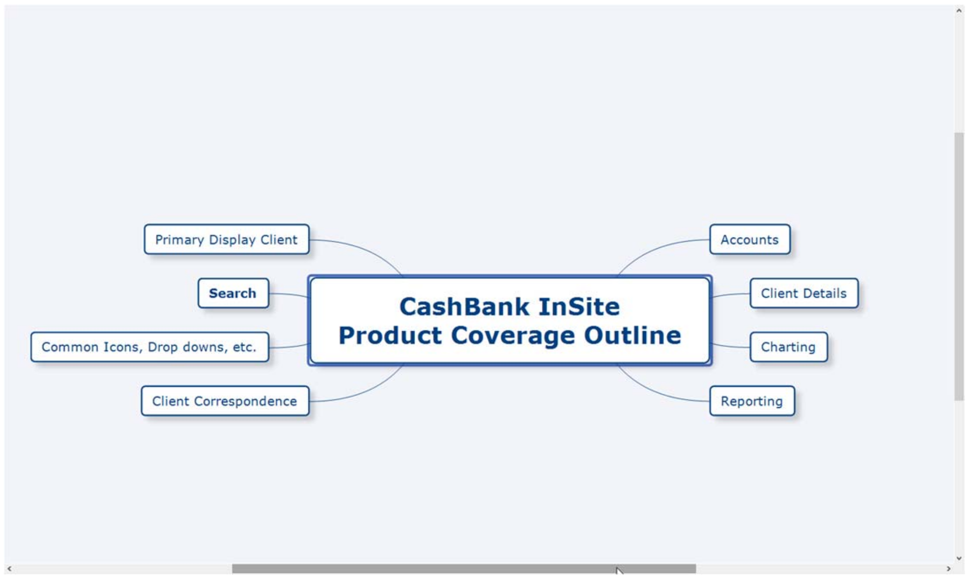
  "Prepare a preliminary finite-state model using StateMaker."

  "Review platform dependencies to identify performance bottlenecks and resource contention."

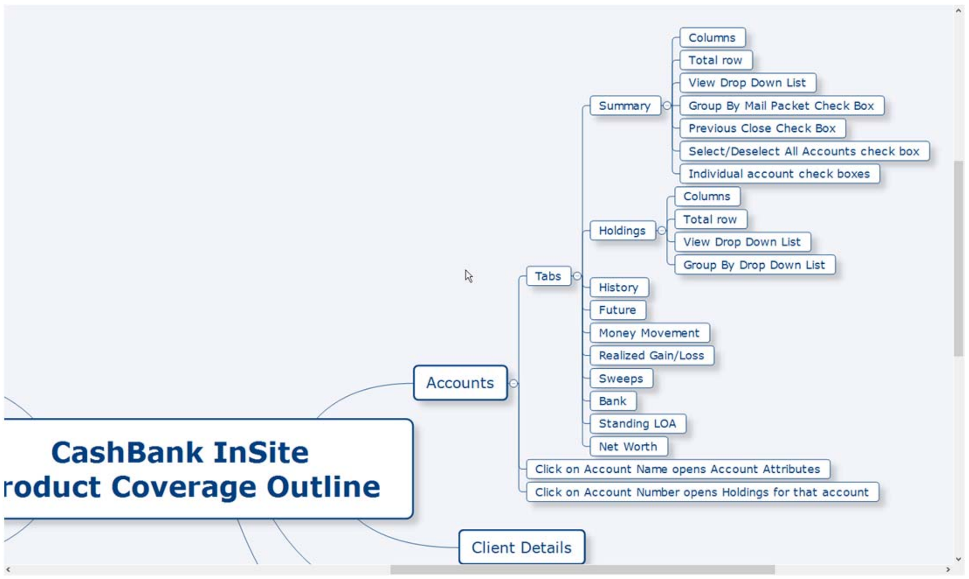
  "Create tools to generate data of arbitrary size and complexity."

  "Review customer support logs for common problems and patterns of misuse."

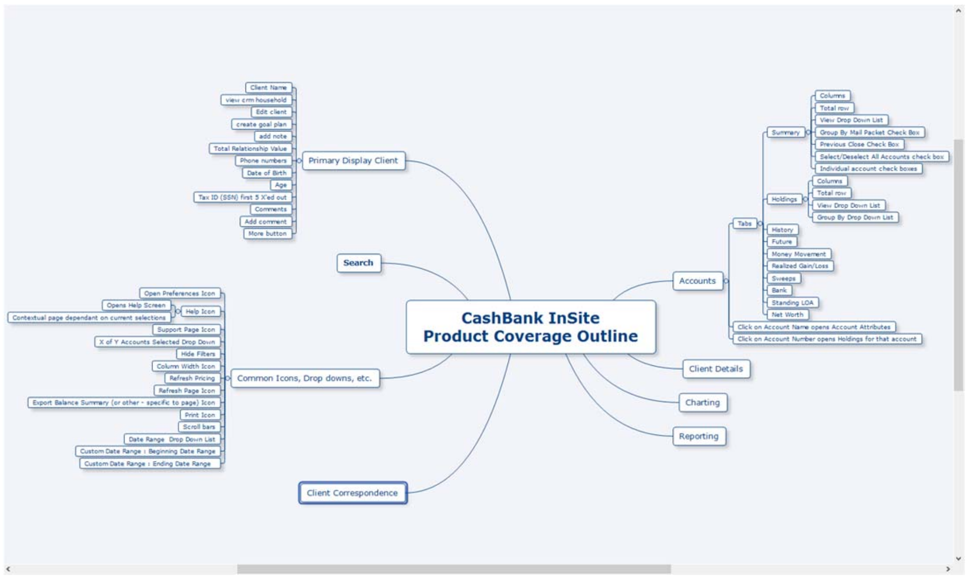
Attend that Feature Meeting



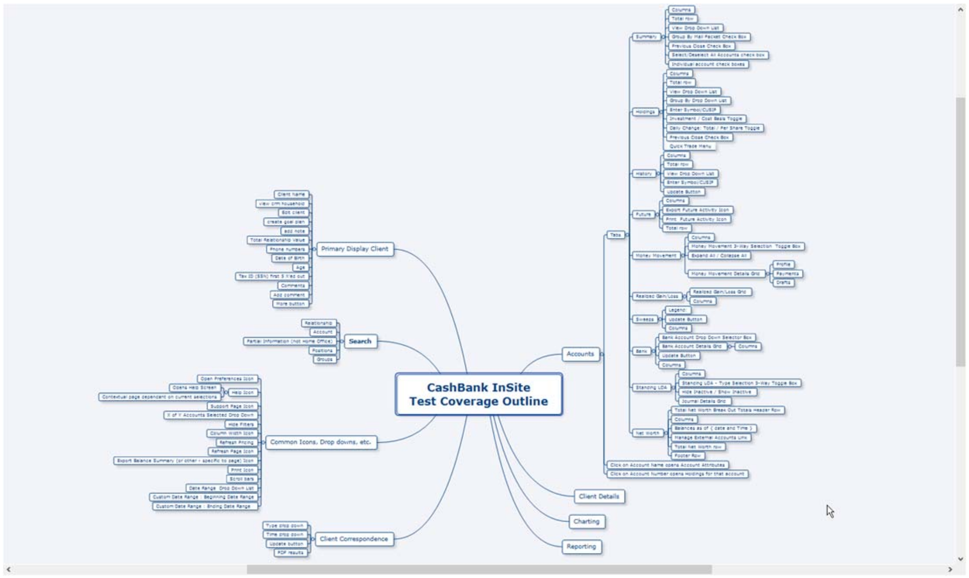
Study the Specs



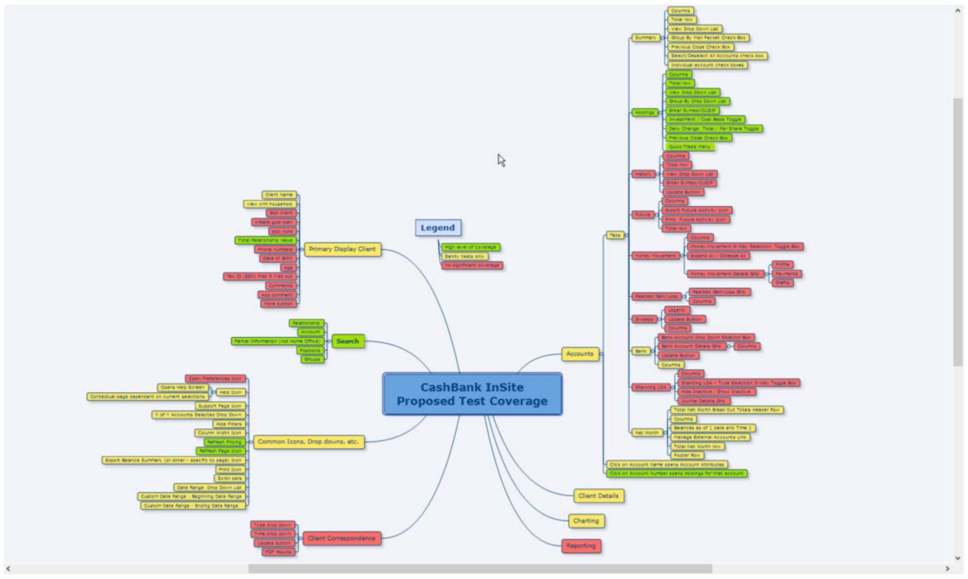
Model the Test Space



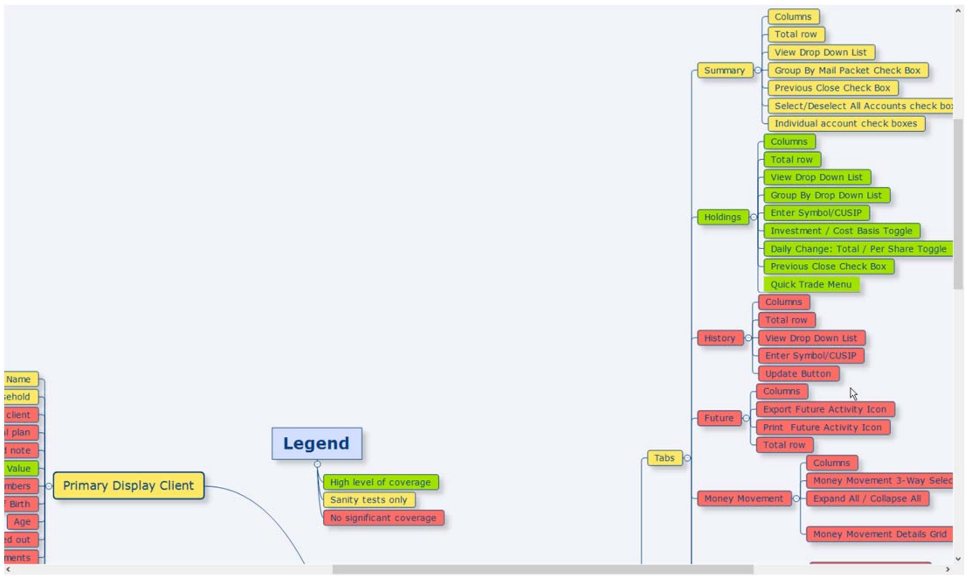
Now Bring in the Client!



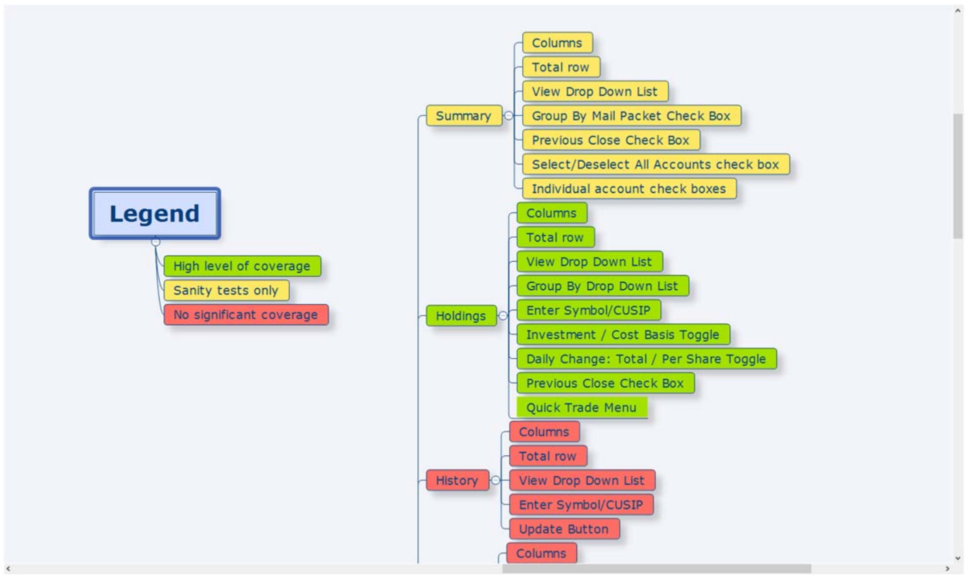
Negotiate Coverage



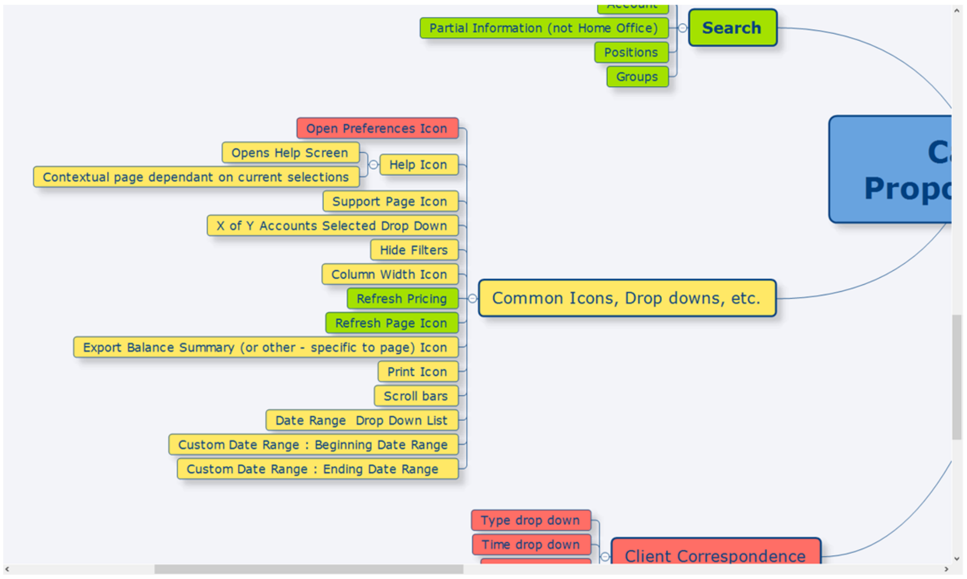
Focus on the Risky Stuff



Feature Area Coverage



Interface Coverage


Search
Partial Information (not Home Office)
Positions
Groups
Common Icons, Drop downs, etc.
Open Preferences Icon
Help Icon
Opens Help Screen
Contextual page dependant on current selections
Support Page Icon
X of Y Accounts Selected Drop Down
Hide Filters
Column Width Icon
Refresh Pricing
Refresh Page Icon
Export Balance Summary (or other - specific to page) Icon
Print Icon
Scroll bars
Date Range Drop Down List
Custom Date Range : Beginning Date Range
Custom Date Range : Ending Date Range
Client Correspondence
Type drop down
Time drop down

More Comprehensive Ideas

- …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."

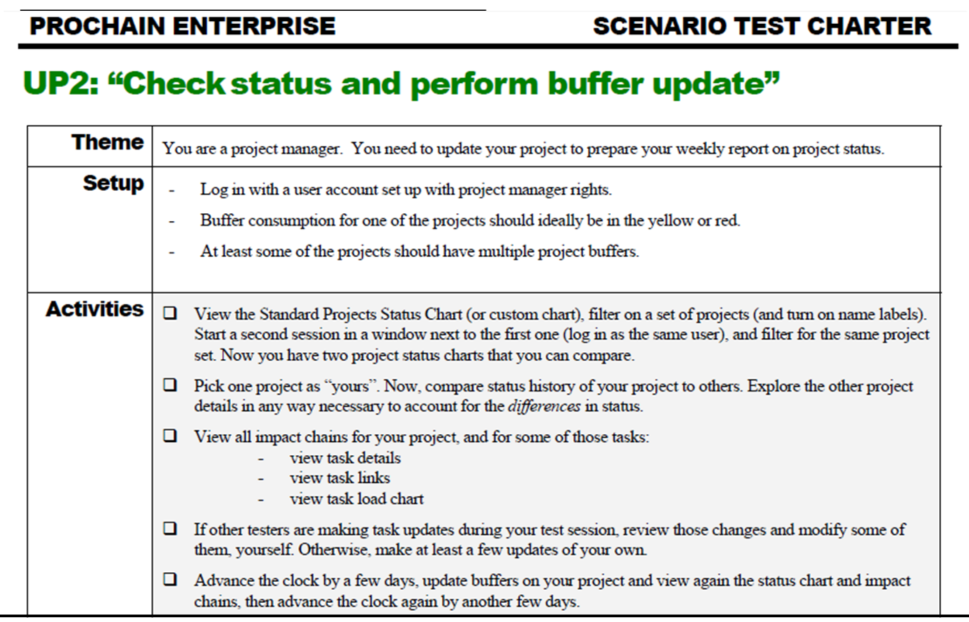
  "Perform steeplechase boundary testing on major data items."

  "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."

(Optional) Formalize Some Charters



PROCHAIN ENTERPRISE **SCENARIO TEST CHARTER**

UP2: "Check status and perform buffer update"

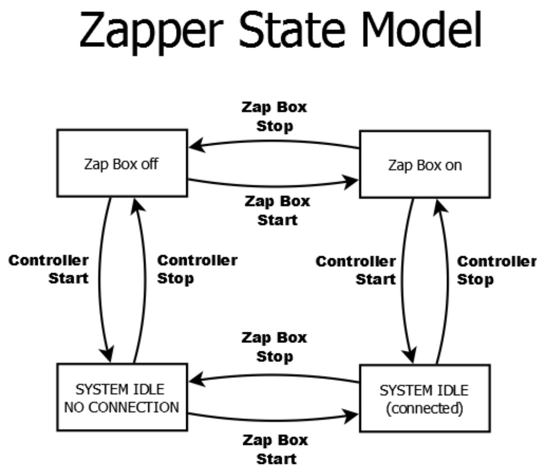
Theme	You are a project manager. You need to update your project to prepare your weekly report on project status.
Setup	- Log in with a user account set up with project manager rights. - Buffer consumption for one of the projects should ideally be in the yellow or red. - At least some of the projects 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. ❑ Advance the clock by a few days, update buffers on your project and view again the status chart and impact chains, then advance the clock again by another few days.

Is This Good Formal Testing?

9.8.1To verify Power Accuracy

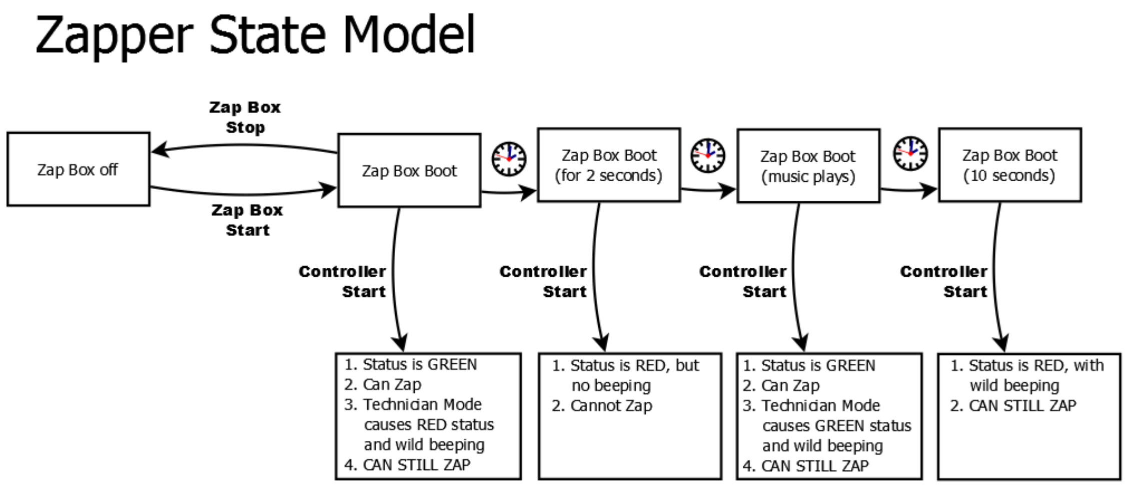
9.8.1.1 Connect the components according to the General Setup document.
9.8.1.2 Power on and connect test jig (instead of electrodes)
9.8.1.3 Power on the Zapper Box.
9.8.1.4 Power on the Control Box.
9.8.1.5 Set default settings of temperature and power for the Zapper Box.
9.8.1.6 Set test jig load to nominal value
9.8.1.7 Select nominal duration and nominal power setting
9.8.1.8 Press the Start button
9.8.1.9 Verify Zapper reports the power setting value ±10% on display.

Assumed State Model for Powering on the System



- There was nothing in the spec about which box to turn on first. The team assumed it didn't matter.
- In the FIRST MINUTE of an exploratory sanity check. The team discovered that it mattered a LOT.

Actual…


Zapper State Model
Zap Box Stop
Zap Box Start
Zap Box off
Zap Box Boot
Zap Box Boot (for 2 seconds)
Zap Box Boot (music plays)
Zap Box Boot (10 seconds)
Controller Start
Controller Start
Controller Start
Controller Start
1. Status is GREEN 2. Can Zap 3. Technician Mode causes RED status and wild beeping 4. CAN STILL ZAP
1. Status is RED, but no beeping 2. Cannot Zap
1. Status is GREEN 2. Can Zap 3. Technician Mode causes GREEN status and wild beeping 4. CAN STILL ZAP
1. Status is RED, with wild beeping 2. CAN STILL ZAP

Prefer Steering to Scripting

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 and **VERIFY** conformance to the requirements and for the presence of any behaviour or attribute that could 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.

After we've learned and tested,
we can decide on formal test cases
and automated checks
IF and HOW and WHEN
they suit our purposes.

Sometimes Extremely Specific Test Design Matters

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 session 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 reported power for the measured electrode from the corresponding power meter reading (use interpolation to synchronize the time stamp of the power meter and generation 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 SOP-QAD-10, or use the equation below for large samples.)

3.5.2.8 The equation for calculating the tolerance interval k is:

$$\boldsymbol{k} = \sqrt{\frac{(\boldsymbol{N}-\mathbf{1})\left(\mathbf{1}+\frac{\mathbf{1}}{\boldsymbol{N}}\right)\boldsymbol{Z}^2_{(\mathbf{1}-\boldsymbol{p})/\mathbf{2}}}{\boldsymbol{\chi}^2_{\boldsymbol{\gamma},\boldsymbol{N}-\mathbf{1}}}}$$

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.)

Skilled, Observant Tester + Oracles = No Need for Excessive Test Documentation!

These two paragraphs replaced 50 pages of overly formal and unhelpful procedural instructions for testing a Class 3 medical device.

3 Test Procedures

3.1 General testing protocol.

In the test descriptions that follow, the word “verify" is used to highlight *specific items that must be checked*. In addition to those items a tester shall, at all times, be alert for *any* unexplained or erroneous behavior of the product. The tester shall bear in mind that, regardless of any specific requirements for any specific test, there is the overarching general requirement that the product shall not pose an unacceptable risk of harm to the patient, including an unacceptable risk using reasonably foreseeable misuse.

3.2 Test personnel requirements

The tester shall be thoroughly familiar with the generator and workstation FRS, as well as with the working principles of the devices themselves. The tester shall also know the working principles of the power test jig and associated software, including how to configure and calibrate it and how to recognize if it is not working correctly. The tester shall have sufficient skill in data analysis and measurement theory to make sense of statistical test results. The tester shall be sufficiently familiar with test design to complement this protocol with exploratory testing, in the event that anomalies appear that require investigation. The tester shall know how to keep test records to credible, professional standard.

Reporting

image credit: istockphoto.com

The Testing Story Is Three Braided Stories

A story about the status of the PRODUCT…

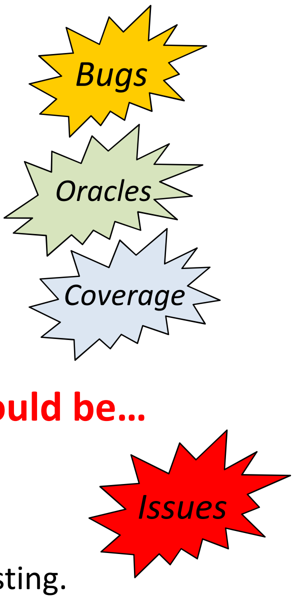
…about what it does, how it failed, and how it might fail...
…in ways that matter to your various clients.

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).

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.



Why Is Part 3 Important?

- *Bugs* threaten the value of the product
- We need to be able to find important problems (especially bugs) quickly, but…
- *Issues* (things that make testing harder or slower) give bugs more time and more opportunity to hide
- We must recognize, identify, and resolve issues (and we might need help with that)
- We must be able to justify our answers when people ask “Why didn’t you find that bug?”

Testers must be credible and accountable.

Quick Testability Review

- Epistemic testability
  - What is the gap between what we know and what we need to know? Is there a real risk here?
- Value-related testability
  - What is the appropriate standard of correctness?
  - Should we be more concerned with risks other than this one?
- Intrinsic testability
  - Is there internal error detection and handling?
  - Does the product produce logs? Is there an API?
  - Is the product "unbuggy"?
- Subjective testability
  - Do we have a good understanding of the test space?
  - Do we have the skills we need to develop and apply useful tools?
- Project-related testability
  - Are we working in close collaboration with the programmers?
  - Could programmer checks and tests now be faster and cheaper than testing later?

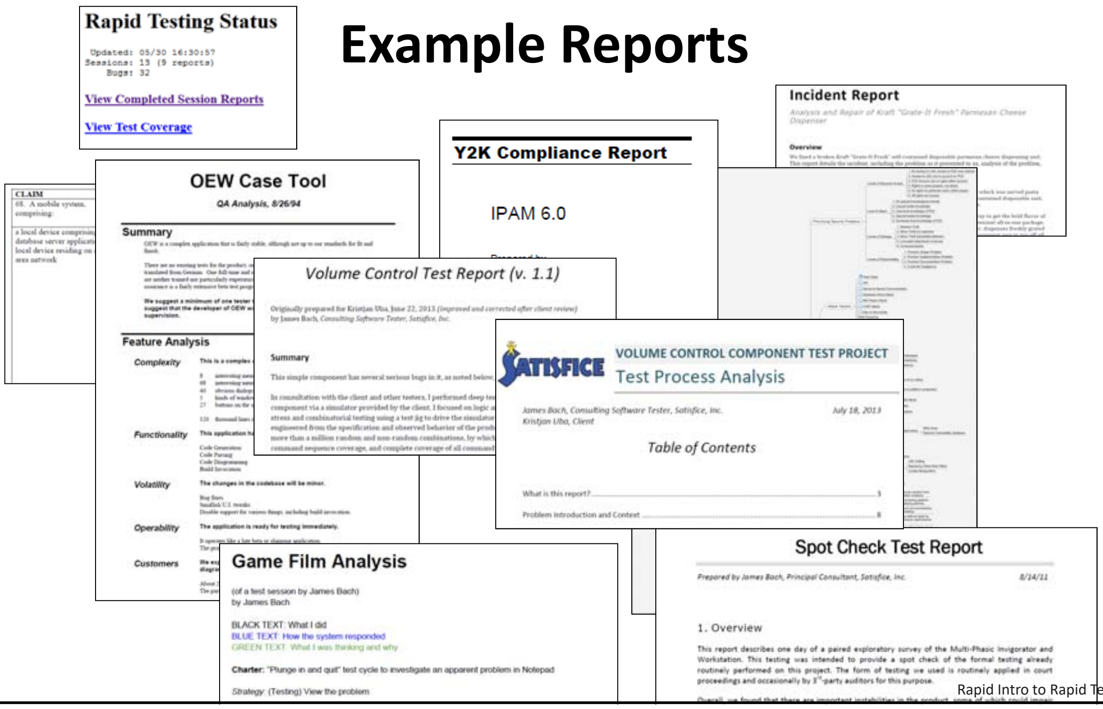
Mission check!

Issues?

Testing Inside Sessions Looks Like This

Testing (T)	Active test design; experimentation, interaction, learning about the product; increasing test coverage.
Bug (B)	Study and investigation of bugs; finding repro steps; looking for similar bugs inside a session. B–time interrupts T-time.
Setup (S)	Work within a session to prepare for testing, to support it, or to follow up on it. Setting up products, tools, environments; studying; analyzing non-bug behaviour… S-time interrupts T-time.
Opportunity	Work within a session that is NOT directed towards fulfilling the charter, but towards the general mission of testing. Chasing after a risk, helping other testers, testing while waiting for something else to happen…
Non-session	Meetings, lunches, breaks, chat, work-related or personal business done outside of a testing session.

Example Reports

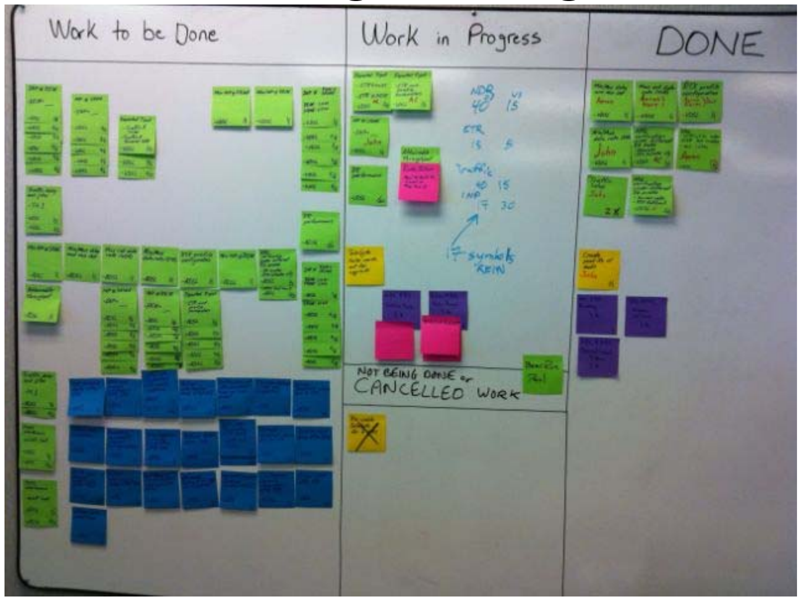


Reporting Considerations

- **Reporter safety:** What will they think if I made no progress?
- **Client:** Who am I reporting to and how do I relate to them?
- **Rules:** What rules and traditions are there for reporting here?
- **Significance of report:** How will my report influence events?
- **Subject of report:** On what am I reporting?
- **Other agents reporting:** How do other reports affect mine?
- **Medium:** How will my report be seen, heard, and touched?
- **Precision and confidence levels:** What distinctions make a difference?

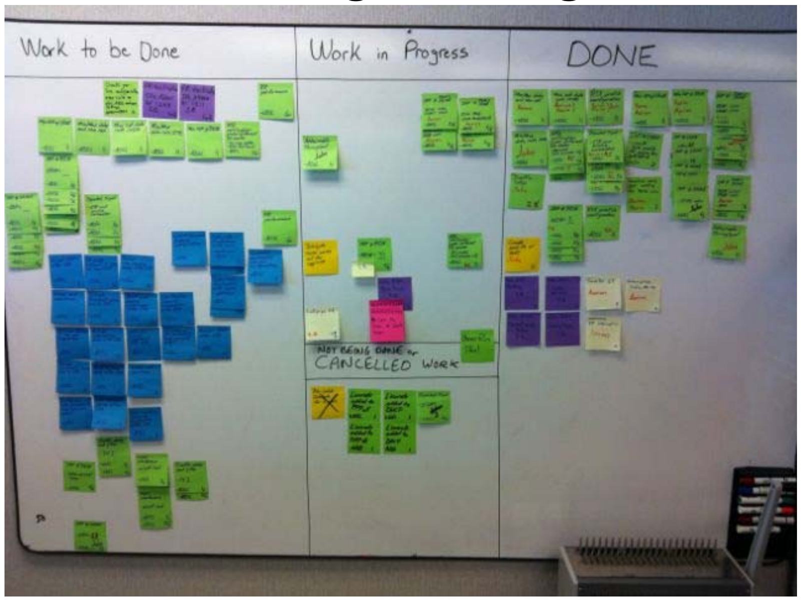
Take responsibility for the communication.

Visualizing Test Progress



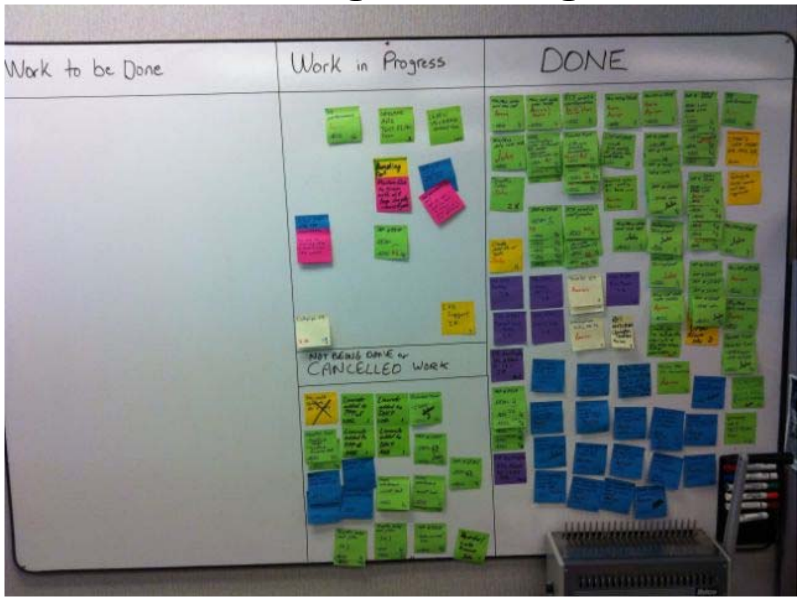
Rapid Intro to Rapid Testing - 115

Visualizing Test Progress



Rapid Intro to Rapid Testing - 116

Visualizing Test Progress



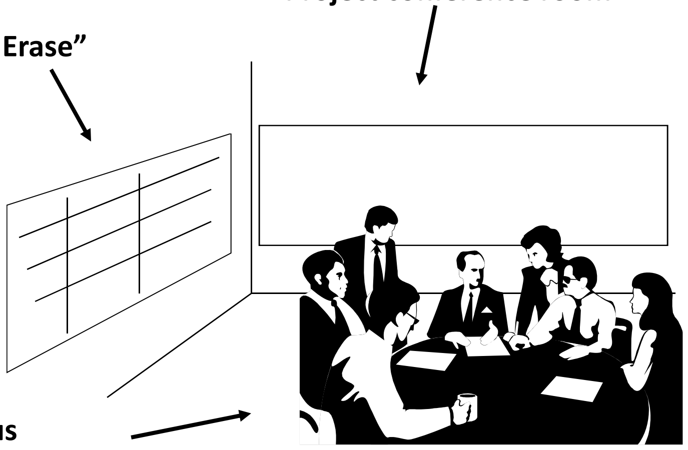
The Dashboard Concept

Large dedicated whiteboard

"Do Not Erase"

Project conference room

Project status meeting



Testing Dashboard

Updated 21/2 Build 38

Area	Effort	C	Q	Comments
File/edit	High	1	🙂	
View	Low	1+	😐	1345, 1363, 1401
Insert	Low	2	🙂	
Format	Low	2+	😐	automation broken
Tools	Blocked	1	☹	crashes bug 1407, 1423
Slideshow	Low	2	☹	animation memory leak
Online help	Blocked	0		new files not delivered
Clip art	Pause	1	😐	need help to test
Connectors	None	1	😐	need help to test
Install	Start 20/3	0		
Compatibility	Start 13/3	0		compatibility lab time scheduled
General GUI	Low	3	🙂	

Product Area

What are the major features or functions?

Area
file/edit
view
insert
format
tools
slideshow
online help
clipart
converters
install
compatibility
general GUI

- 15-30 areas (keep it simple)
- Avoid sub-areas: they're confusing.
- Areas should have roughly equal value.
- Areas together should be inclusive of everything reasonably testable.
- "Product areas" can include tasks or risks- but put them at the end.
- Minimize overlap between areas.
- Areas must "make sense" to your clients, or they won't use the board.

Test Effort

How much testing focus is each area getting right now?

None	Not testing; not planning to test.
Start	No testing yet, but expecting to start soon.
Low	Regression or spot testing only; maintaining coverage.
High	Focused testing effort; increasing coverage.
Pause	Temporarily ceased testing, though area is testable.
Blocked	Can't effectively test, due to blocking problem.
Ship	Going through final tests and wrap-up procedure.

Test Coverage

How much information do we have about each area so far?

0	No coverage	"We don't have any good information about this area."
1	Sanity check	Major functions & simple data. *Can this product work at all? Well enough to be tested?* "We're just getting to know this area."
1+	Surface scraped	More than sanity, but many functions not tested. "We're starting to get a handle on this area."
2	Core functions	All functions touched; common & critical tests executed. *Can this product work in ideal or ordinary conditions?* "We're understanding plenty of risks and coverage ideas."
2+	Increasing	Some data, state, or error coverage beyond level 2. "We're getting a good handle on this area, and we've used lots of techniques and coverage models, including…"
3	Complex cases	Deep data, state, error, or stress testing. SFDIPOT elements well covered. *Will this product work under realistic or extreme usage?* "If there were a serious problem in this area, we'd very likely know about it."

Quality Assessment

Does management see threats to the ship date?

🙂	**"We know of no problems in this area that threaten to stop on-time shipment or interrupt testing, nor do we have any definite suspicions about any."**
😐	**"We know of problems that are possible showstoppers, or we suspect that there could be important problems not yet discovered."**
☹️	**"We're aware of problems in this area that definitely threaten the release schedule or interrupt testing."**

Comments

Use the comment field to explain anything colored red, or any non-green quality indicator.

- Problem ID numbers
- Reasons for pausing, or delayed start
- Nature of blocking problems
- Why area is unstaffed

Using the Dashboard

- **Updates:** 2-5/week, or at each build, or prior to each project meeting.
- **Progress:** Set expectation about the duration of the “Testing Clock” and how new builds reset it.
- **Justification:** Be ready to justify the contents of any cell in the dashboard. The authority of the board depends upon meaningful, actionable content.
- **Going High Tech:** Sure, you can put this on the web, but will anyone actually look at it? A big visible chart gets attention without being asked.

Wrapping Up

Themes

- Take control of your own work.
- Stop doing things that aren't helping.
- Embrace exploration and experimentation.
- Focus on product risk. One size of testing does not fit all.
- Use lightweight, flexible heuristics to guide your work.
- Use the most concise form of documentation that solves the problem.
- Use tools to speed up the work.
- Explain your testing and its value.
- Grow your skills so that you can do all of the above.

Making Testing More Deep, Valuable, Engaged

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?
Executing test cases	Performing experiments
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***

Technical Suggestions

- Resist test cases, scripts, and overstructured models of testing; focus on test activities and the testing story.
- Let risk guide testing activities.
- Test in short, uninterrupted sessions; review and discuss them; seek and provide feedback.
- Avoid premature, excessive formalization.
- Keep documentation concise.
- Use recording tools like an airplane "black box".
- Emphasize exploratory scenario testing.
- Give testers lots of support for tools and learning about them, but don't let tools dominate the discussion. Generally prefer lightweight tools.

Tools?

- DON'T use them to "do" the testing. Tools don't do testing; PEOPLE do.
- DON'T become fixated on tools.
- DO use them to **support** testing.
  - setup and configuration management
  - data generation
  - probing the product
  - visualization
  - logging and recording
  - automated checking (most efficiently at the unit and integration levels; not so much at the GUI)
- Remember that tools amplify whatever we are. If we're excellent testers, tools will extend our excellence. If we're lousy testers, tools will allow us to do bad testing faster and worse than ever.

Social Suggestions

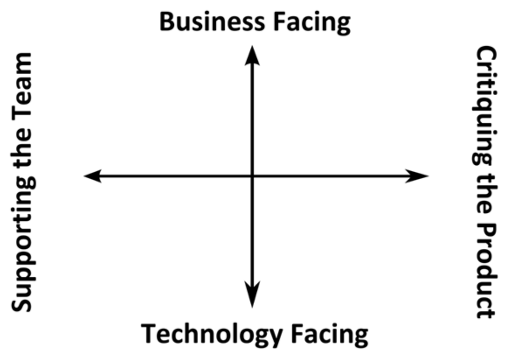
- Practice explaining testing.
- Declare your role and commitments.
- Don’t accept responsibility for the quality of the product.
- Embed yourself (or your testers) with the development team.
- Ask for testability.
- Watch where time and effort are going.
- Note the advantages of developer testing.
- Resist bureaucracy.
- Be a service to the project, not an obstacle.

Summing Up: Themes of Rapid Testing

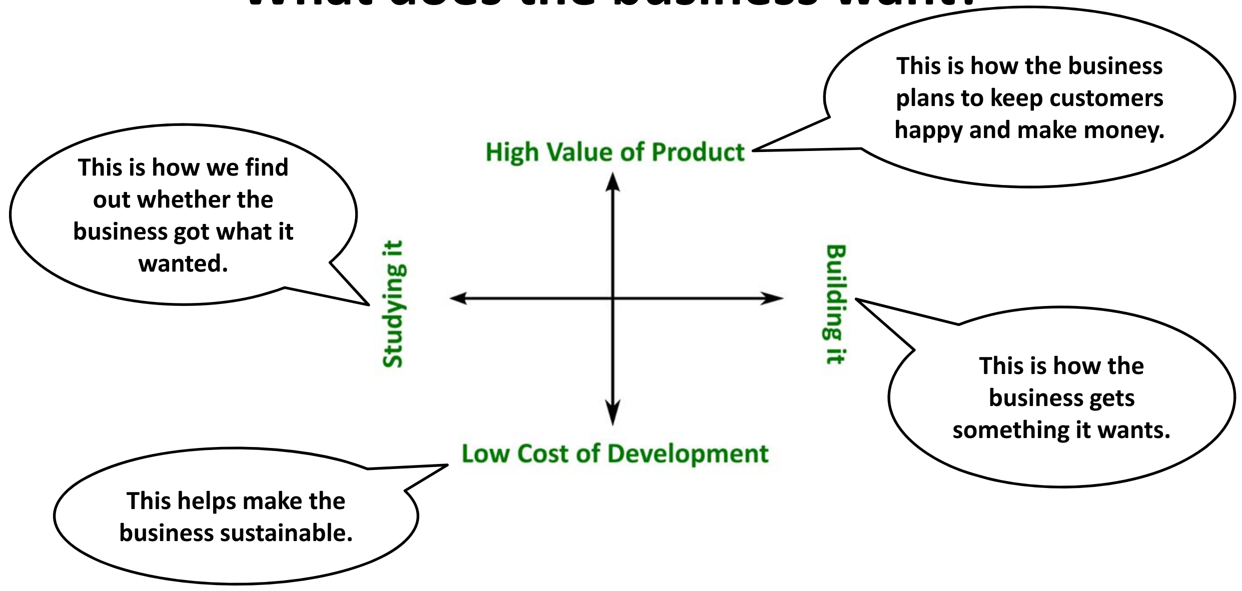
- Put the **tester's mind** at the center of testing.
- Learn to **deal with complexity** and ambiguity.
- Learn to **tell a compelling testing story.**
- Develop **testing skills** through practice, not just talk.
- **Use heuristics** to guide and structure your process.
- Replace “check for…” with **“look for problems in…”**
- **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.

Appendix: Rapid Testing in Agile Contexts

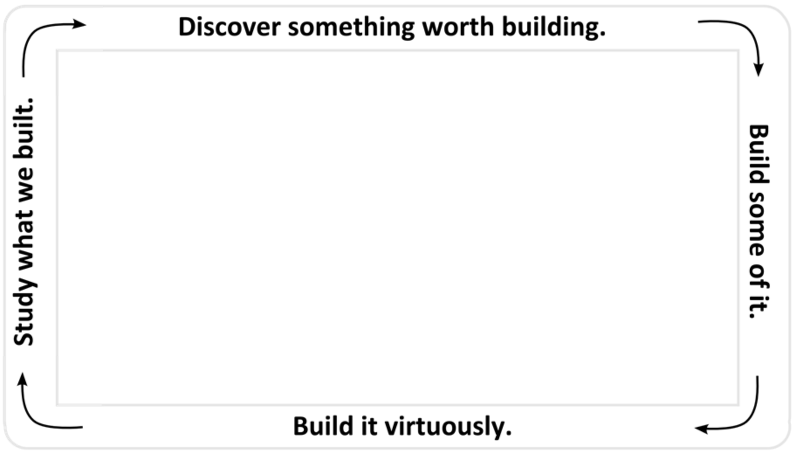
Dimensions of Crispin & Gregory's "Agile Testing Quadrants", based on Marick



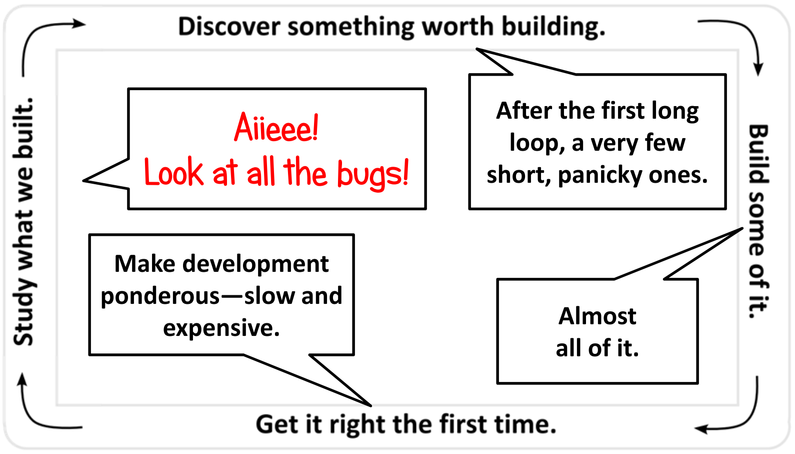
What does the business want?


This is how the business plans to keep customers happy and make money.
High Value of Product
This is how we find out whether the business got what it wanted.
Studying it
Building it
This is how the business gets something it wants.
Low Cost of Development
This helps make the business sustainable.

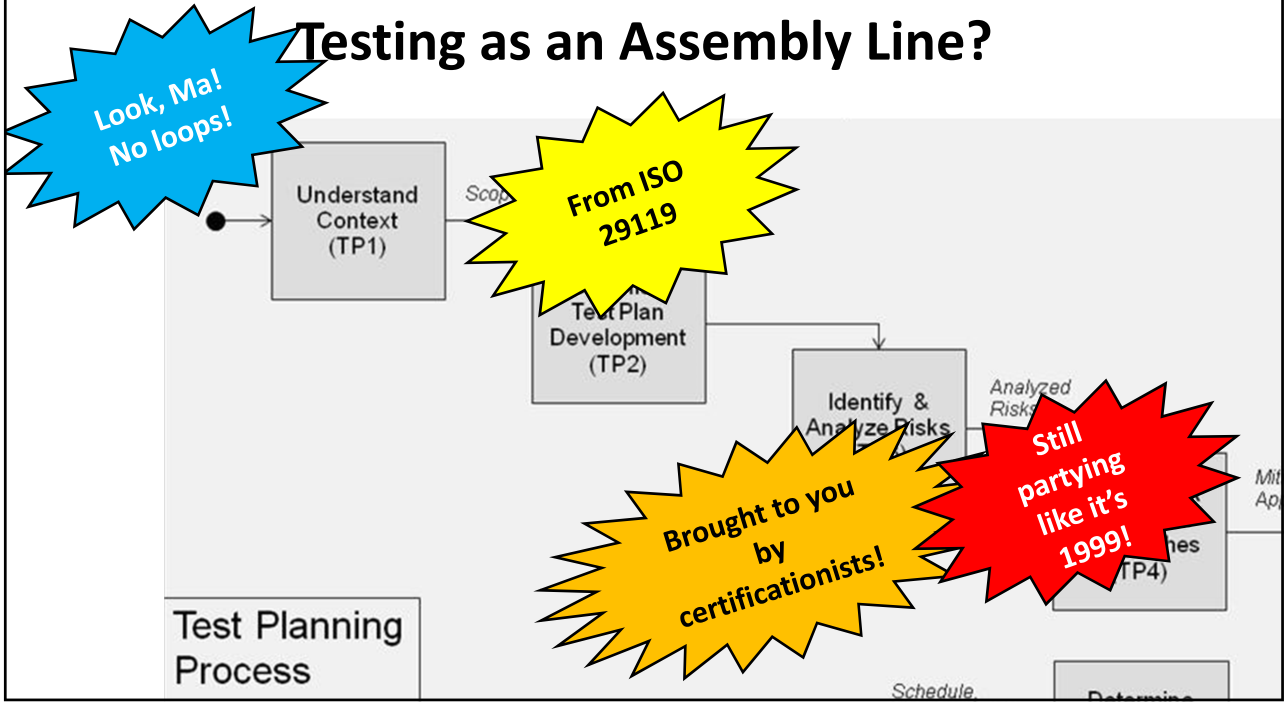
In the Beginning… the Universal Development Cycle…



"Traditional" Development Cycle


Discover something worth building.
Build some of it.
Get it right the first time.
Study what we built.
Aiieee! Look at all the bugs!
After the first long loop, a very few short, panicky ones.
Make development ponderous—slow and expensive.
Almost all of it.

Testing as an Assembly Line?



Then Came
Agile Software Development

Huzzah!

Manifesto for Agile Software Development

We are uncovering better ways of developing software
by doing it and helping others do it.

Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Uncovering is right! These things got covered up over 30 years!

That is, while there is value in the items on the right,
we value the items on the left more.

http://www.agilemanifesto.org

Principles of Agile Software Development

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Principles of Agile Software Development

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Two Cheers for Agile Software Development!

Agile Software Development was possibly the most humanist approach to software development in at least 30 years…

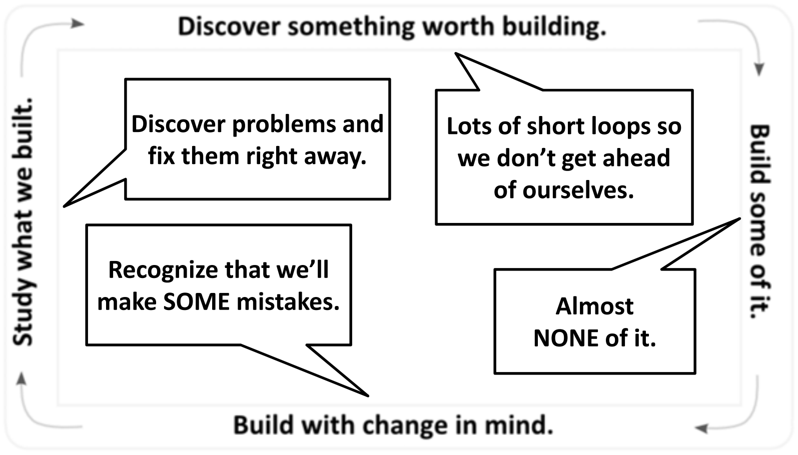
And then (almost immediately) came…

- tribes focused on code craft, team dynamics, process dynamics
- marketers and certifiers
- confusion about testing
- confusion about tests
- confusion about agility

Some Problems With “Agile” Software Development

- Agile’s earliest roots are in eXtreme Programming (XP), which was *extremely* focused on *programmers*. **This was a feature.** Developers were refocusing on their responsibility for the quality of the code.
- There were side effects. In many places, “Agile testing” became dominated by programmer testing, functional correctness, “definition of done”. **This was a bug.**
- Why? Because, in many places, *testing the product* became confused with *checking the code and the build*...
- …yet there can be *many problems* in the relationships between the product and the people you use it; not just problems with the code.
- Until we’ve gained human experience with the product, we don’t know so much where those problems are. and that’s where risk lives.

Agile Development Cycle

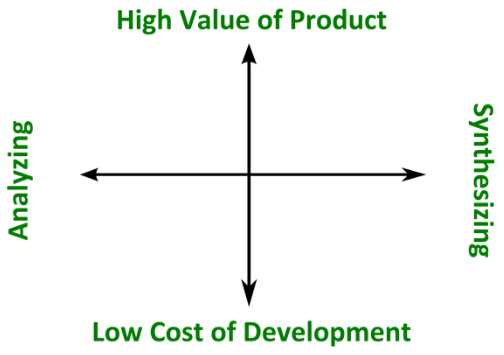

Discover something worth building.
Build some of it.
Build with change in mind.
Study what we built.
Discover problems and fix them right away.
Lots of short loops so we don't get ahead of ourselves.
Recognize that we'll make SOME mistakes.
Almost NONE of it.

What does it really mean to do "Agile Development"?

- Deliver often (so the product can be evaluated)
- Collaborate across roles (take advantage of diversity)
- Develop craftsmanship (focus on excellence and skill)
- Don't be too formal (don't do wasteful things)
- Be prepared to try things, to fail, and learn from it all
- Build and use tools expertly
- **Seek a sustainable pace**

We all serve the business to satisfy the customer. So, refactor "business-facing" and "customer-facing"…

"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software"

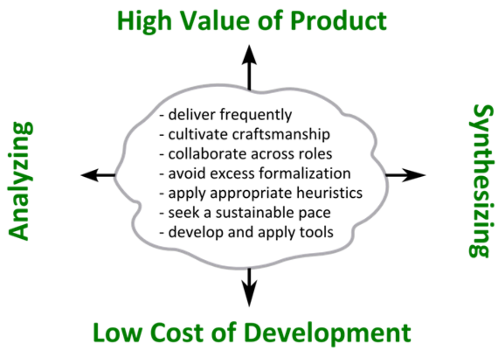


"Continuous attention to technical excellence and good design enhances agility."

(This version avoids displacing testing specialists and more directly addresses the tension between business and technology "facings.")

With those strategic goals, remind ourselves of the core tactics of Agile Software Development…

"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software"



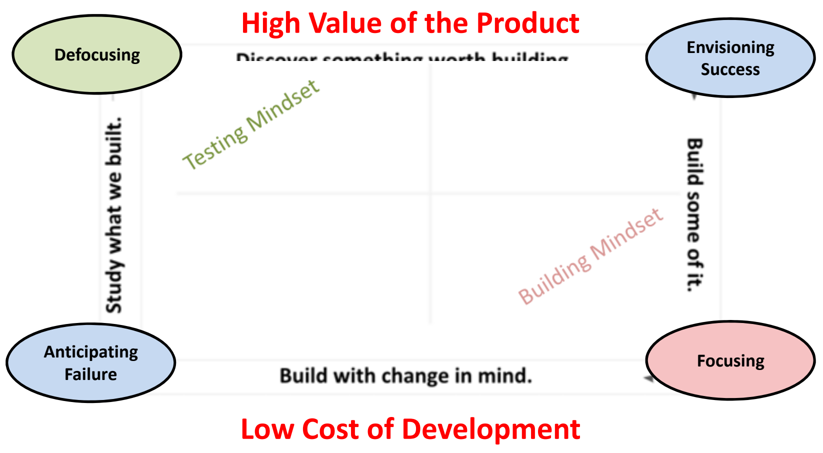
- deliver frequently
- cultivate craftsmanship
- collaborate across roles
- avoid excess formalization
- apply appropriate heuristics
- seek a sustainable pace
- develop and apply tools

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

(These are the core tactics as we see them. You may prefer a slightly different list.)

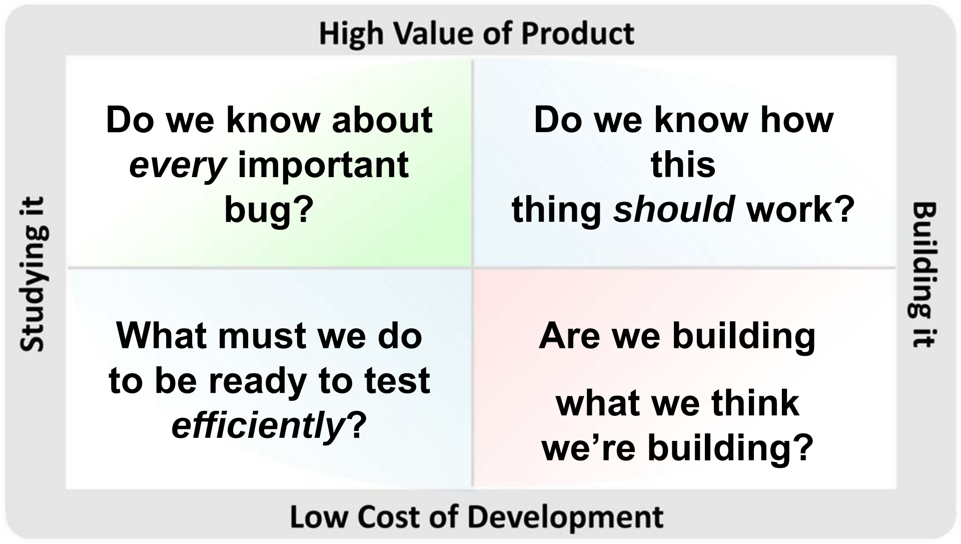
The Agile Development Cycle

"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software"

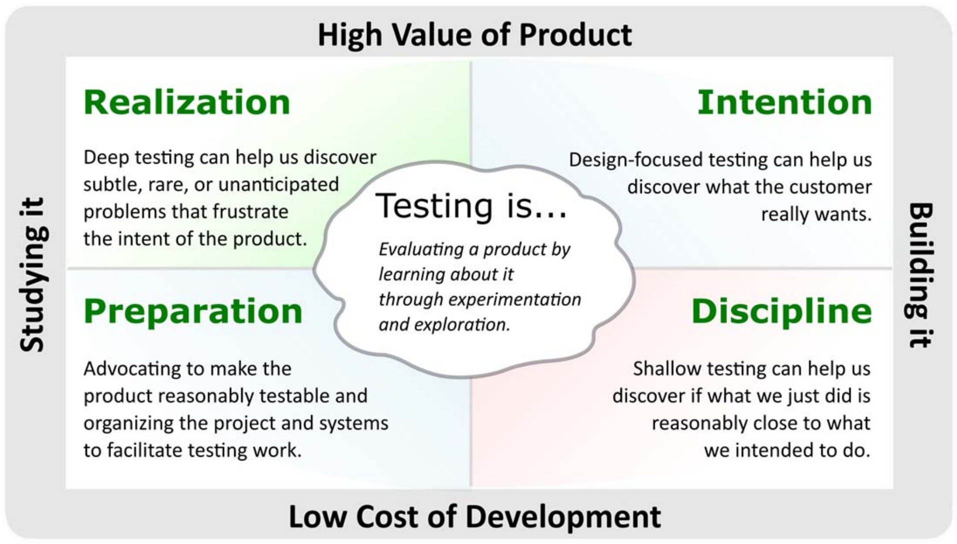


"Continuous attention to technical excellence and good design enhances agility."

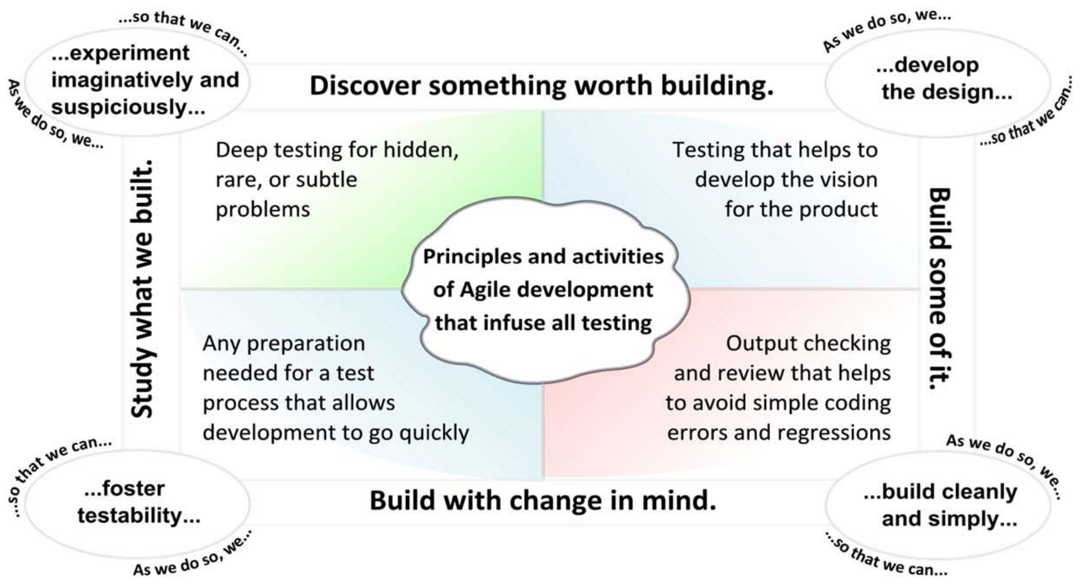
Four Testing Questions


High Value of Product
Do we know about *every* important bug?
Do we know how this thing *should* work?
What must we do to be ready to test *efficiently*?
Are we building what we think we're building?
Studying it
Building it
Low Cost of Development

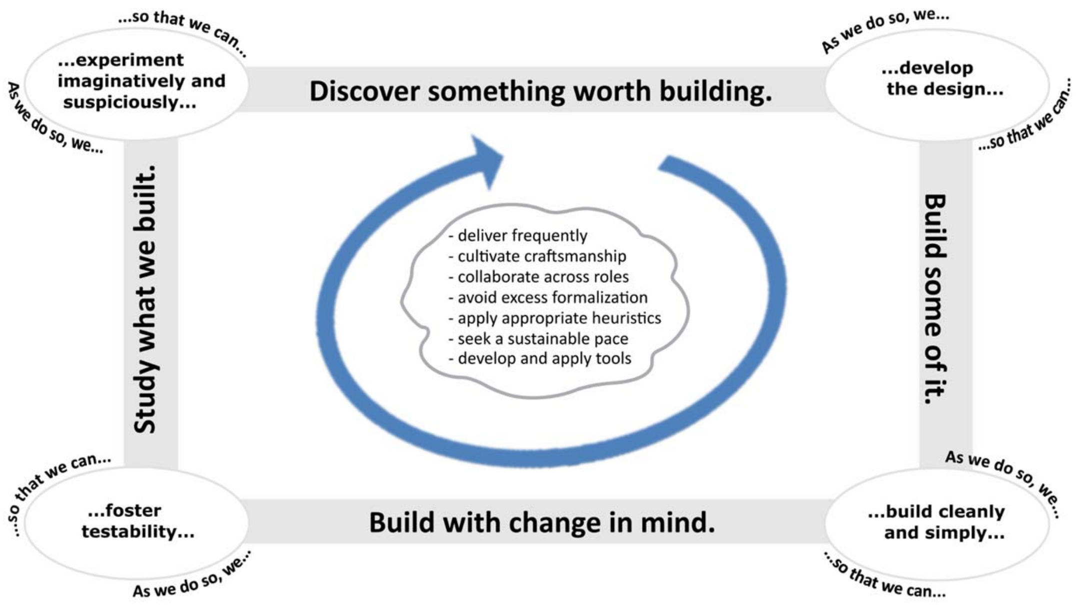
Four Frames for Testing


High Value of Product
Realization
Deep testing can help us discover subtle, rare, or unanticipated problems that frustrate the intent of the product.
Intention
Design-focused testing can help us discover what the customer really wants.
Testing is...
Evaluating a product by learning about it through experimentation and exploration.
Preparation
Advocating to make the product reasonably testable and organizing the project and systems to facilitate testing work.
Discipline
Shallow testing can help us discover if what we just did is reasonably close to what we intended to do.
Studying it
Building it
Low Cost of Development

Testing Is *Woven into Development*

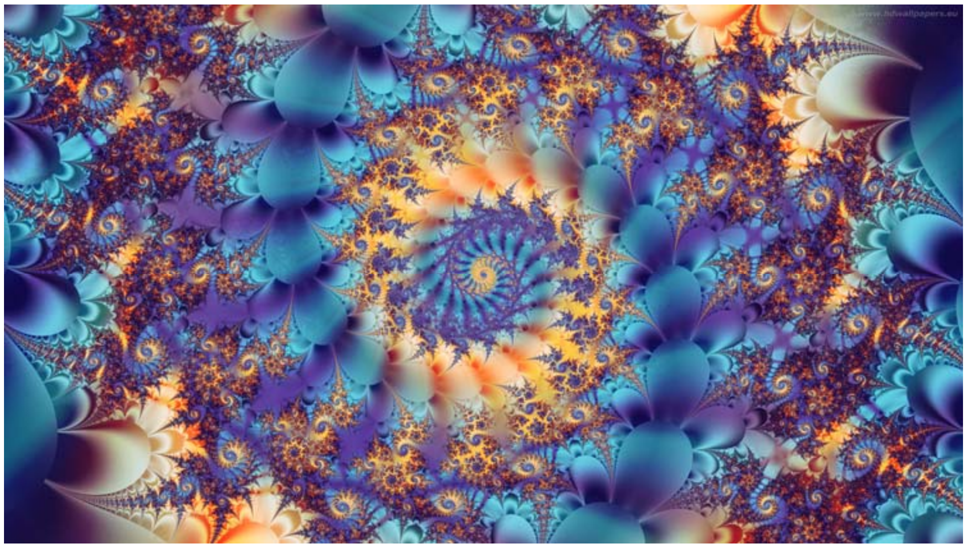

...so that we can...
...experiment imaginatively and suspiciously...
As we do so, we...
Discover something worth building.
As we do so, we...
...develop the design...
...so that we can...
Deep testing for hidden, rare, or subtle problems
Testing that helps to develop the vision for the product
Principles and activities of Agile development that infuse all testing
Any preparation needed for a test process that allows development to go quickly
Output checking and review that helps to avoid simple coding errors and regressions
Study what we built.
Build some of it.
...so that we can...
...foster testability...
As we do so, we...
Build with change in mind.
As we do so, we...
...build cleanly and simply...
...so that we can...

And although these dimensions have a roughly clockwise sequence…

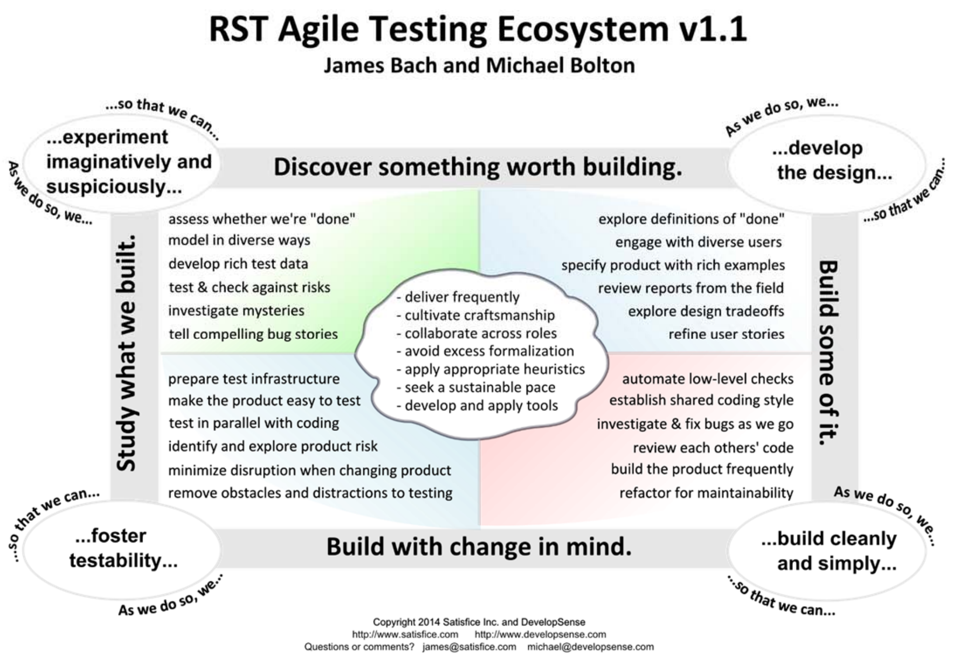


Discover something worth building.
As we do so, we…
…develop the design…
…so that we can…
Build some of it.
As we do so, we…
…build cleanly and simply…
…so that we can…
Build with change in mind.
As we do so, we…
…foster testability…
…so that we can…
Study what we built.
As we do so, we…
…experiment imaginatively and suspiciously…
…so that we can…
- deliver frequently
- cultivate craftsmanship
- collaborate across roles
- avoid excess formalization
- apply appropriate heuristics
- seek a sustainable pace
- develop and apply tools

…development isn't linear. Not even just loopy.



Development is a fractal!


RST Agile Testing Ecosystem v1.1
James Bach and Michael Bolton
...so that we can...
...experiment imaginatively and suspiciously...
As we do so, we...
Discover something worth building.
As we do so, we...
...develop the design...
...so that we can...
Study what we built.
Build some of it.
assess whether we're "done"
model in diverse ways
develop rich test data
test & check against risks
investigate mysteries
tell compelling bug stories
explore definitions of "done"
engage with diverse users
specify product with rich examples
review reports from the field
explore design tradeoffs
refine user stories
- deliver frequently
- cultivate craftsmanship
- collaborate across roles
- avoid excess formalization
- apply appropriate heuristics
- seek a sustainable pace
- develop and apply tools
prepare test infrastructure
make the product easy to test
test in parallel with coding
identify and explore product risk
minimize disruption when changing product
remove obstacles and distractions to testing
automate low-level checks
establish shared coding style
investigate & fix bugs as we go
review each others' code
build the product frequently
refactor for maintainability
...so that we can...
...foster testability...
As we do so, we...
Build with change in mind.
As we do so, we...
...build cleanly and simply...
...so that we can...
Copyright 2014 Satisfice Inc. and DevelopSense
http://www.satisfice.com http://www.developsense.com
Questions or comments? james@satisfice.com michael@developsense.com

Rapid Intro to Rapid Testing - 155

A Word from Our Sponsor (Me)

- Rapid Software Testing is a course, a mind-set, and a skill set about how to do excellent software testing in a way that is very fast, inexpensive, credible, and accountable. I co-author RST with James Bach.
- I teach RST in classes for testers, developers, managers, business analysts, documenters, DevOps people, tech support…
- I also offer advice and consulting on testing and development to managers and executives.

http://www.developsense.com +1 416 992 8378

Rapid Intro to Rapid Testing - 156