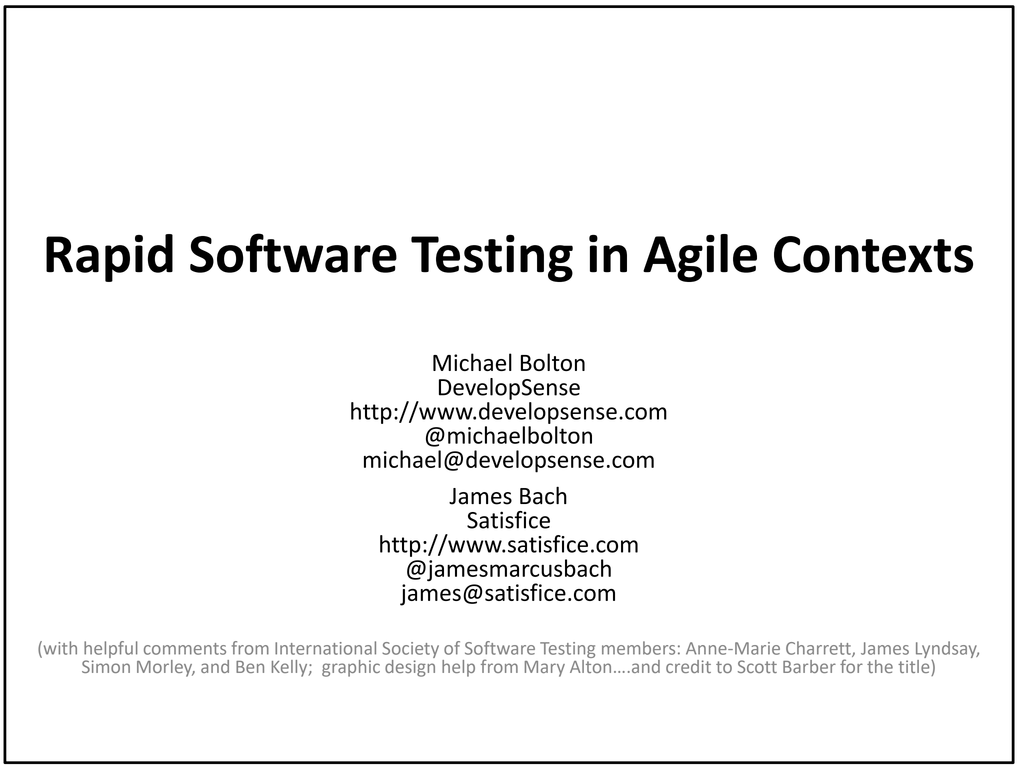


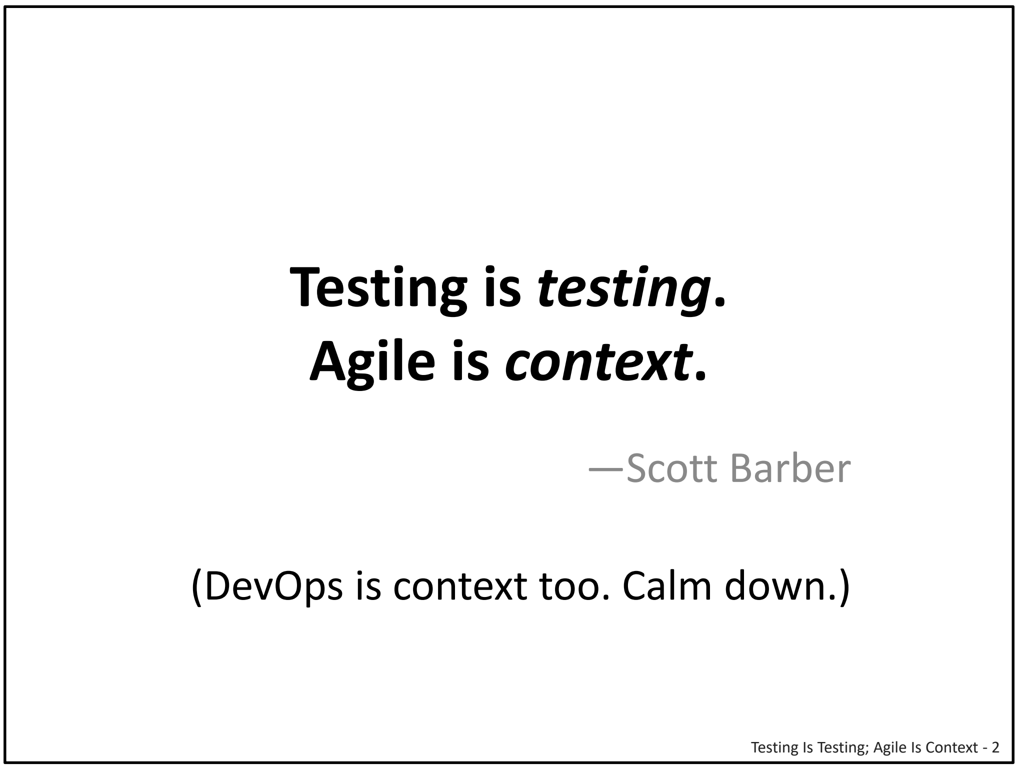
Rapid Software Testing in Agile Contexts

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

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

(with helpful comments from International Society of Software Testing members: Anne-Marie Charrett, James Lyndsay, Simon Morley, and Ben Kelly; graphic design help from Mary Alton….and credit to Scott Barber for the title)

A *very* brief word from our sponsor: Need help with training or consulting in software testing in Agile or other contexts? I help people to solve development and testing problems they didn't know they could solve. Please contact me if you need help. And now, on with the presentation.



Testing is *testing*.
Agile is *context*.

—Scott Barber

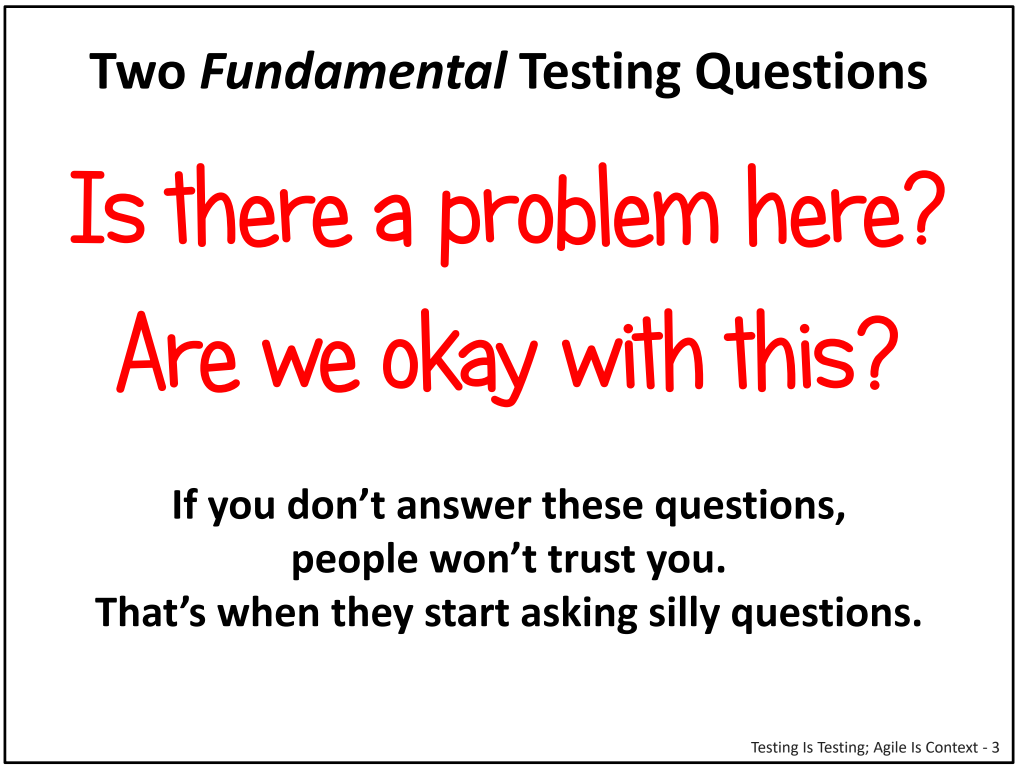
(DevOps is context too. Calm down.)

Testing, as we'll see later in more detail, is *evaluating a product by learning about it through exploration and experimentation.* Testing has always been about that.

Agile software development got its name in the early years of this century as a group of developers met to discuss ways in which they approached software development as a craft. After they declared the Manifesto for Agile Software Development, many different approaches, heuristics, tools, and techniques came to prominence. Excellent testing adapts to all of those different context factors.

DevOps means, at its centre, "development and operations people working together to accomplish the goals of the business." That's a fine thing — but how badly have we lost our way when that sounds like a radical, innovative idea?

In any case, testing is always testing — the process of evaluating something by learning about it through exploration and experimentation. Testing deepens our understanding of things by our interactions with them. Testing might include occasional bursts of confirming things that we know, but serious testing involves *challenging* what we know.



Two *Fundamental* Testing Questions

Is there a problem here?

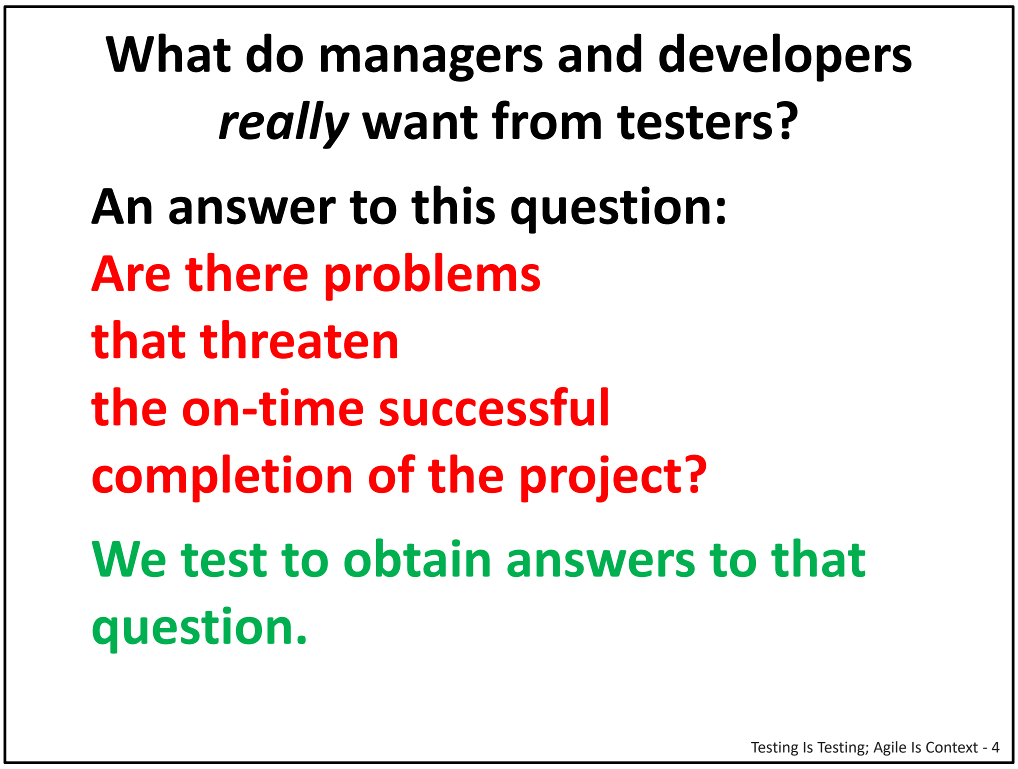
Are we okay with this?

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

As a tester, looking down at what I'm testing, I'm focused on finding problems that threaten the value of the product. I'm continually asking myself, "Is there a problem here?" You can read more about that:

http://www.developsense.com/blog/2009/09/pass-vs-fail-vs-is-there-problem-here/
http://www.developsense.com/blog/2007/04/big-questions-of-testing/

As I do that, since I'm always working on things that are to some degree new, I'm encountering problems and obstacles. I might not have access to important information; the developers might not be available; the product might be in rough shape, and investigating and reporting the bugs might take time away from deeper testing. Testability might be lacking. My equipment and tools might not be up to the task at hand. I'm also imagining looking up over my shoulder at my client and the other stakeholders, who are looking down and seeing those stumbling blocks, not only in the testing but in the overall project. So I'm also asking everyone concerned, "Are we okay with this?"



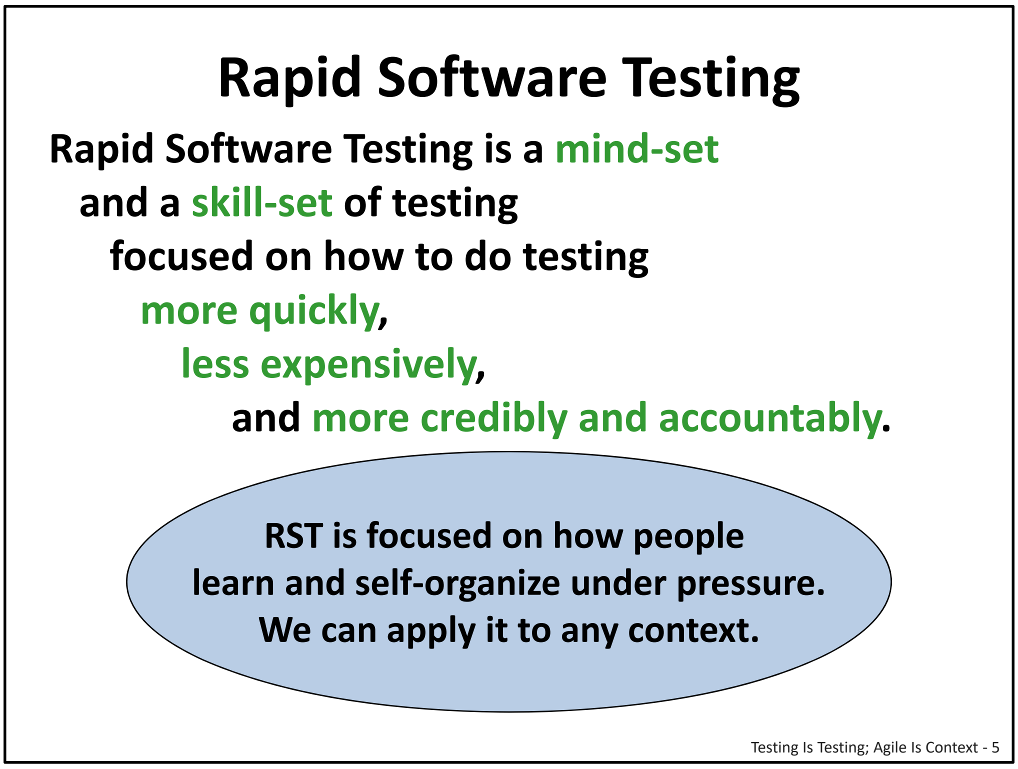
What do managers and developers *really* want from testers?

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.

I haven't always been a tester. As a former developer, and a former project manager for a best-selling piece of software, I can assure you": almost everything that we're asked to do as testers centres around this fundamental question: Are there problems that threaten the on-time, successful completion of the project? Note that I'm using "project" in a very expansive sense: the project might be to release a product, to update a web site, to develop a new feature, to modify a line of code, or to sustain a business… anything that people are doing for which they ask our help.0



Rapid Software Testing

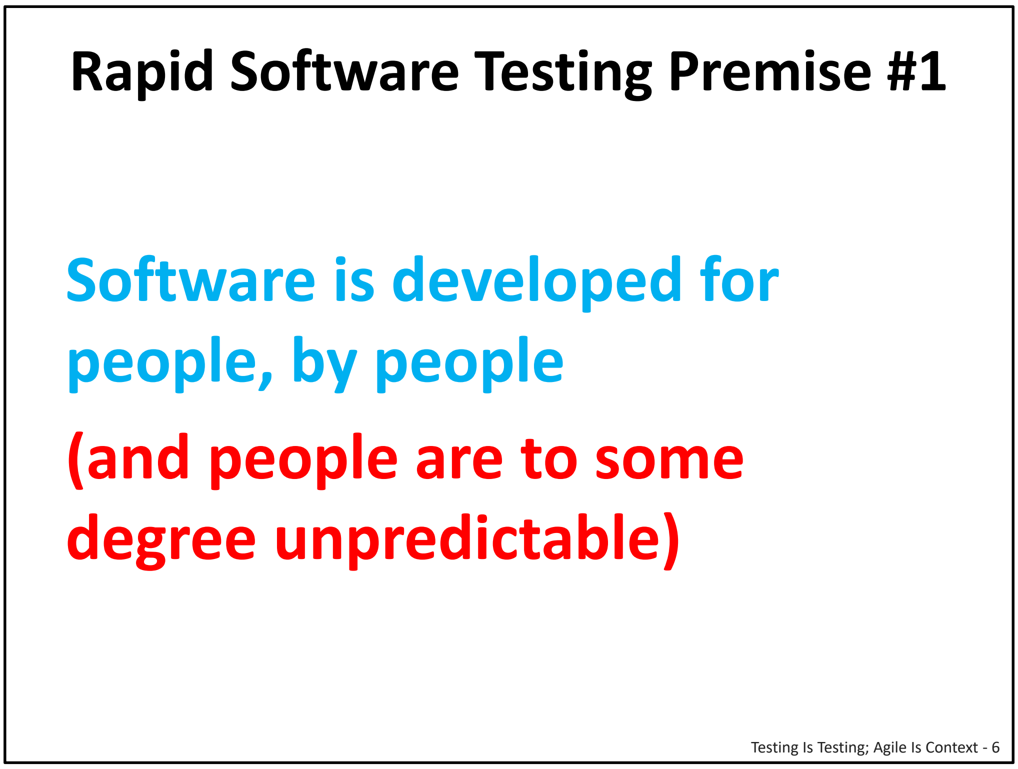
Rapid Software Testing is a mind-set and a skill-set of testing focused on how to do testing more quickly, less expensively, and more credibly and accountably.

RST is focused on how people learn and self-organize under pressure. We can apply it to any context.

Rapid Software Testing is a mind set–a particular way of looking at the world; and a skill set—a particular set of things that we practice and get better at. Rapid Software Testing involves considerations of skill, personal integrity, improved focus on the underlying need for testing tasks, improved appreciation for the stakeholders of testing tasks, and knowledge of the possible techniques and tools that could be brought to bear to improve efficiency.

Rapid Software Testing is intended to be a context-driven approach to software testing. The methodology is designed to adapt to any kind of testing context. We've applied these approaches in financial institutions, in court cases, in testing of medical devices, in commercial shrink-wrapped software, in retail management, in insurance companies, to games… and our students have applied them in many other contexts as well.

Rapid Software Testing isn't centred around the development model, but around preparing the skills and the mindset of the tester to respond to the development model, whatever that might be. The development model of a project is an element of its context. Rapid Software Testing can be applied to traditional development, Waterfall, Scrummerfall, Moneyball, Agile, Fragile… because



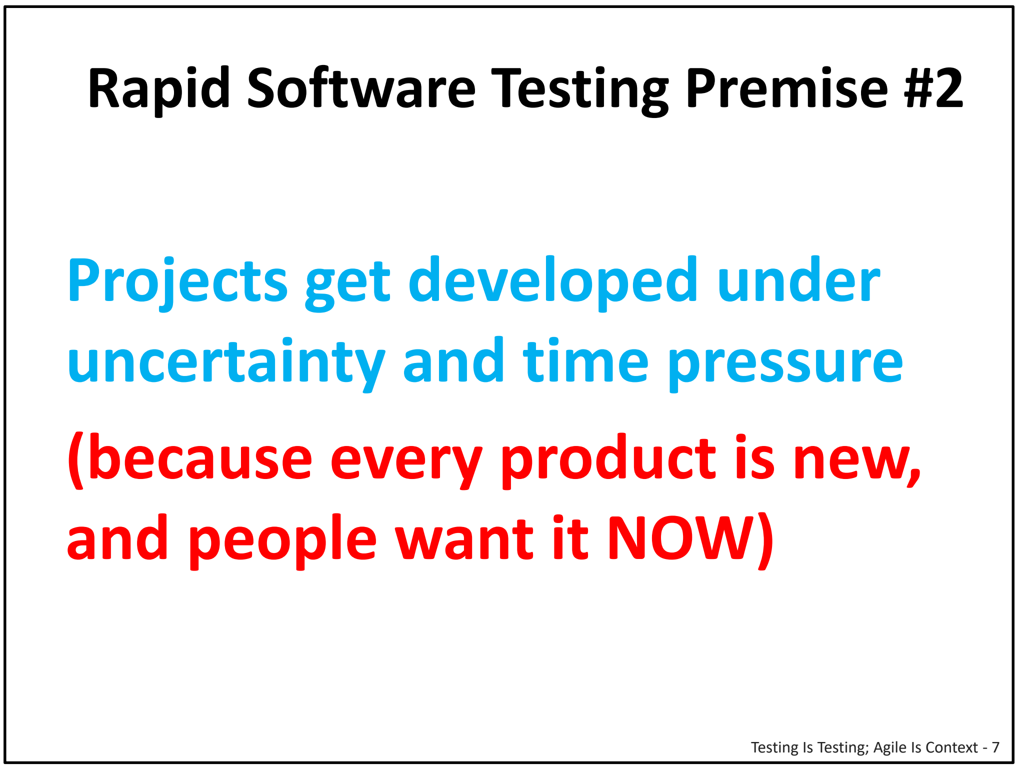
Rapid Software Testing Premise #1

Software is developed for people, by people

(and people are to some degree unpredictable)

Software projects and products are relationships between people, who are creatures both of emotion and rational thought. Yes, there are technical, physical, and logical elements as well, and those elements are very substantial. But software development is dominated by human aspects: politics, emotions, psychology, perception, and cognition. A project manager may declare that any given technical problem is not a problem at all for the business. Users may demand features they will never use. Your fabulous work may be rejected because the programmer doesn't like you. Sufficiently fast performance for a novice user may be unacceptable to an experienced user. Quality is always value to some person who matters. Product quality is a relationship between a product and people, never an attribute that can be isolated from a human context.

http://www.developsense.com/blog/2012/09/premises-of-rapid-software-testing-part-1/



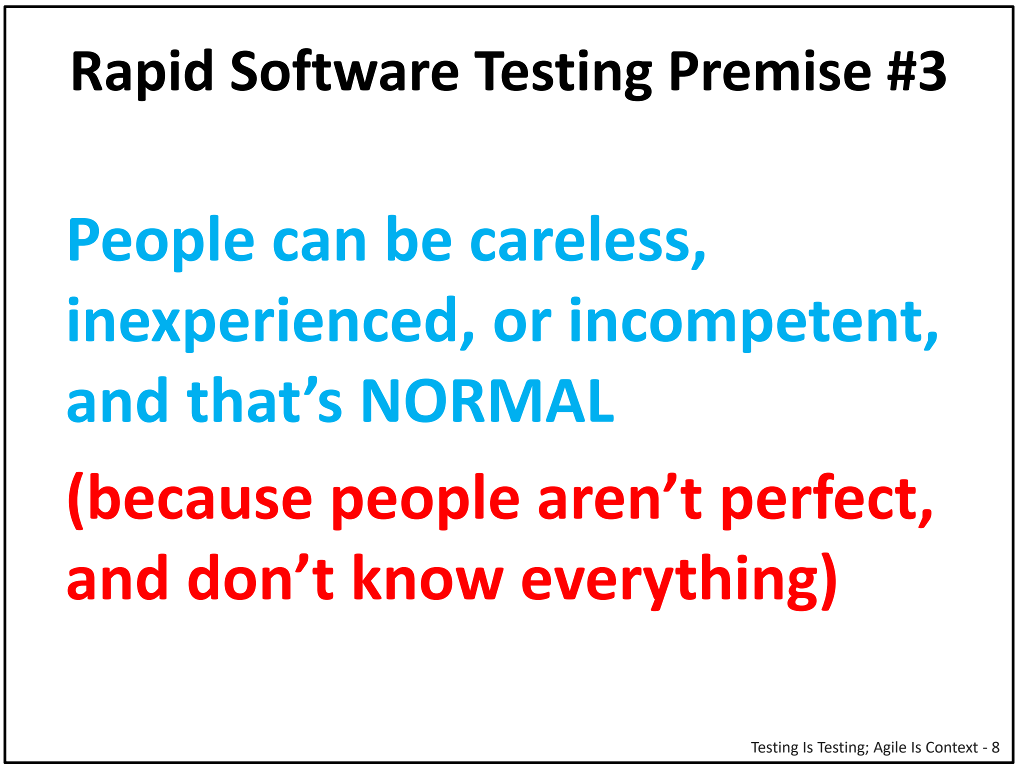
Rapid Software Testing Premise #2

Projects get developed under uncertainty and time pressure

(because every product is new, and people want it NOW)

Each project occurs under conditions of uncertainty and time pressure. Some degree of confusion, complexity, volatility, and urgency besets each project. The confusion may be crippling, the complexity overwhelming, the volatility shocking, and the urgency desperate. There are simple reasons for this: novelty, ambition, and economy. Every software project is an attempt to produce something new, in order to solve a problem. People in software development are eager to solve these problems. At the same time, they often try to do a whole lot more than they can comfortably do with the resources they have. This is not any kind of moral fault of humans. Rather, it's a consequence of the so-called "Red Queen" effect from evolutionary theory (the name for which comes from Through the Looking Glass): you must run as fast as you can just to stay in the same place. If your organization doesn't run with the risk, your competitors will—and eventually you will be working for them, or not working at all.

http://www.developsense.com/blog/2012/09/premises-of-rapid-software-testing-part-1/



Rapid Software Testing Premise #3

People can be careless, inexperienced, or incompetent, and that’s NORMAL

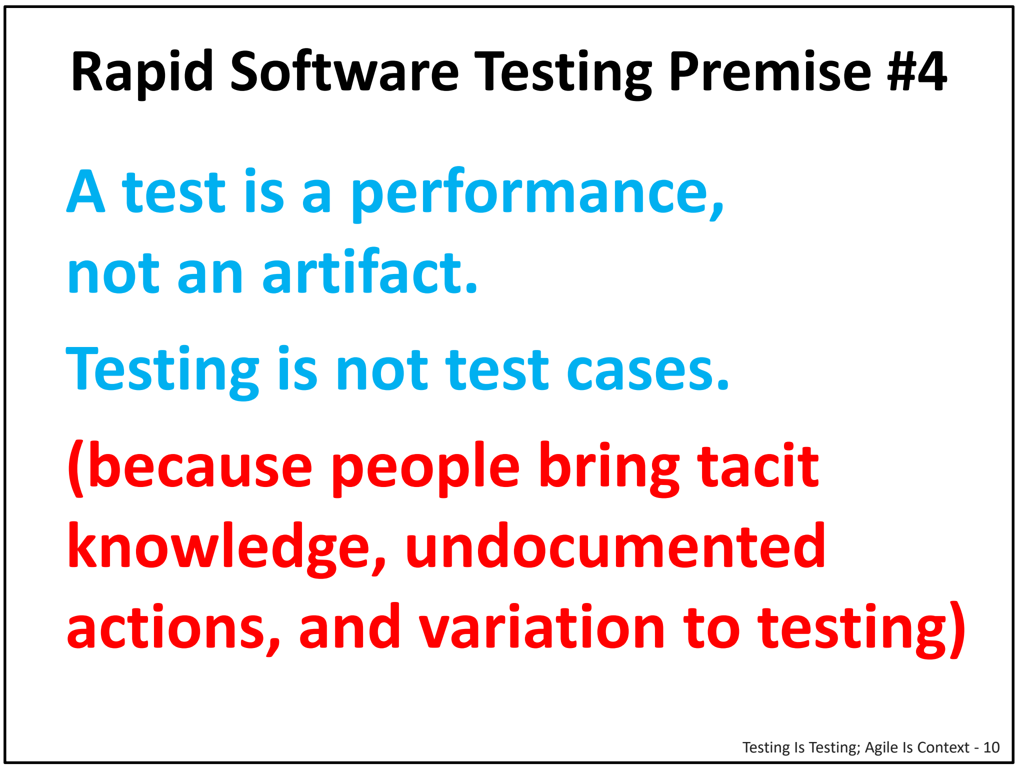
(because people aren’t perfect, and don’t know everything)

Despite our best hopes and intentions, some degree of inexperience, carelessness, and incompetence is normal. This premise is easy to verify. Start by taking an honest look at yourself. Do you have all of the knowledge and experience you need to work in an unfamiliar domain, or with an unfamiliar product? Have you ever made a spelling mistake that you didn’t catch? Which testing textbooks have you read carefully? How many academic papers have you pored over? Are you up to speed on set theory, graph theory, and combinatorics? Are you fluent in at least one programming language? Could you sit down right now and use a de Bruijn sequence to optimize your test data? Would you know when to *avoid* using it? Are you thoroughly familiar with all the technologies being used in the product you are testing? Probably not—*and that’s okay*. It is the nature of innovative software development work to stretch the limits of even the most competent people. Other testing and development methodologies seem to assume that everyone can and will do the right thing at the right time. We find that incredible. Any methodology that ignores human fallibility is a fantasy. By saying that human fallibility is normal, we’re not trying to defend it or apologize for it, but we are pointing out that we must expect to encounter it in ourselves and in others, to deal with it compassionately, and make the most of our opportunities to learn our craft and build our skills.

http://www.developsense.com/blog/2012/09/premises-of-rapid-software-testing-part-1/

So…

THERE IS RISK, AND THERE WILL BE BUGS



Rapid Software Testing Premise #4

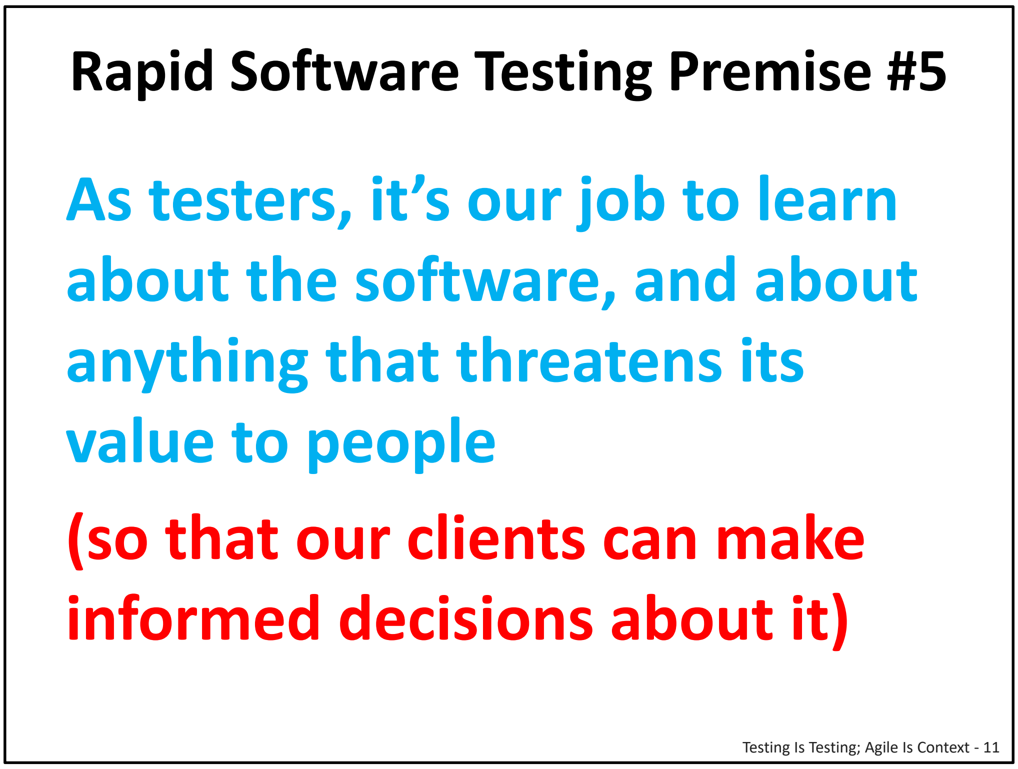
A test is a performance, not an artifact.

Testing is not test cases.

(because people bring tacit knowledge, undocumented actions, and variation to testing)

A test is an activity; it is a performance, not an artifact. Most testers will casually say that they "write tests" or that they "create test cases." That's fine, as far as it goes. That means they have conceived of ideas, data, procedures, and perhaps programs that automate some task or another; and they may have represented those ideas in writing or in program code. Trouble occurs when any of those things is confused with the ideas they represent, and when the representations become confused with actually testing the product. This is a fallacy called reification, the error of treating abstractions as though they were things. Until some tester engages with the product, observes it and interprets those observations, *no testing has occurred*. Even if you write a completely automatic checking process, the results of that process must be reviewed and interpreted by a responsible person.

http://www.developsense.com/blog/2012/09/premises-of-rapid-software-testing-part-2/



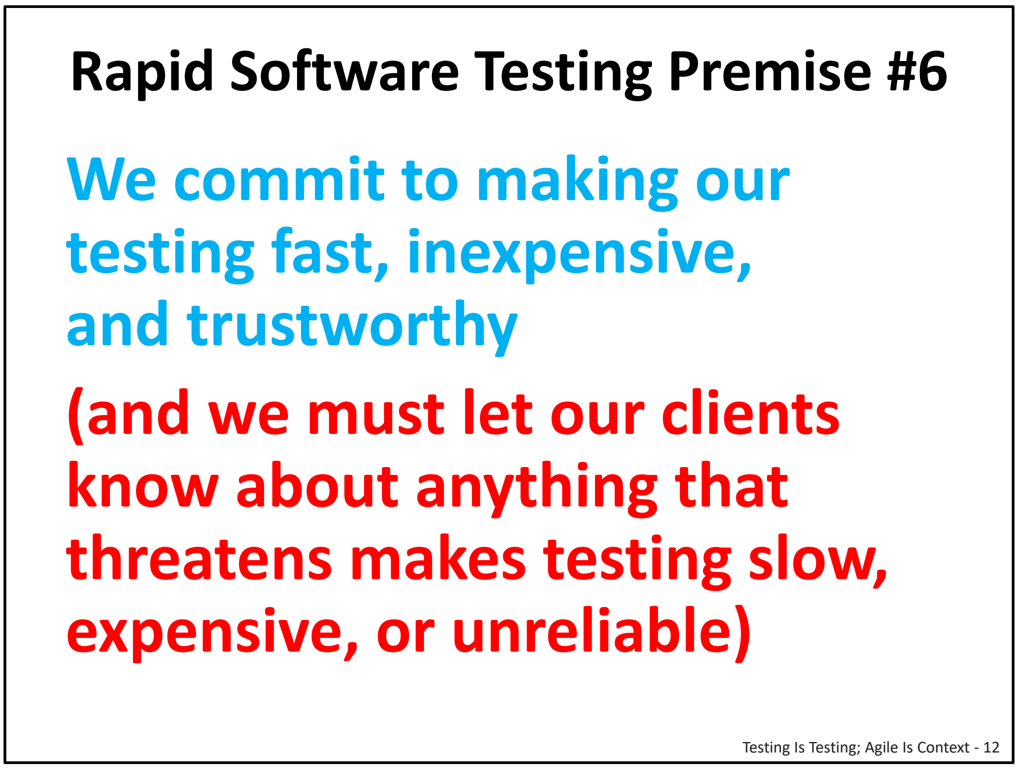
Rapid Software Testing Premise #5

As testers, it’s our job to learn about the software, and about anything that threatens its value to people

(so that our clients can make informed decisions about it)

Testing’s purpose is to discover the status of the product and any threats to its value, so that our clients can make informed decisions about it. There are people that have other purposes in mind when they use the word “test.” For some, testing may be a ritual of checking that basic functions appear to work. This is not our view. We are on the hunt for important problems. We seek a comprehensive understanding of the product. We do this in support of the needs of our clients, whoever they are. The level of testing necessary to serve our clients will vary. In some cases the testing will be more formal and simple, in other cases, informal and elaborate. In all cases, testers are suppliers of vital information about the product to those who must make decisions about it. **Testers light the way.**

http://www.developsense.com/blog/2012/09/premises-of-rapid-software-testing-part-2/



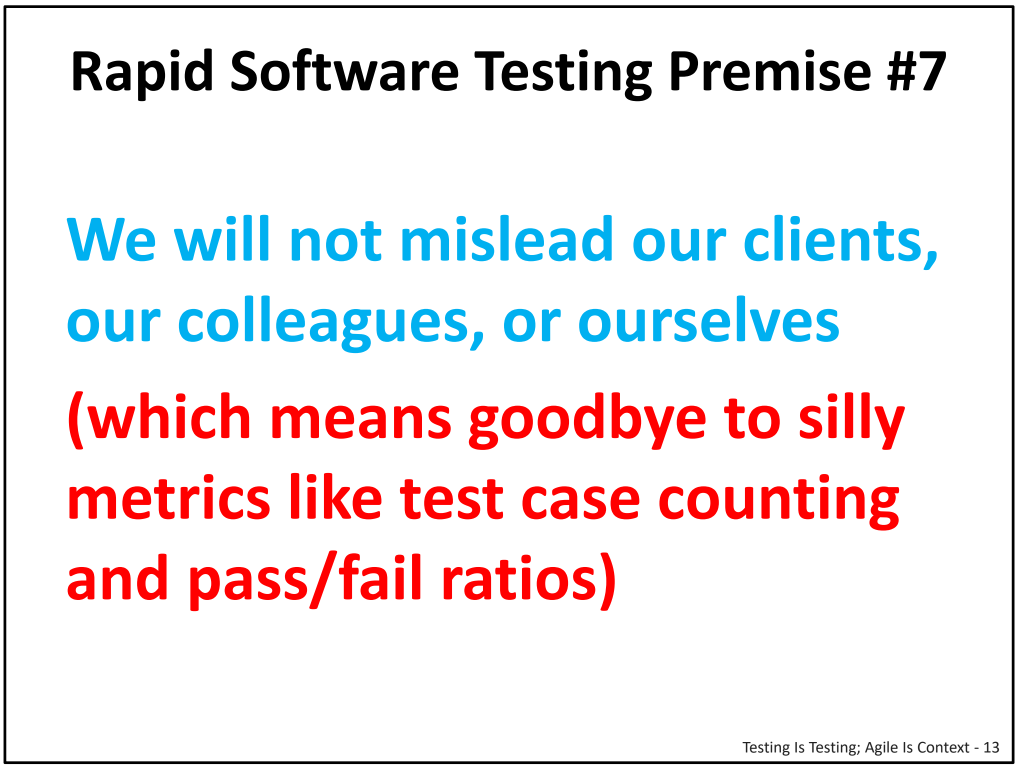
Rapid Software Testing Premise #6

We commit to making our testing fast, inexpensive, and trustworthy

(and we must let our clients know about anything that threatens makes testing slow, expensive, or unreliable)

We commit to performing credible, cost-effective testing, and we will inform our clients of anything that threatens that commitment. Rapid Testing seeks the fastest, least expensive testing that completely fulfills the mission of testing. We should not suggest million dollar testing when ten dollar testing will do the job. It's not enough that we test well; we must test well given the limitations of the project. Furthermore, when we are under constraints that may prevent us from doing a good job, testers must work with the client to resolve those problems. Whatever we do, we must be ready to justify and explain it.

http://www.developsense.com/blog/2012/09/premises-of-rapid-software-testing-part-3/



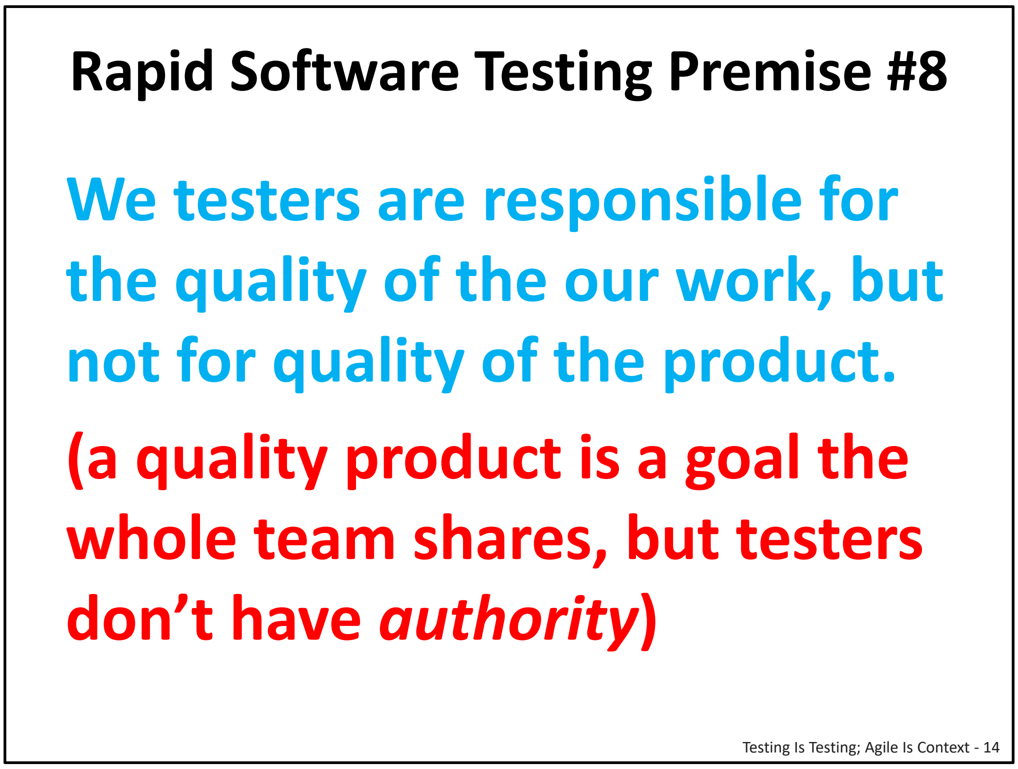
Rapid Software Testing Premise #7

We will not mislead our clients, our colleagues, or ourselves

(which means goodbye to silly metrics like test case counting and pass/fail ratios)

We will not knowingly or negligently mislead our clients and colleagues. This ethical premise drives a lot of the structure of Rapid Software Testing. Testers are frequently the target of well-meaning but unreasonable or ignorant requests by their clients. We may be asked to suppress bad news, to create test documentation that we have no intention of using, or to produce invalid metrics to measure progress. We must politely but firmly resist such requests unless, in our judgment, they serve the better interests of our clients. At minimum we must advise our clients of the impact of any task or mode of working that prevents us from testing, or creates a false impression of the testing.

http://www.developsense.com/blog/2012/09/premises-of-rapid-software-testing-part-3/



Rapid Software Testing Premise #8

We testers are responsible for the quality of the our work, but not for quality of the product.

(a quality product is a goal the whole team shares, but testers don't have *authority*)

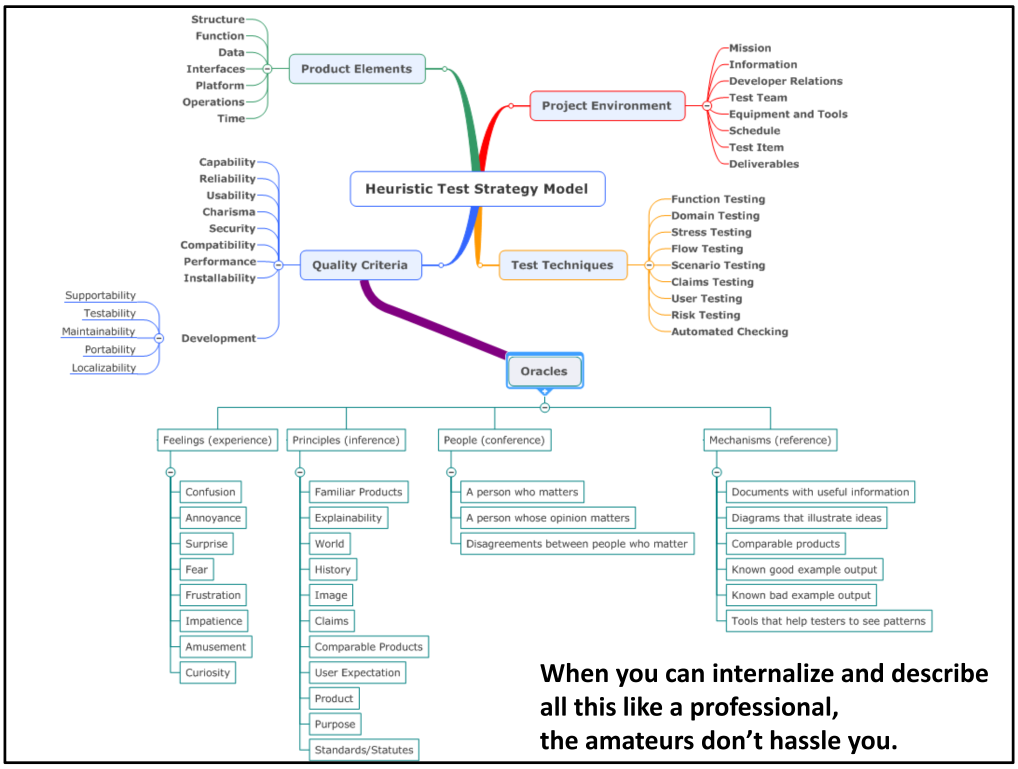
Testers accept responsibility for the quality of their work, although they cannot control the quality of the product. Testing requires many interlocking skills. Testing is an engineering activity requiring considerable design work to conceive and perform. Like many other highly cognitive jobs, such as investigative reporting, piloting an airplane, or programming, it is difficult for anyone not actually doing the work to supervise it effectively. Therefore, testers must not abdicate responsibility for the quality of their own work. By the same token, we cannot accept responsibility for the quality of the product itself, since it is not within our span of control. Only programmers and their management control that. Sometimes testing is called "QA." If so, we choose to think of it as quality *assistance* (an idea due to Cem Kaner) or quality *awareness*, rather than quality *assurance*.

http://www.developsense.com/blog/2012/09/premises-of-rapid-software-testing-part-3/

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.
- **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!
- **Models.** Respecting both tacit and explicit knowledge.



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

Lots of references here:

The Heuristic Test Strategy Model: http://www.satisfice.com/tools/htsm.pdf

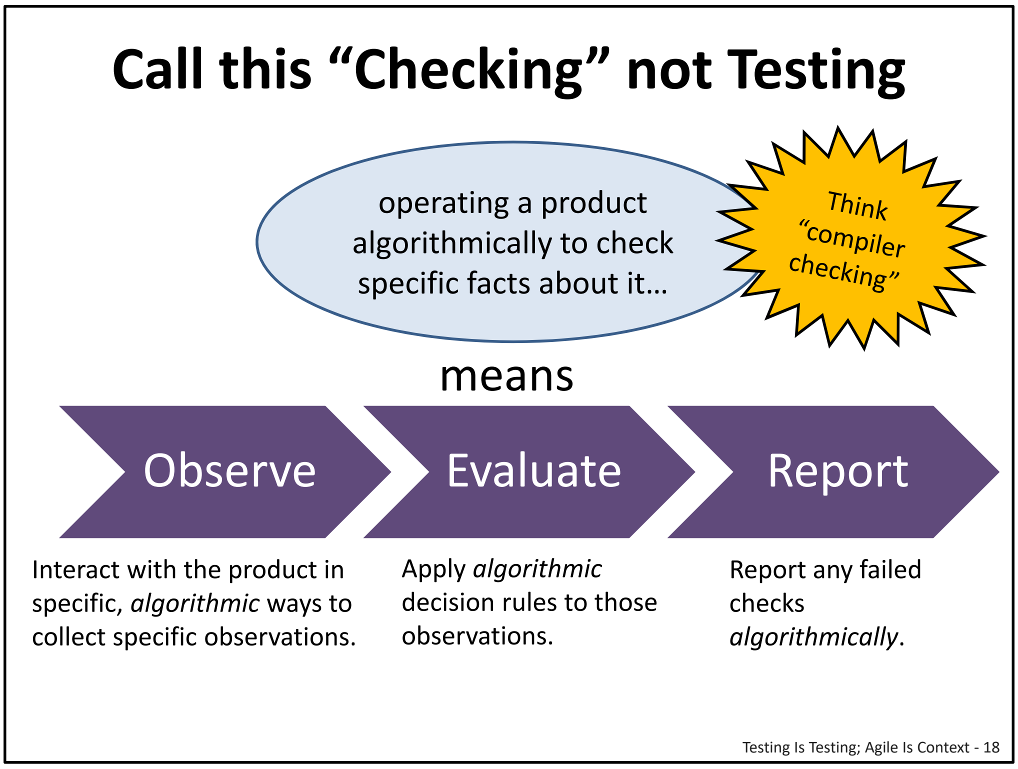
How Models Change: http://www.developsense.com/blog/2014/07/how-models-change/

Oracles: http://www.developsense.com/blog/2015/09/oracles-from-the-inside-out/

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 check, and do many other things besides. But *testing* can't be automated.

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



Call this "Checking" not Testing

operating a product algorithmically to check specific facts about it…

Think "compiler checking"

means

Observe
Evaluate
Report

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

Apply *algorithmic* decision rules to those observations.

Report any failed checks *algorithmically*.

A *check* has three parts.

It requires an *observation*
The observation is linked to a *decision rule*
The observation and the rule can be applied *algorithmically*.

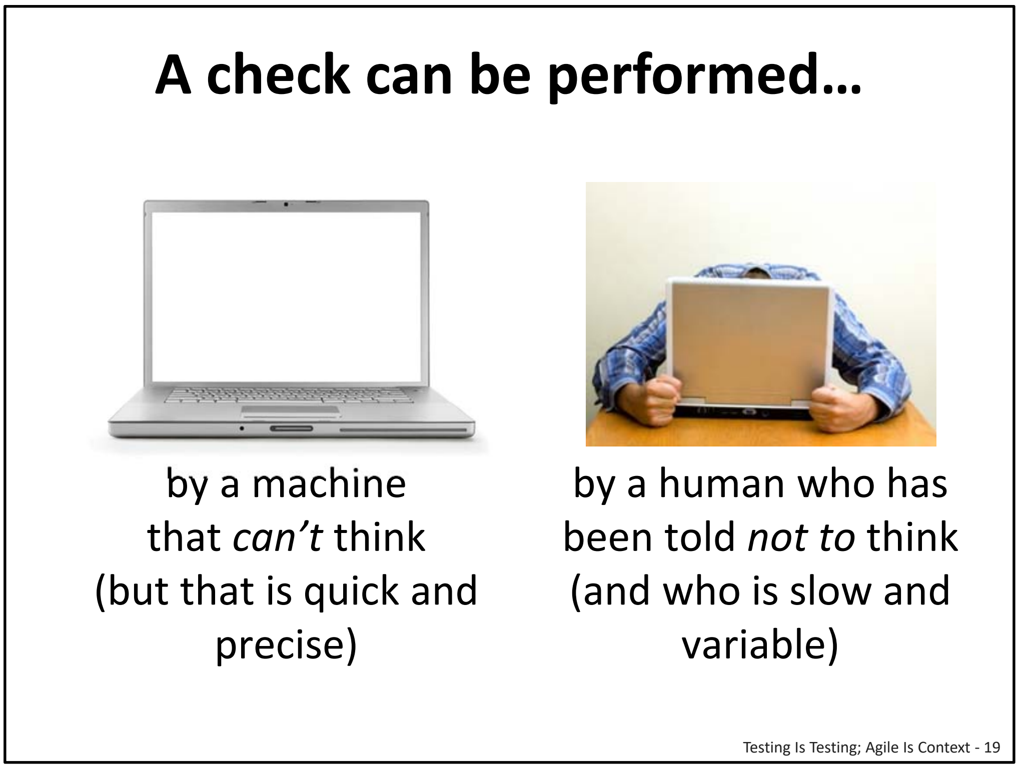
TESTING: A questioning activity that employs skills, senses, emotions and intelligence that we are unable to automate.
CHECKING: An information gathering activity that, *in principle*, could be done by machine.

Testing encompasses checking, not the other way round.

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

Some of our programmer friends have objected to this, say "Come on, that's a silly distinction." Our response is "Oh? Then why do you distinguish between 'compiling' and 'programming'? Why aren't you as excited about 'automating' programming as you are about 'automating' testing?"

Despite what certain Agilists might have you believe, checking is *not* new. In one of the earliest books on computer programming (1957), Dan McCracken refers to "program checkout". Jerry Weinberg has told us that checking was important in the early days because computer time was expensive, programmers were cheap, and the machinery was so unreliable.



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)

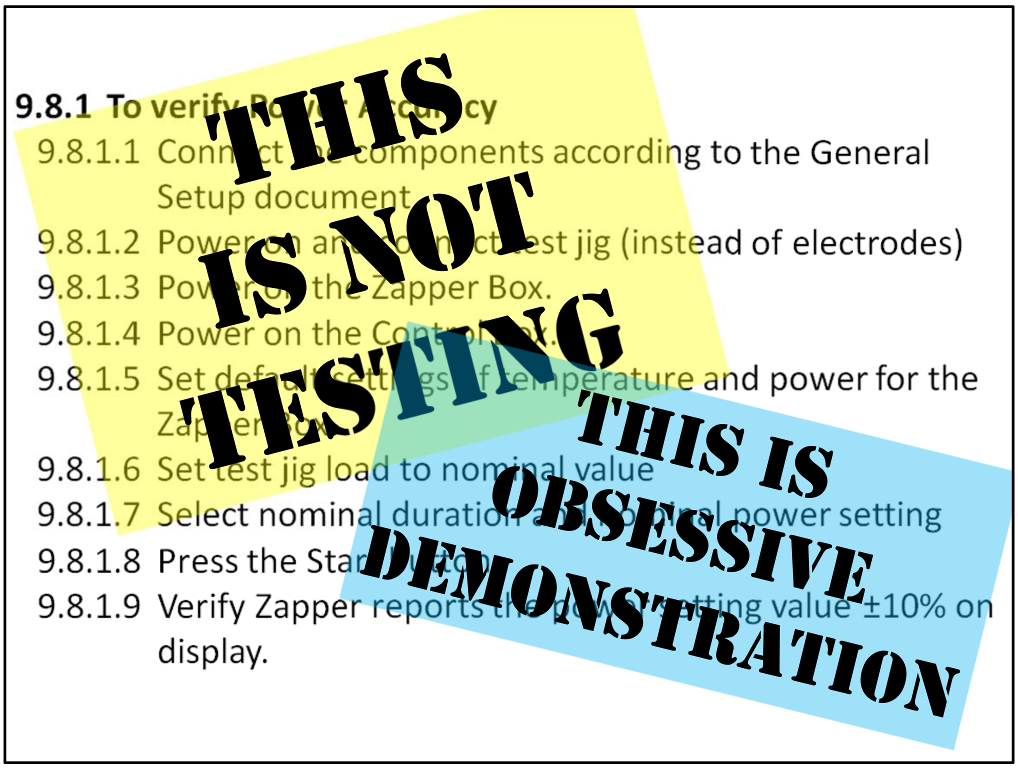
Even though a check itself is skill-free, good checking is surrounded by activities that require many skills, including testing skill, programming skill, and project management skill.
Before the check happens,

- someone must recognize and identify a risk. That takes testing skill.
- someone must translate that risk into a test idea. That takes testing skill.
- someone must express a test idea as a one-bit, yes-or-no question. That takes testing and programming skill.
- someone must turn the question into code. That takes programming skill.
- someone must determine the trigger—an event or action that launches the check. That takes testing skill.
- someone must encode the trigger so that it can be acted on automatically. That takes programming skill.

After the check happens,

- someone must read the bit. That takes programming skill.
- someone must aggregate bits. That takes programming skill.
- someone must design a report. That takes design and testing skill.
- someone must encode the report. That takes programming skill.
- someone must observe the report. That takes testing skill.
- someone must determine meaning. That takes testing and product management skill.
- someone must determine significance. That takes testing and project management skill.
- someone must respond. That takes at least one of several of the skills above.

Variability is not necessarily a liability of human testing. *Requisite variety* is essential to finding problems that can appear in diverse ways in diverse places.


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.

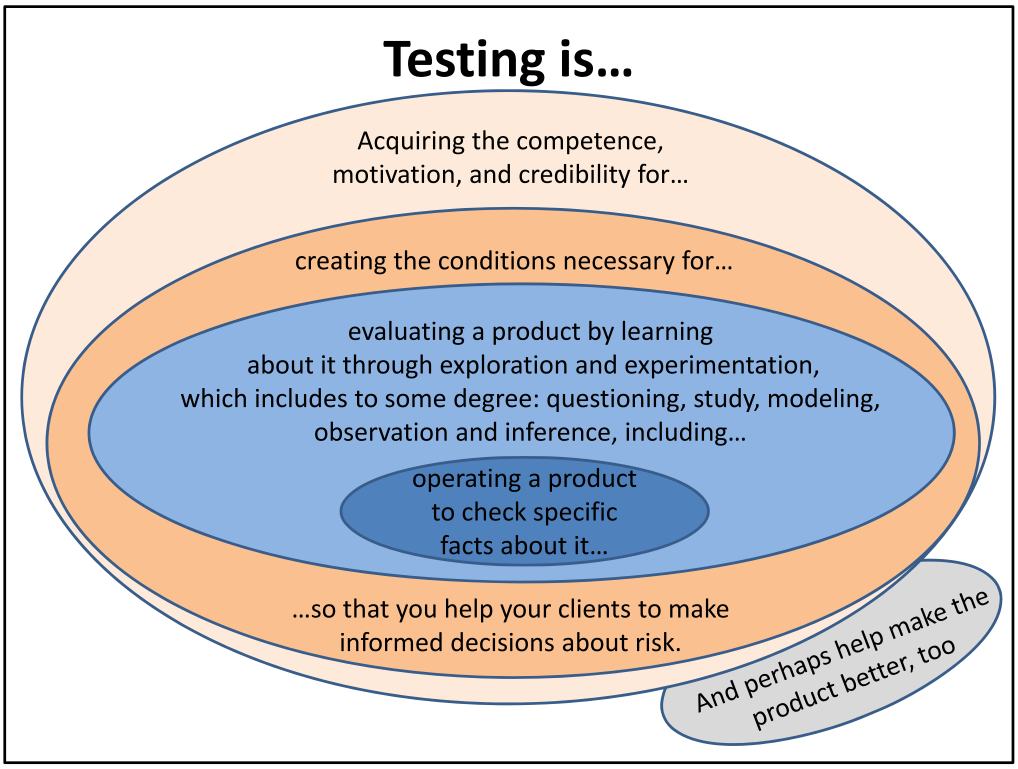
Testing is not test cases. The artifacts of testing are not "tests"; tests are what you think and what you do. This is exactly the same as pointing out that the requirements are not the requirements document; that the territory is not the map. We've talked about this in several ways.

Short: http://www.developsense.com/blog/2017/02/the-test-case-is-not-the-test/

A little longer: http://www.satisfice.com/blog/archives/1346

Medium: "Test Cases Are Not Testing: Toward a Culture of Test Performance" by James Bach & Aaron Hodder (in http://www.testingcircus.com/documents/TestingTrapeze-2014-February.pdf#page=31)

Long: James Bach on "Test Cases are Not Testing"
https://www.youtube.com/watch?v=JLVP_Z5AoyM


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 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

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

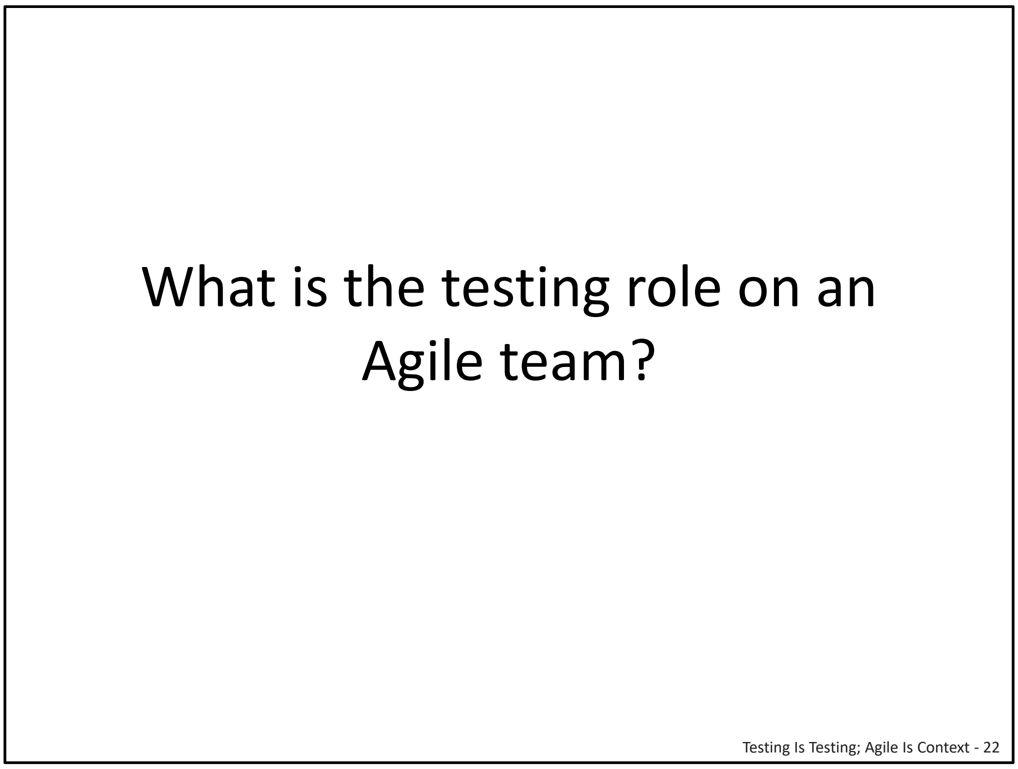
In the past, we've said that testing is "questioning a product in order to evaluate it". The "questions" consist of ordinary questions about the idea or design of the product, or else questions implicit in the various ways of *configuring* and *operating* the product. (Asking questions about the product without any intent to operate it is usually called review rather than testing.) The product "answers" by exhibiting behavior, which the tester *observes* and *evaluates*. To evaluate a product is to infer from its observed behavior how it will behave in the field, and to identify important problems in the product.

Cem Kaner prefers this definition:
"Software testing is a technical investigation for the purpose of revealing the quality of a software product on behalf of stakeholders."

Kaner's definition means the same thing as our definition in every material respect. However, we sometimes prefer more concise wording.

Jerry Weinberg says that "testing is gathering information with the intention of informing a decision", another definition with which we agree.

One implication of all of these definitions is that you can test a product that doesn't yet exist in operable form. Review—the questioning process that precedes what we call test execution—is still part of testing if it serves the process of test design and test execution, or if it helps otherwise in our evaluation of the product.



What is the testing role on an Agile team?

One role for a tester on an Agile team: a *test jumper*. See http://www.satisfice.com/blog/archives/1372

What a Role Is NOT…

- a declaration of the only things you are allowed to do (not a prison)
- a declaration of the things that you and you only are allowed to do (not a fortress)
- permanent and unchanging
- like a tattoo that you can't remove

See http://www.developsense.com/blog/2015/06/on-a-role/.

What a Role Is…

- a commitment to perform some service(s)
- an idea to focus commitments
- a way to help organize effort on a team
- a heuristic for explaining or defining work
- like a hat that you wear

See http://www.developsense.com/blog/2015/06/on-a-role/.

What is the developer role?

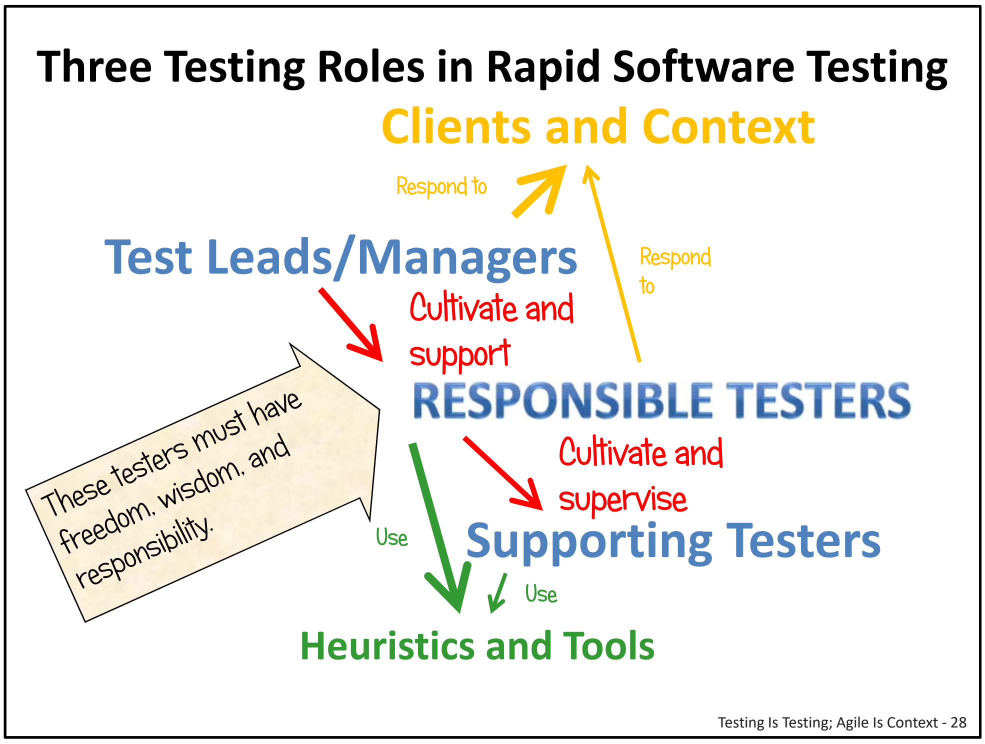
- *To develop* is to connect the world of humans to the world of machinery by creating new machinery to satisfy humans.
- When someone is developing, that person has adopted (if only for that time) a developer's role.
- *A developer's role* includes
  - developing one's self as a developer
  - understanding the world of machinery
  - connecting with the clients of development
  - preparing and maintaining development tools
  - writing code
  - obtaining and applying feedback

What is the testing role?

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

What do we mean by *good* testing?

- Focused on the mission
- Adapted to the context
- Fast
- Inexpensive
- Diversified
- Risk-focused



Three Testing Roles in Rapid Software Testing

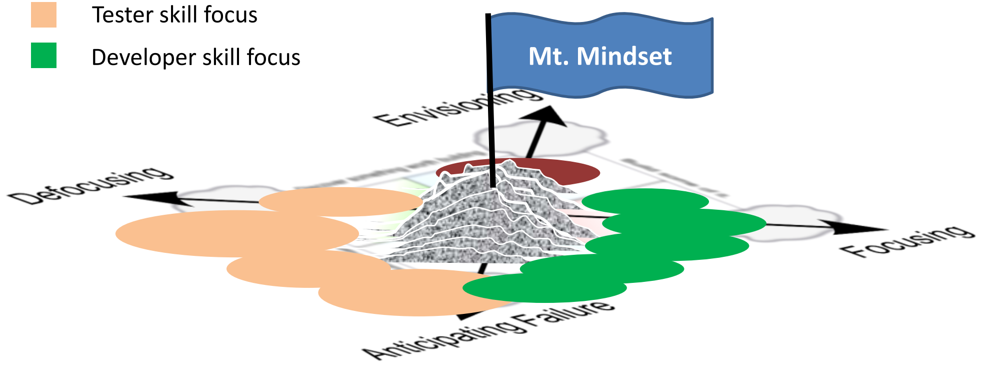
To understand what we mean by “Responsible Testers”, see http://www.satisfice.com/blog/archives/1364.

Why have a *dedicated* testing role? Because changing mindsets is HARD.

🟥 Business analyst skill focus

🟧 Tester skill focus

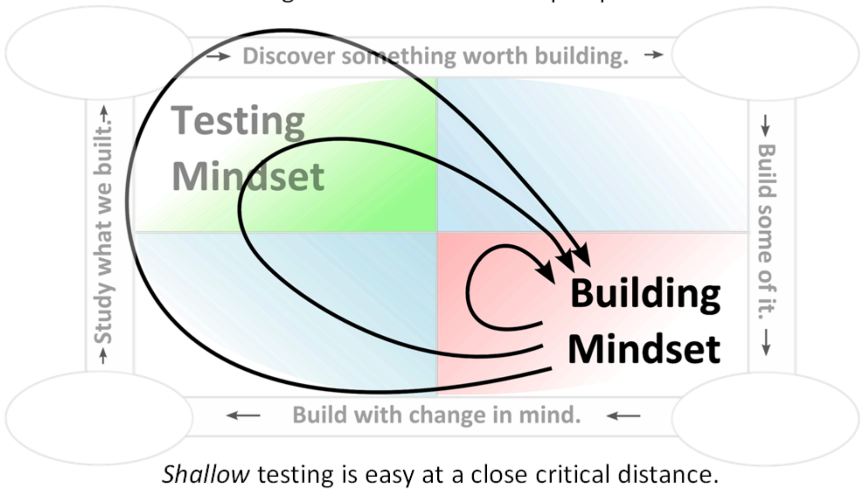
🟩 Developer skill focus



NOTE: We do NOT claim that this work *must* be done by different people, or that the people *must* have different roles. We DO claim that having skilled people focused on testing on an Agile team (collaborating with each other) is a powerful heuristic for addressing the mindset switching problem.

Critical Distance

"Critical Distance" refers to the difference between one perspective and another.
Testing benefits from diverse perspectives.



Shallow testing is easy at a close critical distance.
Deeper testing, more like users, or more focused on more diversified risk,
tends to require or create more distance from the builder's mindset.

Relax… a role is a heuristic, not a prison.

- If I am a developer, can I do testing? Of course!
- *As a developer, you already do testing*. **And** you will have to sharpen your skills and cope with certain handicaps and biases if you want to do *great* testing.
- If I am a tester, can I make quality better? Sure!
- **And** if you do that, you will have adopted, at least temporarily, some kind of developer role. It's hard to wear two hats at once.
- If I am a goalie, can I score goals, too? No rule against it!
- **But** if you come forward, your team's goal is open. **And** the person covering it can't use his hands.
- If I am a janitor, can I offer suggestions to the CEO? Hey, why not?
- **But** *her* role is not necessarily to listen, or to comply.
- If I am not the driver of a car, can I grab the steering wheel?
- Go ahead… **but only if** the driver is incapacitated.

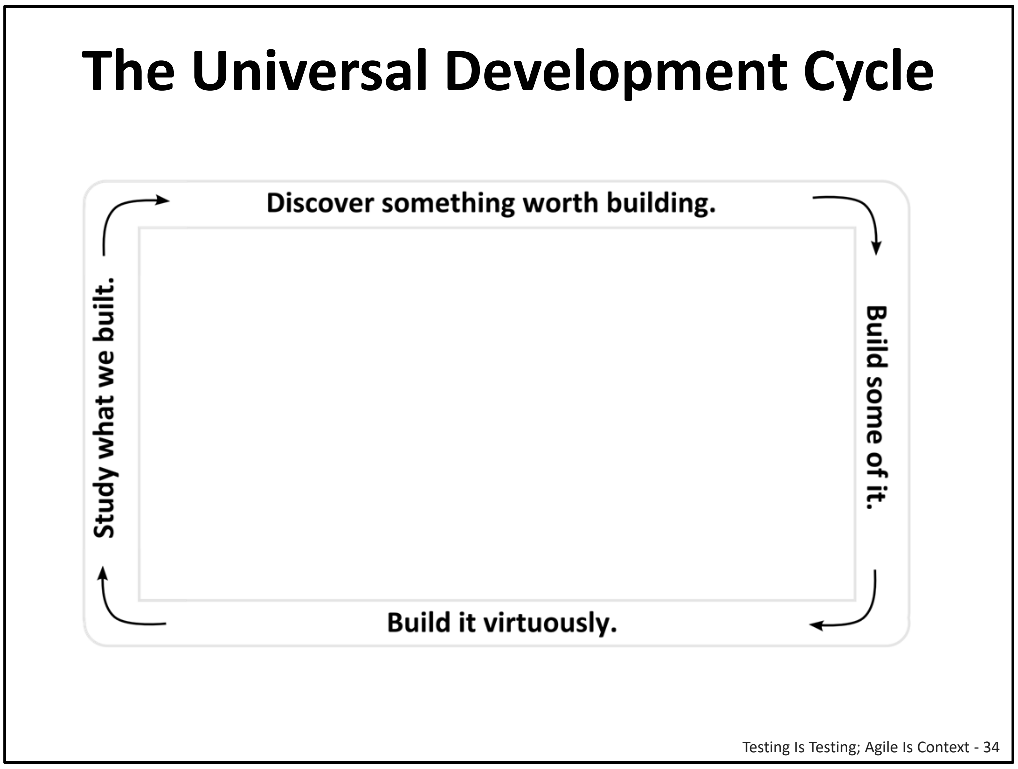
Should testers ONLY look for bugs?

Of course not. But finding *deep, rare, subtle, hidden, or intermittent* problems that threaten the value of the product is an important task in creating a valuable product. That task requires skill and critical distance. Skilled testers, working at close social distance but far critical distance, focus on that task.

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

How do we do development?

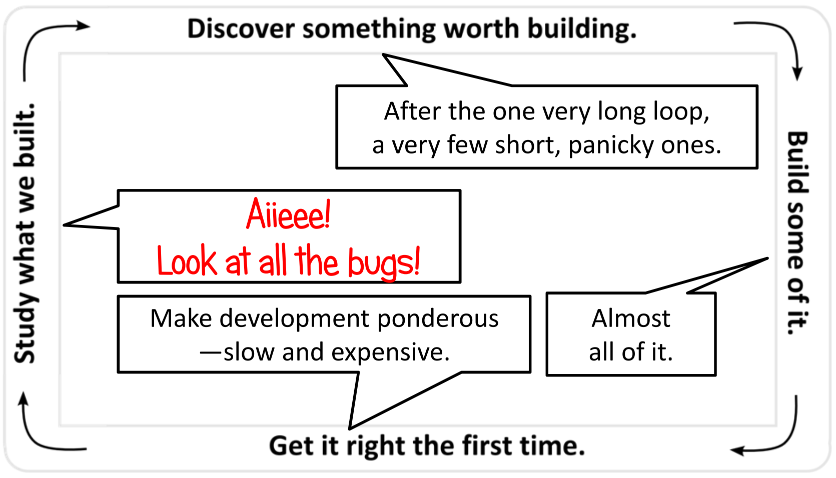
- Discover something worth building
- Build some of it
- Build it *virtuously*
- Study what we’ve built
- …and iterate!



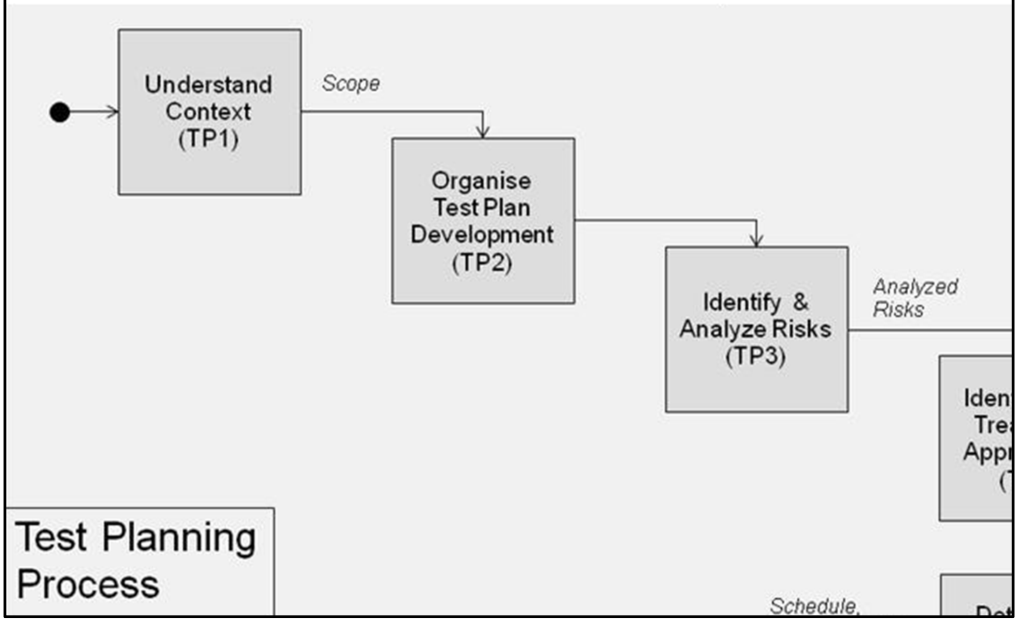
The Universal Development Cycle

I’d like to see the center bubble larger

Traditional Development Cycle


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

Plus: Testing as an Assembly Line



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

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

http://www.agilemanifesto.org

Core Agile Heuristics

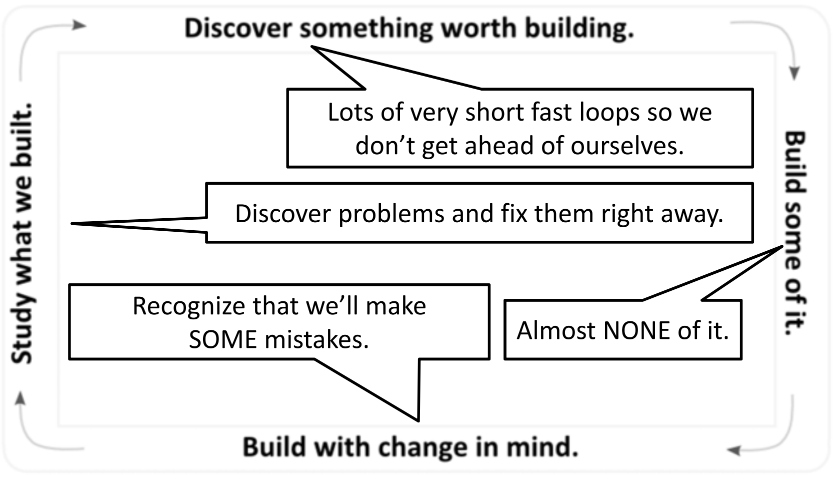
"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

"Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."

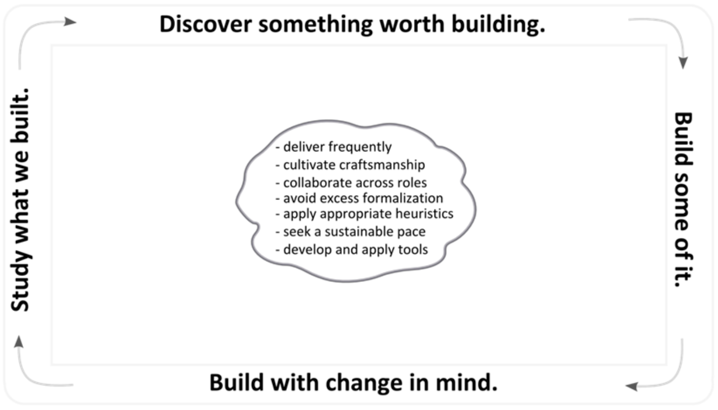
What does it mean to do "Agile Development"?

- Deliver frequently
  - short sprints help prevent us from getting ahead of ourselves
- Collaborate across roles
  - building a quality product is a goal we all share
- Cultivate craftsmanship
  - "being Agile" won't help us without study, practice and skill
- Avoid excessive formalization
  - 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 Cycle


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

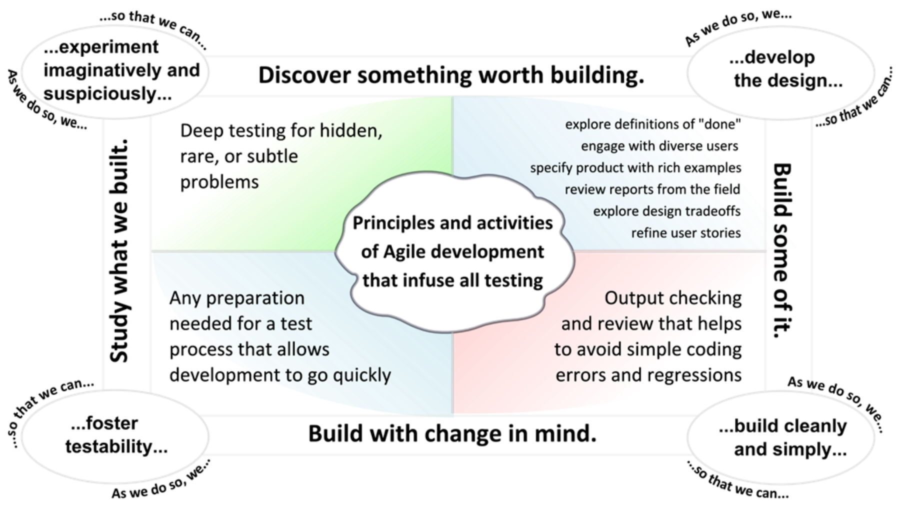
Agile Development Heuristics
(which ought to be infused with testing)


Discover something worth building.
Build some of it.
Build with change in mind.
Study what we built.
- deliver frequently
- cultivate craftsmanship
- collaborate across roles
- avoid excess formalization
- apply appropriate heuristics
- seek a sustainable pace
- develop and apply tools

Developing the Design

- Exploring definitions of done
  - and expecting that they will be re-evaluated on the basis of new information
- Engaging with diverse users
  - recognizing a broad interpretation of "user"
- Specifying product with rich examples
  - remembering that examples aren't tests, but they help
- Reviewing reports from the field
  - informing design and testing with real-world problems
- Exploring design trade-offs
  - considering cost and value throughout
- Refining user stories
  - developing rich models of the product in its context

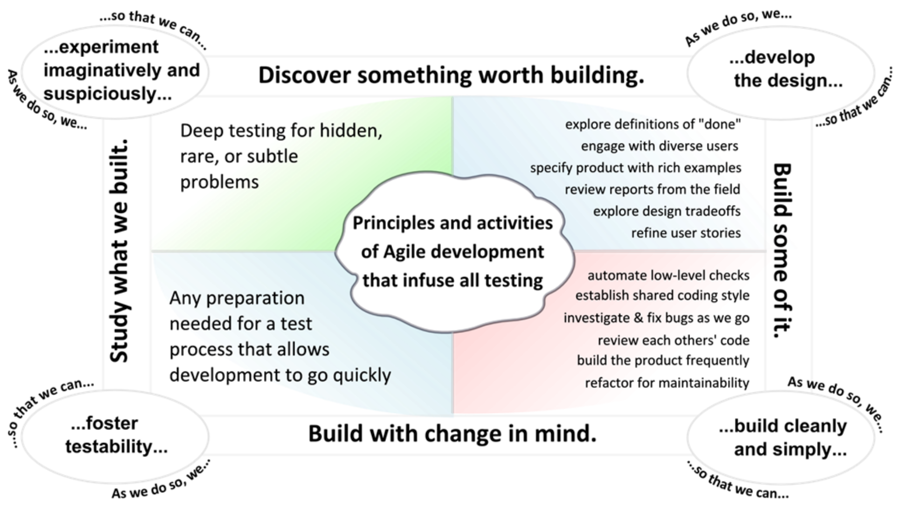
Developing the Design


...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
explore definitions of "done"
engage with diverse users
specify product with rich examples
review reports from the field
explore design tradeoffs
refine user stories
Principles and activities of Agile development that infuse all testing
Study what we built.
Build some of it.
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
...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...

Building Cleanly and Simply

- Automating low-level checks
  - producing a vastly more testable product
- Establishing shared coding style
  - reducing the overhead of interpretation
- Reviewing each other's code
  - swatting bugs before they even get into the house
- Building the product frequently
  - making it possible to obtain constant feedback
- Re-factoring for maintainability
  - simplifying while anticipating change
- Investigating and fixing bugs as we go
  - swatting more bugs before they hide behind the drywall

Building with Change in Mind



...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...
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.
Deep testing for hidden, rare, or subtle problems
explore definitions of "done"
engage with diverse users
specify product with rich examples
review reports from the field
explore design tradeoffs
refine user stories
Principles and activities of Agile development that infuse all testing
Any preparation needed for a test process that allows development to go quickly
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

Different Types of Testability

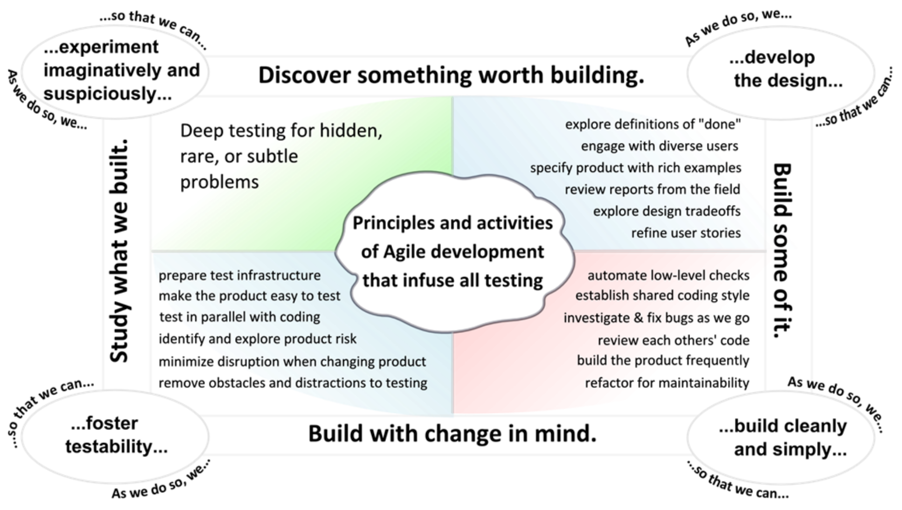
- Project-related testability
  - good, close relations on the team; good infrastructure
- Intrinsic testability
  - visibility and controllability built into the product
- Epistemic testability
  - a developing understanding of the risk gap
- Value-related testability
  - knowing what clients and customers value
- Subjective testability
  - the skill-set and mindset of the tester, relative to the product and the test space

http://www.satisfice.com/tools/testable.pdf

Fostering Testability

- Preparing test infrastructure
  - selecting and developing tools and environments
- Making the product easy to test
  - advocating for visibility and controllability
- Identifying and exploring product risk
  - digging up buried assumptions
- Minimizing disruption when changing product
  - making testing a service, not an obstacle
- Removing obstacles and distractions to testing
  - addressing issues so testing can go quickly and smoothly
- Testing in parallel with coding
  - accepting anything our clients want tested at any time

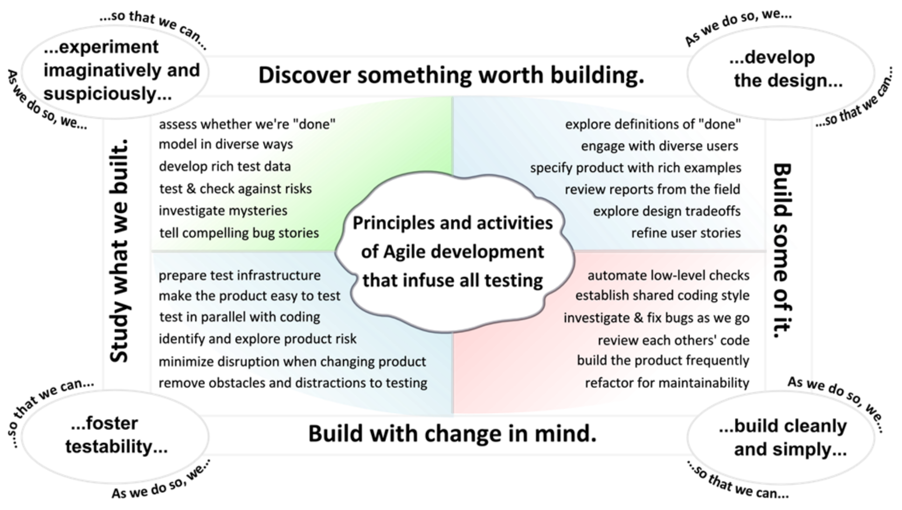
Fostering Testability


...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
explore definitions of "done"
engage with diverse users
specify product with rich examples
review reports from the field
explore design tradeoffs
refine user stories
Study what we built.
Build some of it.
Principles and activities of Agile development that infuse all testing
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...

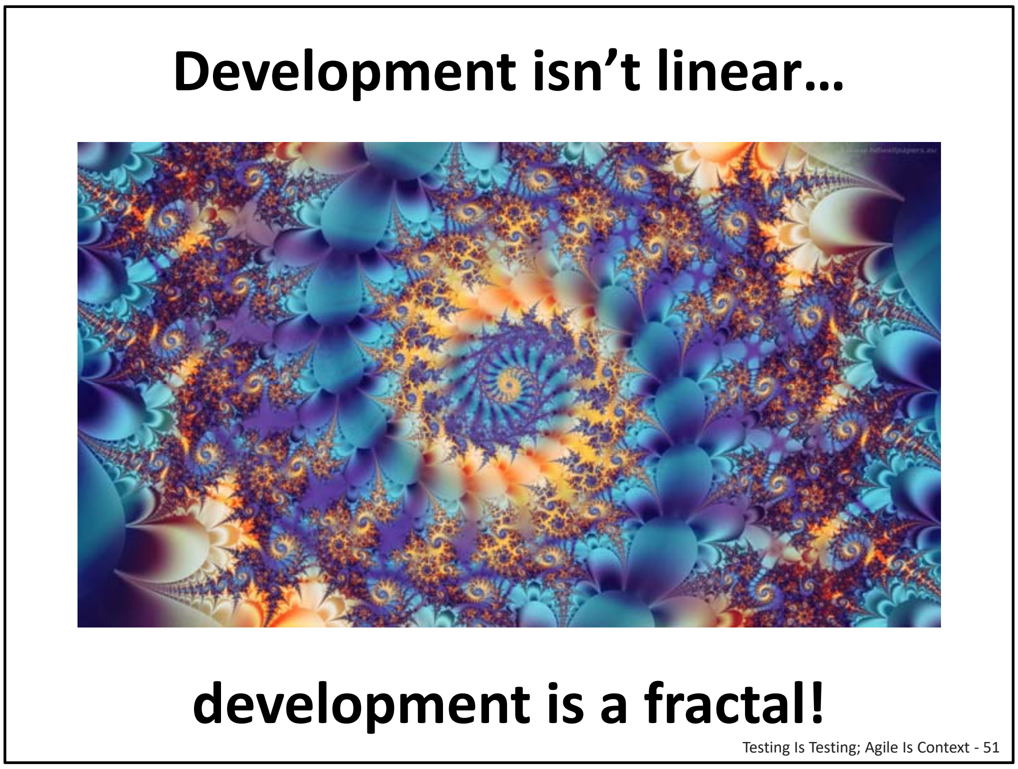
Experimenting Imaginatively and Suspiciously

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

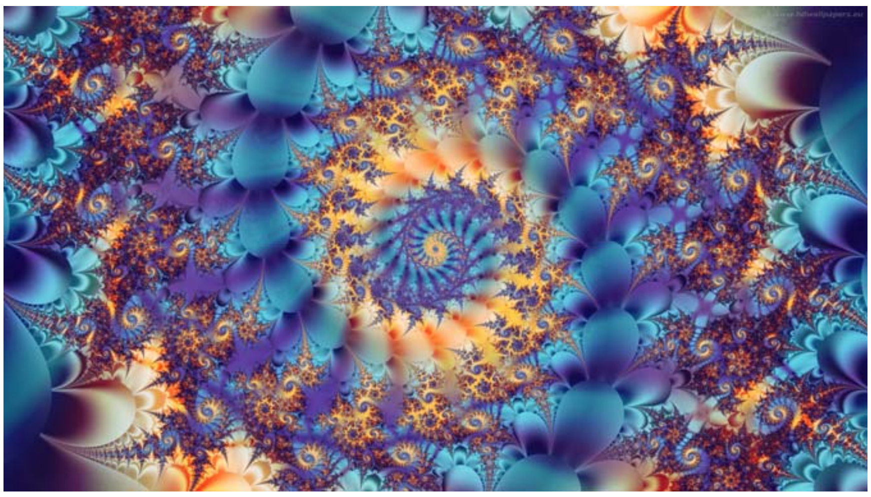
RST's Agile Quadrants in Detail



...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...
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
Principles and activities of Agile development that infuse all testing
Study what we built.
Build some of it.
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...



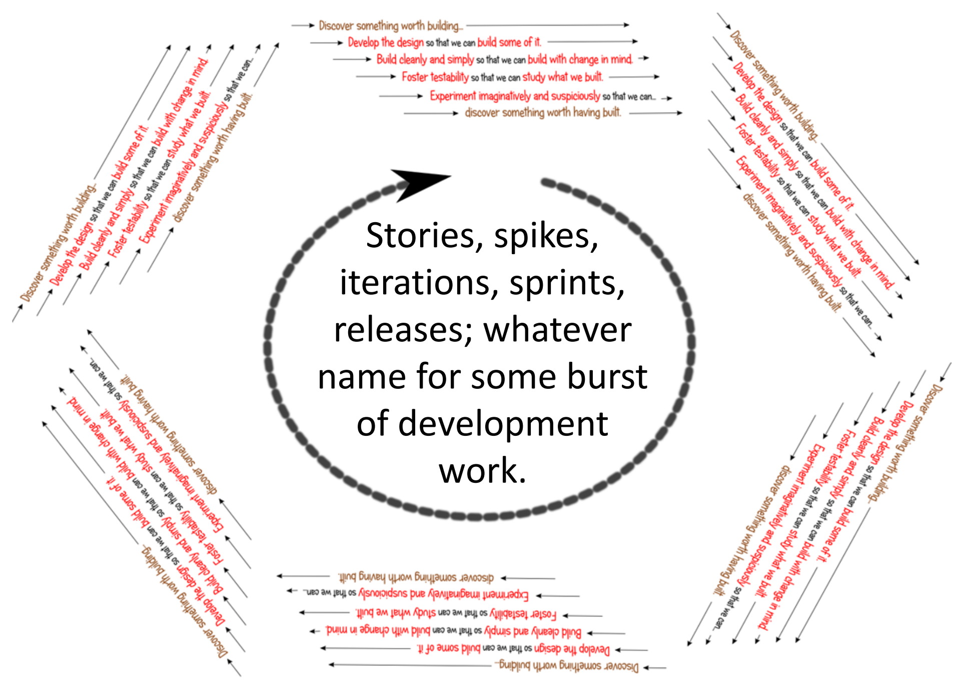
Development isn't linear…



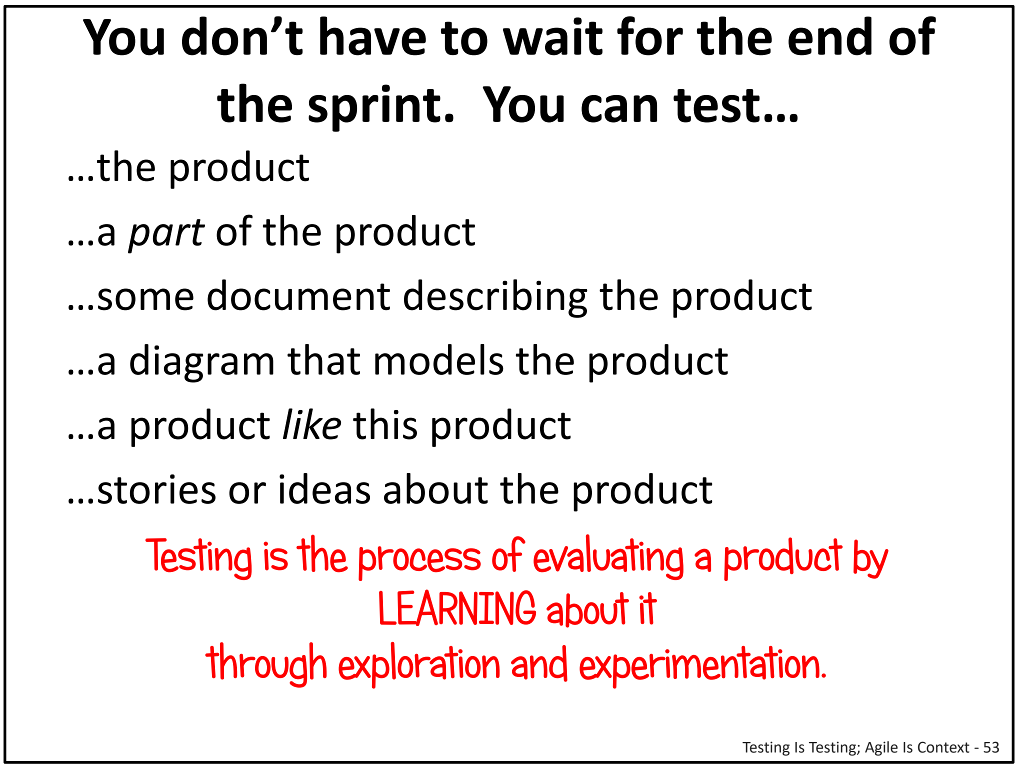
development is a fractal!

Neither development nor the testing that happens within it is a linear process, like one that happens on an assembly line. We are not simply assembling a product; we're developing it, in loops of analysis, design, coding, and evaluation. Each step feeds back into subsequent iterations of the work. Building a new product requires us to learn how to build the product, which we learn by trying to build the product. That learning happens at every level, from an entire project, to development of a feature, to a story about that feature, to a sprint, to an object or class to a line of code… Each iteration of building may include discoveries, failed attempts, flashes of insight, minor stumbles, corrections... Each iteration may be disrupted by new requirements, or newly-recognized aspects of requirements we thought we understood.

Agile development is not about doing everything as quickly as possible. Agile development is about responding quickly and nimbly when things knock us off balance. Rapid response to change requires rapid feedback—the kind of feedback that Rapid Software Testing is intended to provide.


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.

Stories, spikes, iterations, sprints, releases; whatever name for some burst of development work.



You don’t have to wait for the end of the sprint. You can test…

…the product

…a *part* of the product

…some document describing the product

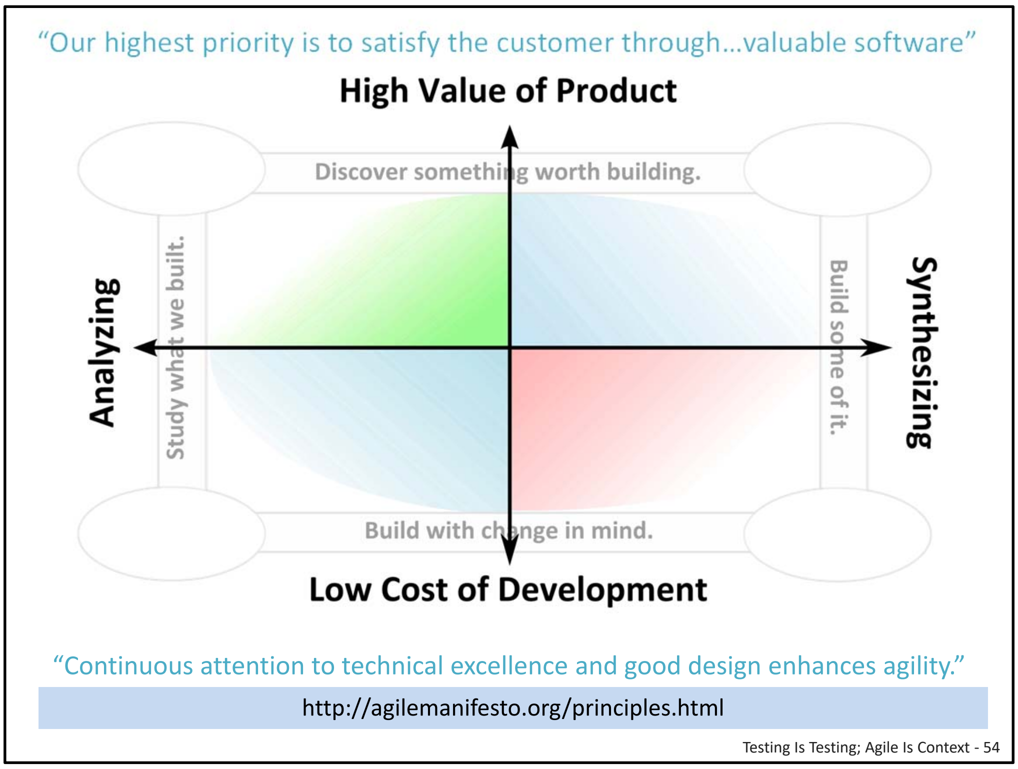
…a diagram that models the product

…a product *like* this product

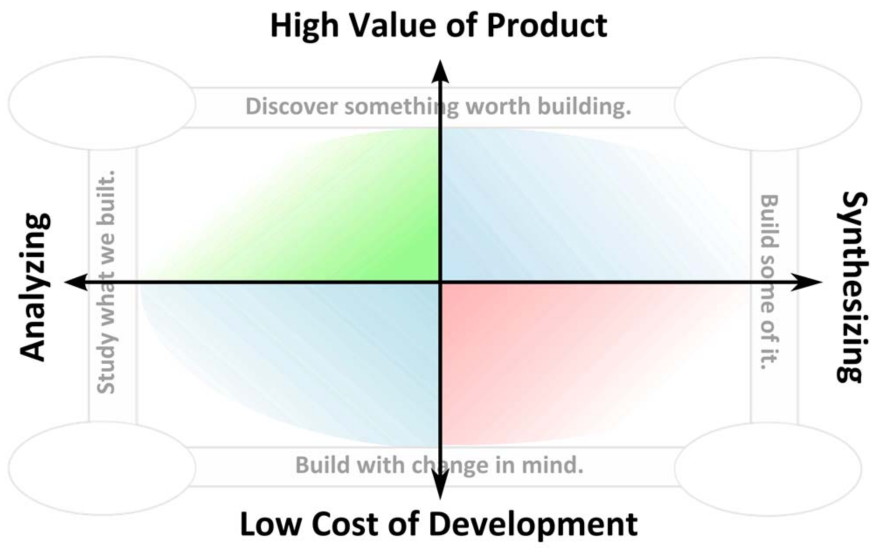
…stories or ideas about the product

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

Think expansively about what might constitute a “product”. A product is something that someone has produced. Even if it’s an only an idea, you can explore it, and you can perform thought experiments on it.



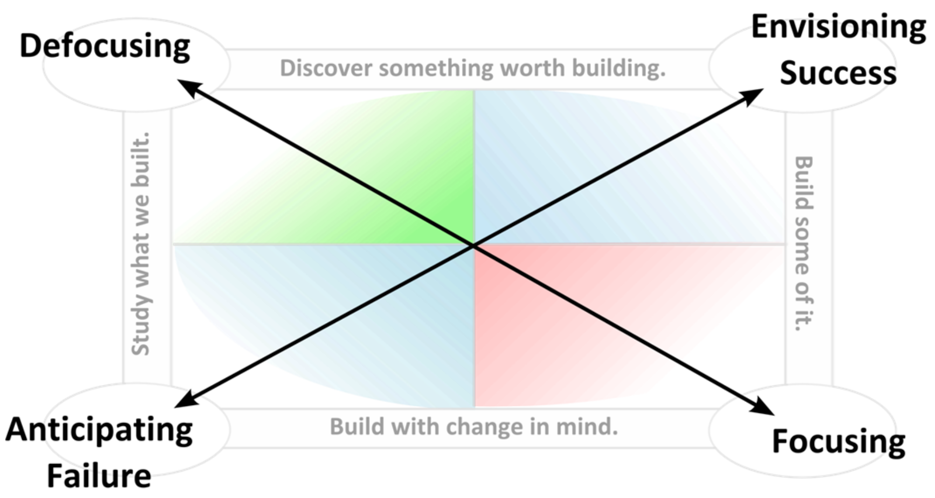
“Our highest priority is to satisfy the customer through…valuable software”



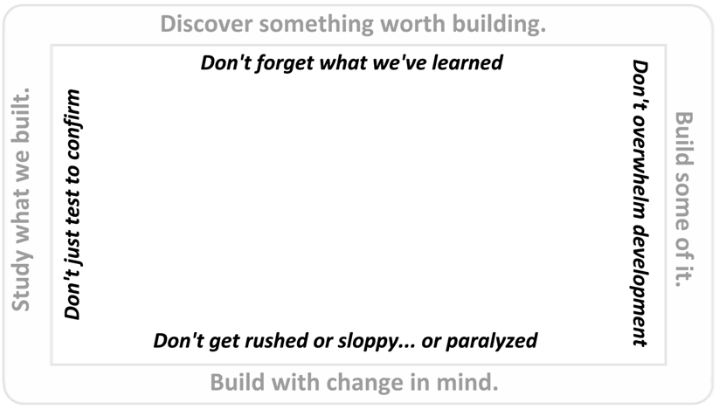
“Continuous attention to technical excellence and good design enhances agility.”

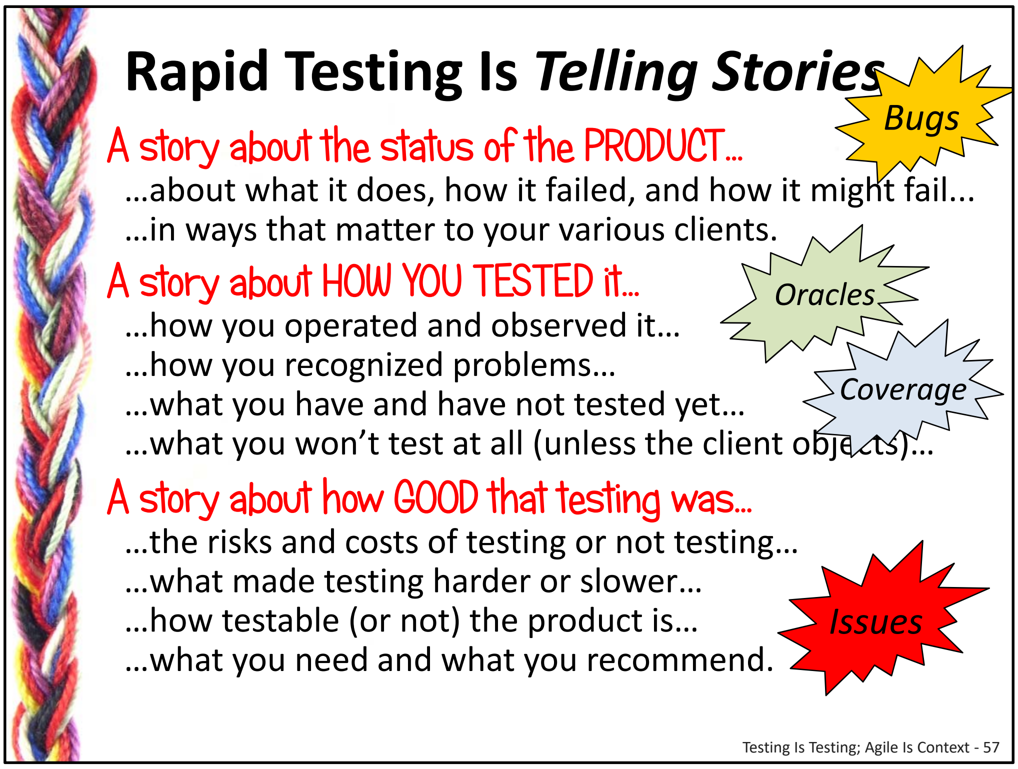
http://agilemanifesto.org/principles.html

Productive Polarities



Don’t…


Discover something worth building.
Don't forget what we've learned
Don't overwhelm development
Build some of it.
Don't get rushed or sloppy... or paralyzed
Build with change in mind.
Don't just test to confirm
Study what we built.



Rapid Testing Is *Telling Stories*

Bugs

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…

Oracles

…how you operated and observed it…
…how you recognized problems…
…what you have and have not tested yet…

Coverage

…what you won't test at all (unless the client objects)…

A story about how GOOD that testing was…

…the risks and costs of testing or not testing…
…what made testing harder or slower…
…how testable (or not) the product is…
…what you need and what you recommend.

Issues

Technical Suggestions

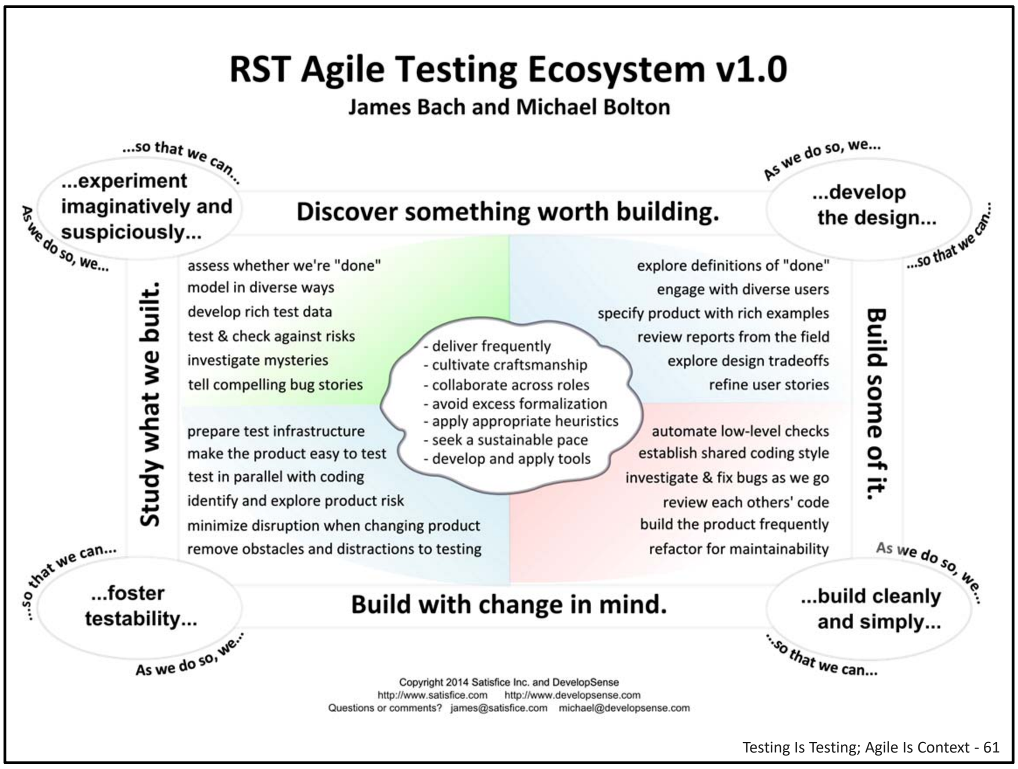
- Resist test cases and scripts; focus on test activities and the testing story.
- Let risk guide your testing; focus testing on risks.
- Test in short, uninterrupted sessions; review and discuss them; seek and provide feedback.
- Avoid premature, excessive formalization.
- Keep documentation concise.
- Use recording tools like an airplane “black box”.
- Emphasize exploratory scenario testing.
- ASK FOR TESTABILITY!

Tools?

- DON'T use them to "do" the testing. Tools don't do testing; YOU do.
- DON'T become fixated on tools.
- DO prefer lightweight tools, in general.
- 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)

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.



RST Agile Testing Ecosystem v1.0

James Bach and Michael Bolton

...experiment imaginatively and suspiciously...

...so that we can...

Discover something worth building.

As we do so, we...

...develop the design...

...so that we can...

Build some of it.

- explore definitions of "done"
- engage with diverse users
- specify product with rich examples
- review reports from the field
- explore design tradeoffs
- refine user stories

- 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

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.

- assess whether we're "done"
- model in diverse ways
- develop rich test data
- test & check against risks
- investigate mysteries
- tell compelling bug stories

- 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

As we do so, we...

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

Further Reading

- In addition to the bibliographies in Agile Testing and More Agile Testing, have a look at…
- “Test Cases Are Not Testing: Toward a Culture of Test Performance” by James Bach & Aaron Hodder
  - http://www.testingcircus.com/documents/TestingTrapeze-2014-February.pdf#page=31)
- Testing Problems are Test Results
  - http://www.developsense.com/blog/2011/09/testing-problems-are-test-results/
- Testers: Get Out of the QA Business
  - http://www.developsense.com/blog/2010/05/testers-get-out-of-the-quality-assurance-business/
- Testing and Checking Refined
  - http://www.satisfice.com/blog/archives/856
- RST Methodology: Responsible Tester
  - http://www.satisfice.com/blog/archives/1364

Further Reading

- Test Jumpers: One Vision of Agile Testing
  - http://www.satisfice.com/blog/archives/1372
- Done, the Relative Rule, and the Unsettling Rule
  - http://www.developsense.com/blog/2010/09/done-the-relative-rule-and-the-unsettling-rule/
- The Undefinition of Done
- http://www.developsense.com/blog/2011/07/the-undefinition-of-done/
- At Least Three Good Reasons for Testers to Learn to Program
  - http://www.developsense.com/blog/2011/09/at-least-three-good-reasons-for-testers-to-learn-to-program/