Refactoring the Agile Testing Quadrants

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

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

Given, When, Then

GIVEN that I am a human being
AND a passionate, committed tester
WHEN people talk about Agile Software Development
AND reduce it to a bunch of formulaic keywords
AND reduce testing to mechanistic checking
AND reduce “qualifications” to multiple choice questionnaires
AND dismiss deep, skilled, rich, inexpensive, fast testing
AND don’t help to make life better for people
THEN I get upset
AND I have too much to talk about in only one hour

Why I'm becoming a grumpy old guy:

Increasingly, testing is confused with "checking builds".

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

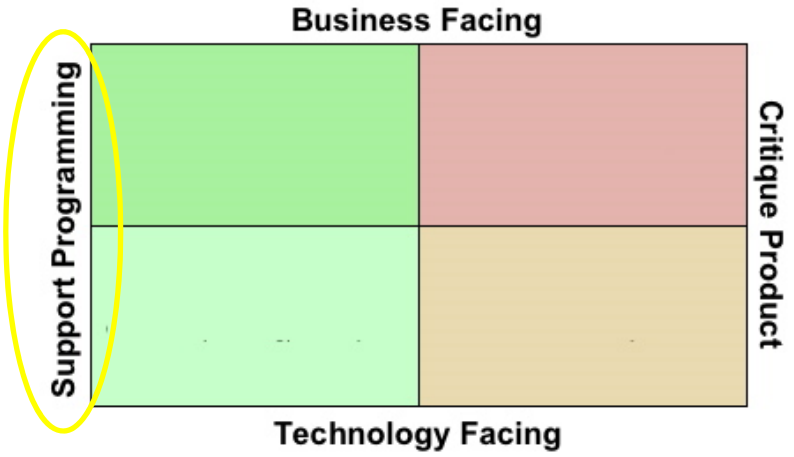
Tools are great. **We should use them.** We should use them **a lot** to help us develop an understanding of our products. **Tools can help us to be powerful.**

But what I'm seeing at conferences and in talk about testing often looks like elaborate attempts to **avoid making contact** with the software, our clients, our customers, and our mission.

Our Problems with the Agile Testing Quadrants: A History

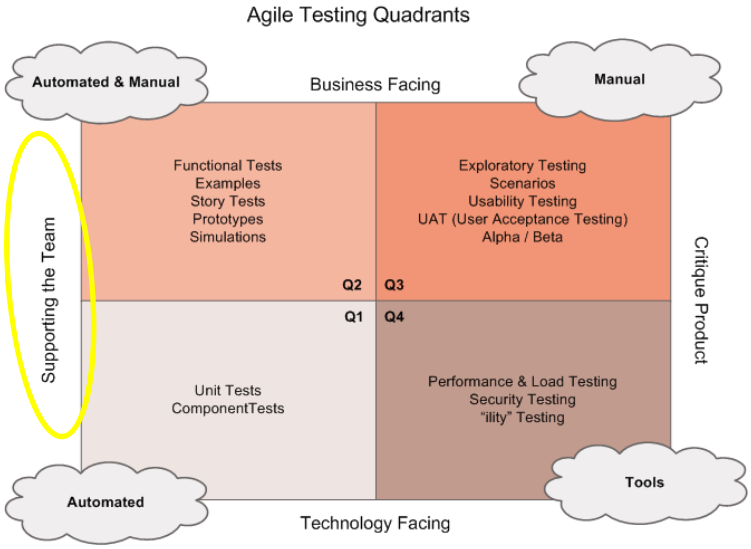
- James encountered the quadrants first in 2003 or so, when Brian Marick explained them to him; I started to hear about them shortly after that.
- I participated in the Agile Testing Mailing list, which seemed to exalt processes and tools, but not talk about *testing* very much.
  - There was lots of talk about checking, but they didn't call it that—but in fairness, back then, I didn't either.
- I abandoned the list in 2008 or so, after I got tired of what I felt was misrepresentation and dumbing-down of testing.
- **I feel that the quadrants helped, and still help, to feed that misrepresentation.**
- We have learned much more about (agile) testing and how to discuss it since the quadrants first arrived. It's time for a major refactoring.

Marick's Original



See http://www.exampler.com/old-blog/2003/08/21/, http://www.exampler.com/old-blog/2003/08/22/#agile-testing-project-2, and subsequent posts.

Crispin & Gregory's (Earlier) Version



See Crispin & Gregory, *Agile Testing*

Supporting Programming or the Team?

- Marick's original and his comments on it frame simple output checks as more "integral" to the programming process than vigorous testing.
  - Maybe he was talking about lower critical distance; okay.

Critical Distance and Social Distance

By **critical distance** we mean
A difference between two ways of thinking about some thing,
or *an absence of knowledge about some thing in favor of other things.*

By **social distance** we mean
any barrier to or absence of harmony and cooperation among people.

Cultivate critical distance.
Eliminate social distance.

Supporting Programming or the Team?

- Marick's original and his comments on it frame simple output checks as more "integral" to the programming process than vigorous testing.
  - Maybe he was talking about lower critical distance; okay.
  - Nonetheless, let's treat programming and testing all as connected together, in super-rapid feedback loops. It's *agile development*, right?
- The Crispin & Gregory version implies that critique is not supporting the team, or not the work of programming.
  - Testing—critiquing the product—IS supporting the team!
  - Programmers can provide powerful and valuable critique!
- It also implies that that testers do not belong in Agile unless they are programmers; unless they write code.
  - Testers may *or may not* write code, use particular tools, or apply particular skills. Context matters. The mission matters.

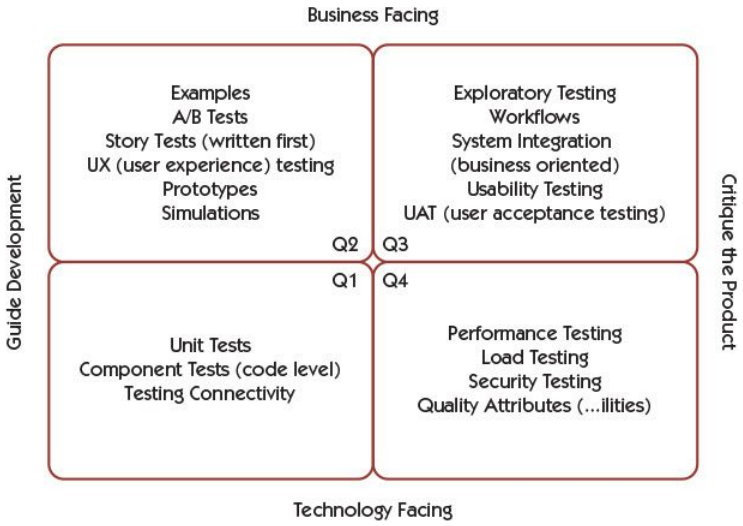
Automated, Manual, Tools… Wait… Huh?

- Tools are not remarkable in testing. Good testers use them anywhere, *everywhere*, for lots of purposes.
- There is no such thing as "manual" or "automated" testing, just as there isn't "manual" or "automated" programming.
  - See http://www.developsense.com/blog/2013/02/manual-and-automated-testing/
- There may be useful distinctions in *the means by which we interact with the product* — say, via the GUI, via APIs, or debuggers.
- It may be relevant to consider how *naturalistic* our interaction is.
  - We might focus on user tasks, and operate the product at the surface, as users do. But we might also do things that No User Would Ever Do
- It may be relevant to account for what, specifically, we're observing and examining.
  - Are we looking at the whole system, or only at components of it?

Reification (turning tests into *things*)

- Test cases are not tests; examples are not tests.
- Tests are not artifacts; they're performances.
- The most important parts of testing (tacit knowledge, social judgment, context awareness) cannot be scripted or encoded.
- It is pointless to discuss whether "business people" can "read the tests" because what they can read are not tests – they are partial representations of testing activity (or else they are checks).
- Trying to communicate testing primarily through writing or code (processes and tools; contracts; comprehensive documentation) is inconsistent with important Agile principles.
  - Instead: prefer conversation and demonstration of testing work

Crispin & Gregory's (Newer) Version



See Crispin & Gregory, *More Agile Testing*

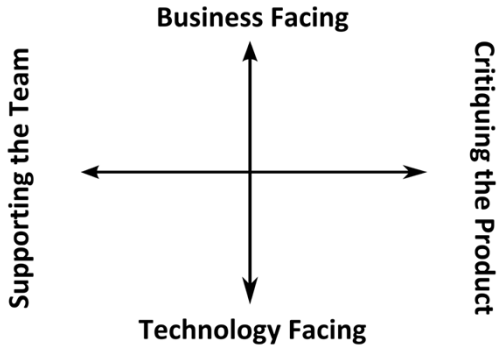
Why you might like the quadrants:

Because they represent a generic diversified test strategy!

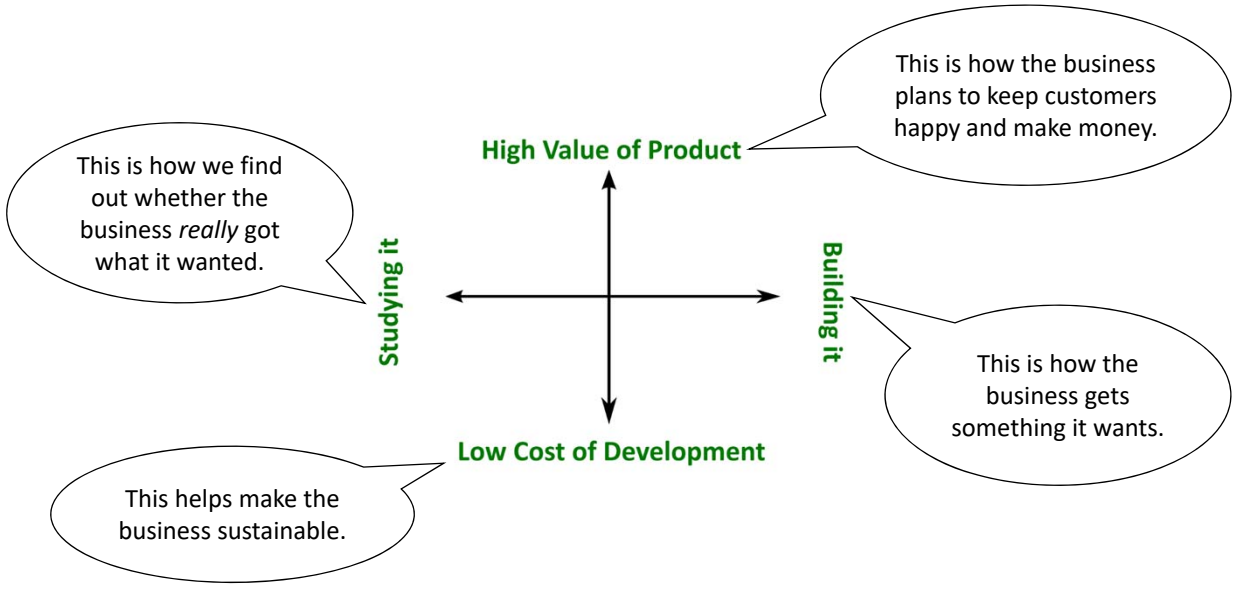
Crispin & Gregory v2: some progress, but…

- The second version omits and therefore successfully avoids the automated/manual/tools problem. That's a definite improvement.
- "Guiding development" is still odd, seeming to put the testing cart before the design, programming, and management horse.
- Both versions pin certain techniques and approaches to certain quadrants in ways that seem confusing.
  - Isn't TDD a form of exploratory development?
  - Is testing connectivity a first-quadrant activity?
  - Can we not test component using an exploratory approach?
  - "Business oriented" systems integration is listed, but "technology oriented" systems integration is missing. Shouldn't that warrant a mention?
  - Aren't "-ilities" (capability, reliability, testability…) relevant everywhere?
  - Is security testing technology facing? Isn't it business facing?

Dimensions of Crispin/Gregory "Agile Testing Quadrants" Based on Marick



Let's refactor those in terms of what the business wants.



So “facings” are beside the point.

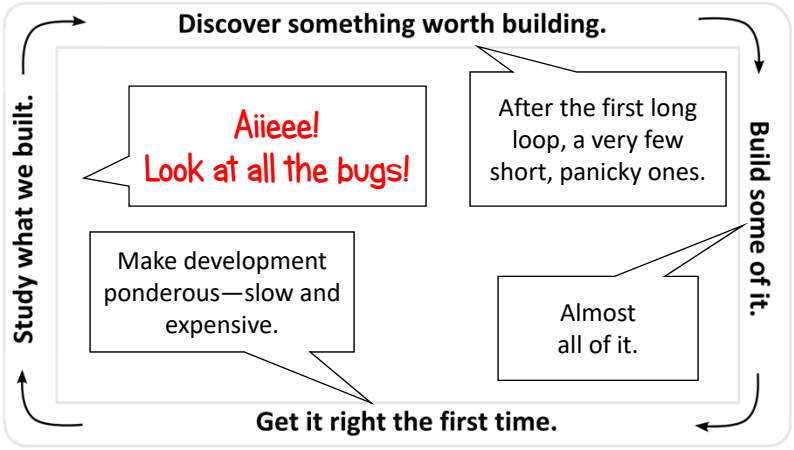
- THE BUSINESS needs us to produce something of value.
- THE BUSINESS needs us to do that efficiently.
- THE BUSINESS needs to learn what it values over time rather than guessing at the beginning of the project and freezing those guesses.

Hence, the core heuristic of agile: continually re-focus on value (in order to produce value) and develop software in ways that reduce the *cost* of change (rather than reduce the *need* for change).

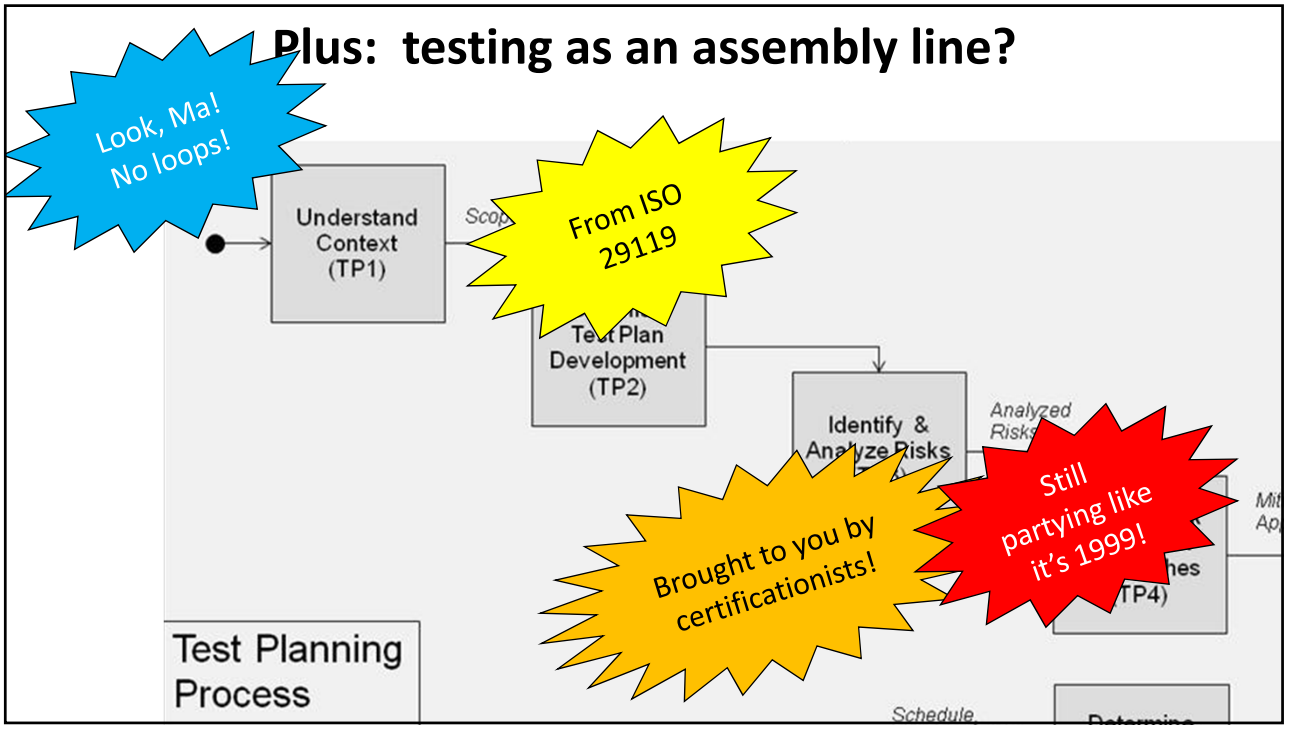
- So: “technology-facing” simply means doing things that help us to build the product and to **build with change in mind** – an activity our business clients need but typically do not *directly* care about (or sometimes even know about.)
- That’s cool, because the business hires us, as technical experts, to take care of that stuff for them. That’s the service we provide!

In the Beginning… the Universal Development Cycle…

"Traditional" Development Cycle



Plus: 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

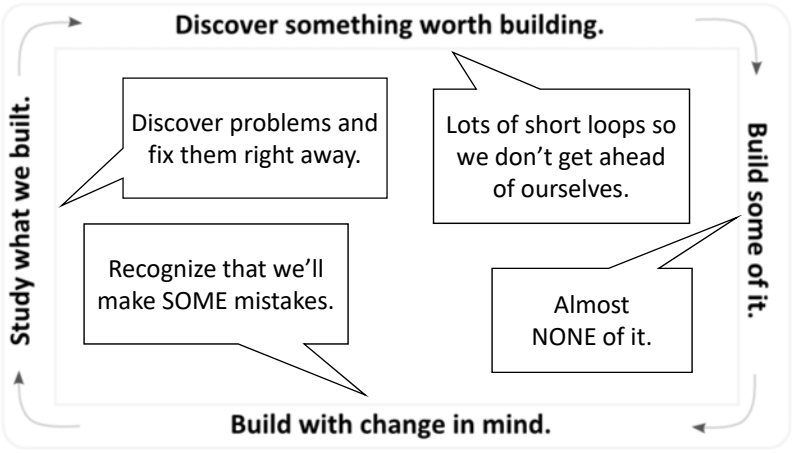
Principles of Agile Software Development

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.

Agile Development Cycle



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

- Deliver often (so the product can be evaluated)
- Collaborate across roles
- Develop craftsmanship
- Don't be too formal
- Be prepared to try things, to fail, and learn
- Build and use tools expertly
- Seek a sustainable pace

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 (craftspeople, empaths, and stickynoters)
- marketers and certifiers
- confusion about testing
- confusion about tests
- confusion about agility

Some Problems With "Agile" Software Development

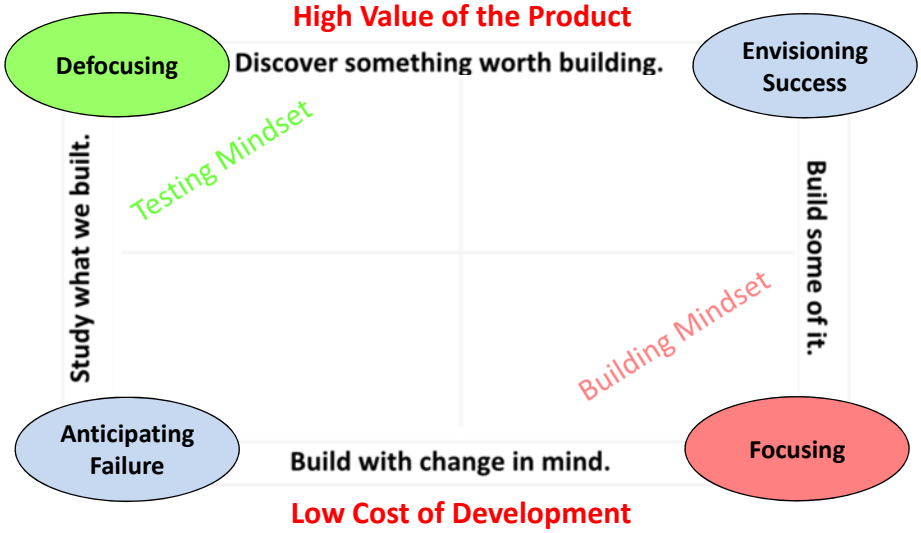
- Agile's earliest roots are in eXtreme Programming (XP), which was *extremely* focused on *programmers*. (This was much more a feature, and much less a bug.)
- A bug: in many places, "Agile testing" became dominated by things in programmers' mindsets: unit testing, functional correctness, solving problems with code, "definition of done"...
- And, in many places, testing became confused with output checking...
- …yet there can be *many problems* in the relationships between people and the product.
- We don't know where those problems are… and that's where risk lives.

So what would testing look like in Agile contexts?

Individuals and interactions over processes and tools	Focus on the skill set and the mindset of the individual tester
Working software over comprehensive documentation	Eliminate wasteful documentation; emphasize investigation and learning
Customer collaboration over contract negotiation	Answer the needs of the client and the team
Responding to change over following a plan	Respond rapidly to the ever-changing mission of testing.

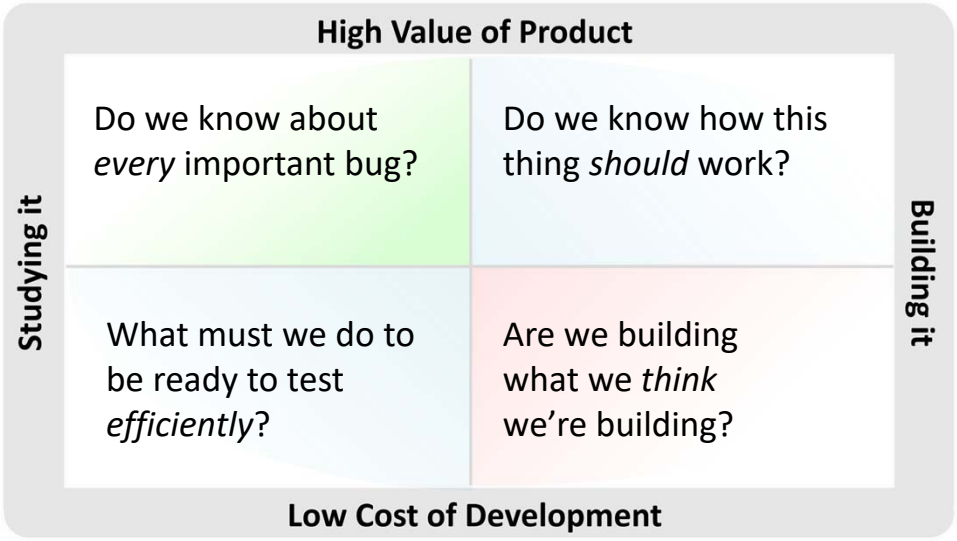
The Agile Development Cycle

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

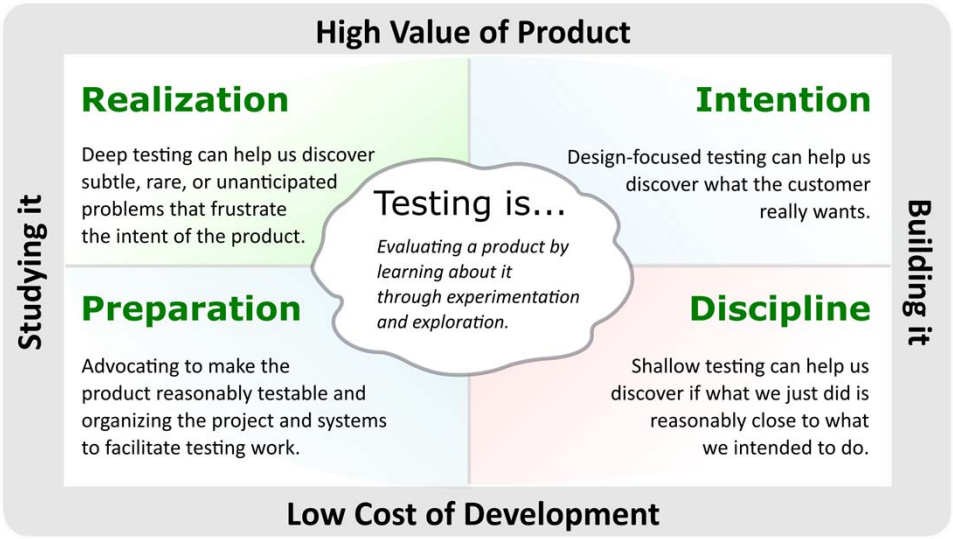


"Continuous attention to technical excellence and good design enhances agility."

Four Testing Questions

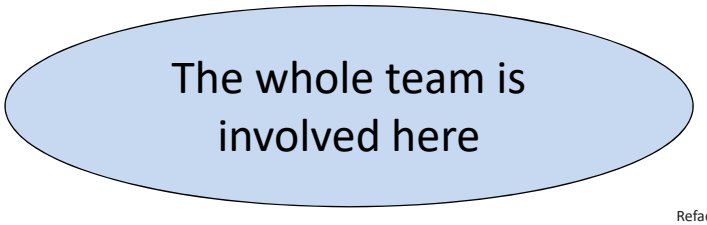


Four Frames for Testing



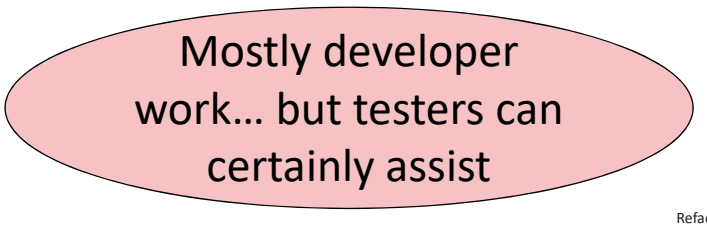
Intention: Developing the Design

- Establishing quality criteria
- Engaging with diverse users
- Specifying product *with* (not "by") rich examples
- Reviewing reports from the field
- Exploring design trade-offs
- Refining user stories



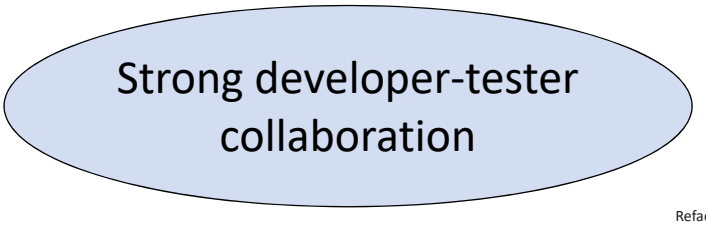
Discipline: Building Cleanly and Simply

- Automating low-level checks
- Establishing and adhering to a shared coding style
- Investigating and fixing bugs as we go
- Reviewing each other's code
- Integrating the product frequently
- Refactoring for maintainability



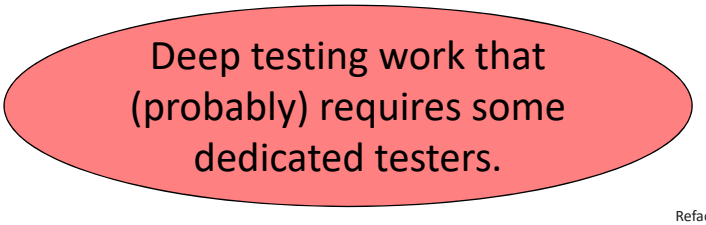
Preparation: Fostering Testability

- Preparing test environments and tools
- Designing for intrinsic testability
- Testing in parallel with coding
- Providing access to all levels of the product
- Minimizing trouble when changing the product
- Removing obstacles and distractions to testing

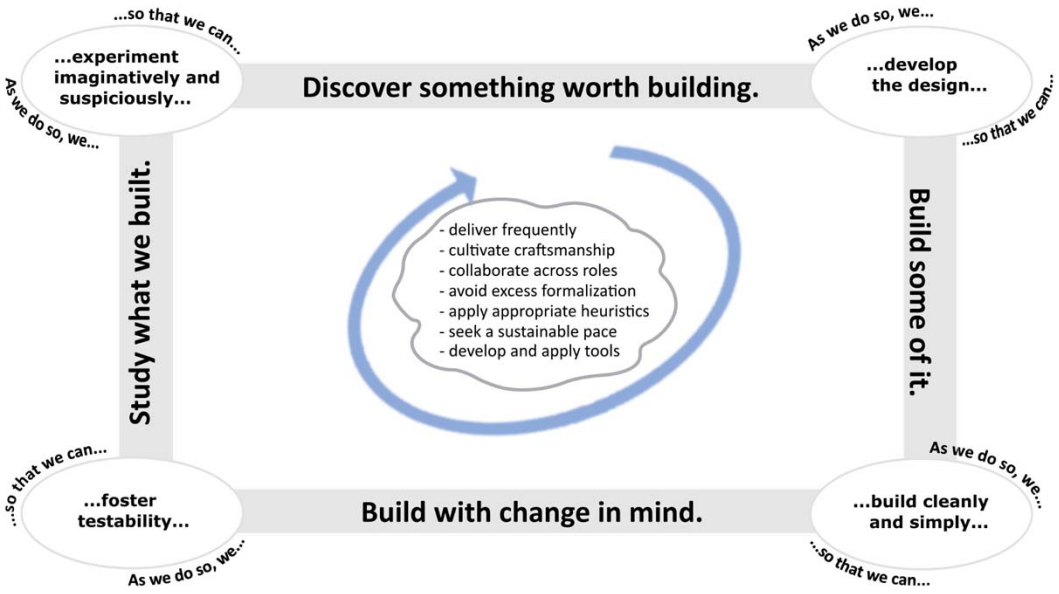


Realization: Experimenting Imaginatively and Suspiciously

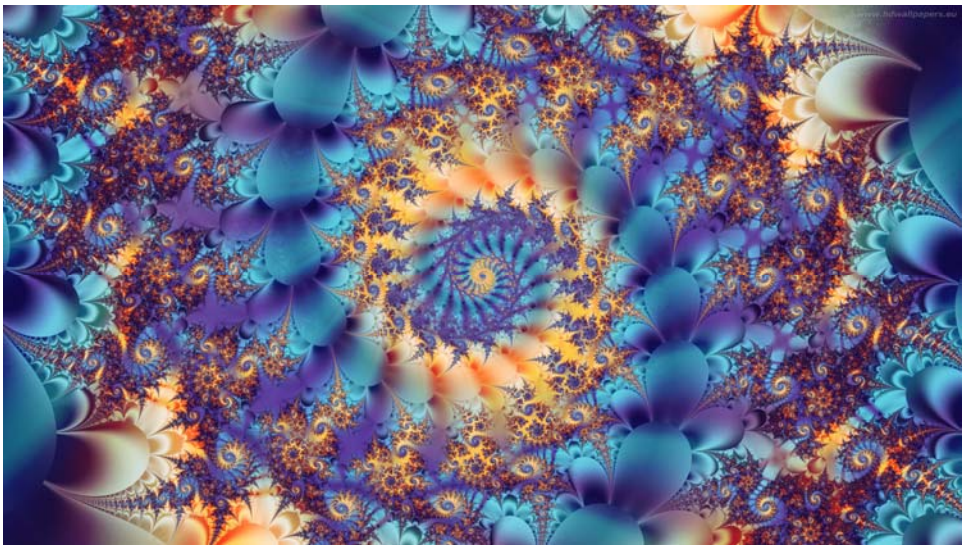
- Skeptically assessing whether we're "done"—or **not done *yet*.**
- Modelling systems in diverse ways—beyond Given, When, Then
- Developing rich test data—challenging the product
- Focusing testing and checking on suspected risk
- Investigating mysteries—aiding the developers
- Telling compelling bug stories—studying testing and risk



And although these dimensions have a roughly clockwise sequence…

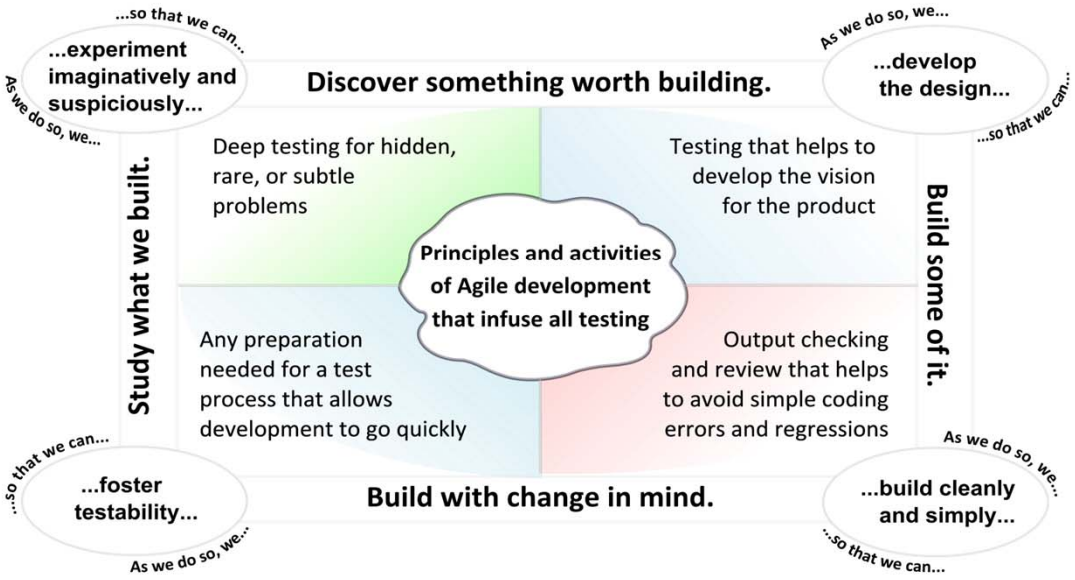


…development isn't linear. Not even just loopy.



Development is a fractal!

Testing is *woven into development*, at every level.



RST's Agile Quadrants in Detail