The Absolute Beginner's Guide to (Agile) Software Development (and Rapid Software Testing)

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)

Some people have strange ideas about software development.

They think of THIS as "efficiency"...

image credit: istockphoto.com

But they forget that that the product was DESIGNED.

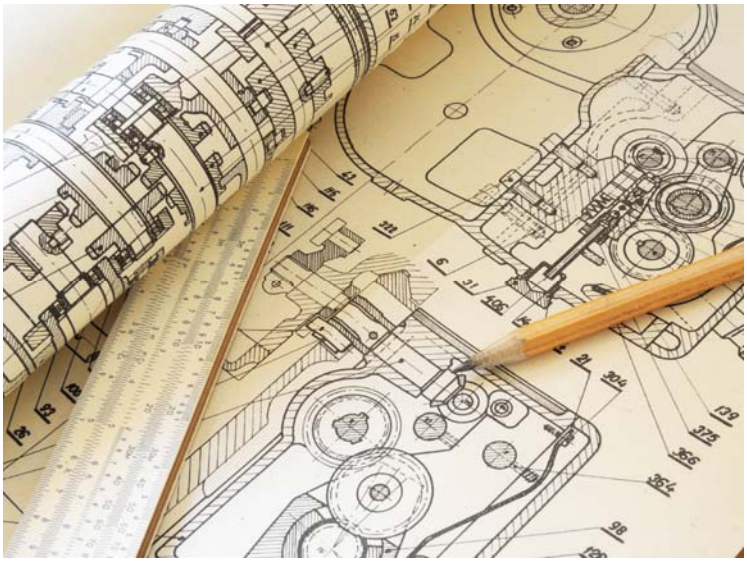


image credit: istockphoto.com

And they forget that the FACTORY had to be built.

image credit: istockphoto.com

AND they forget that the factory had to be DESIGNED.

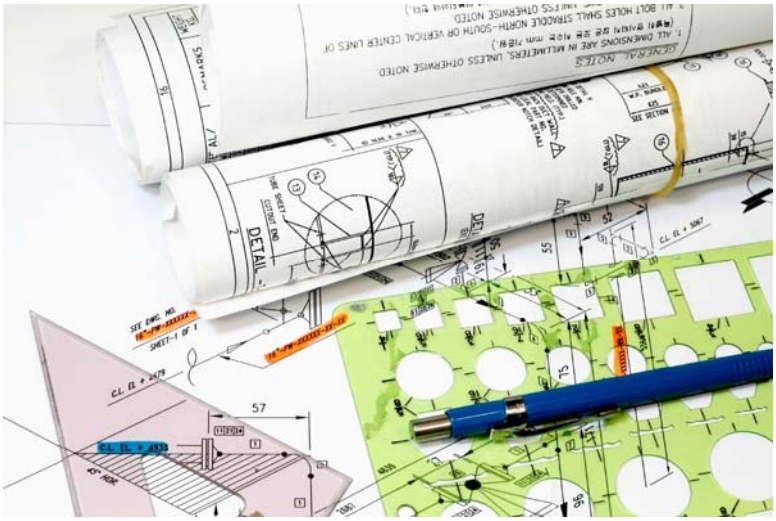


image credit: istockphoto.com

Plus: notice anything missing from all this?

They forget that PEOPLE did all this stuff.

Software development is not factory work.
It's not about the part where we make
a million copies of the same thing.

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

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

image credit: istockphoto.com

They forget that designs must be INTERPRETED.

image credit: istockphoto.com

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

image credit: istockphoto.com

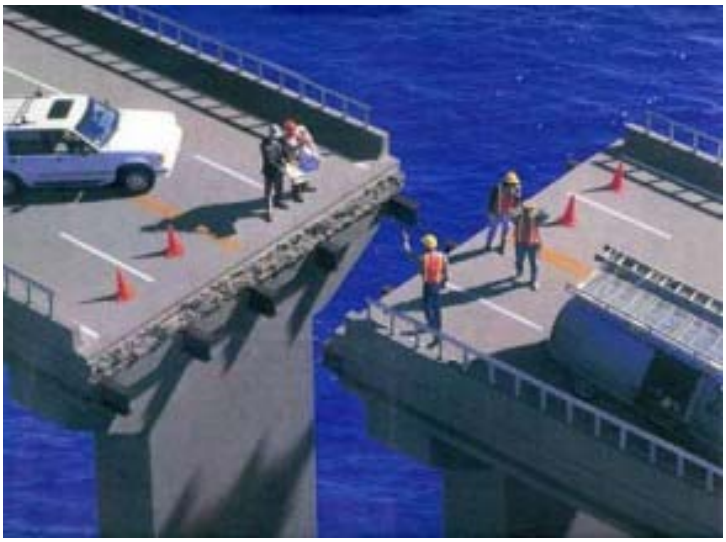
They forget that even building MODELS can be messy.

image credit: istockphoto.com

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

image credit: istockphoto.com

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



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

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

image credit: istockphoto.com

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

image credit: istockphoto.com

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

image credit: istockphoto.com

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

image credit: istockphoto.com

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

image credit: istockphoto.com

They become fixated on numbers and charts and dashboards.

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

image credit: istockphoto.com

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

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

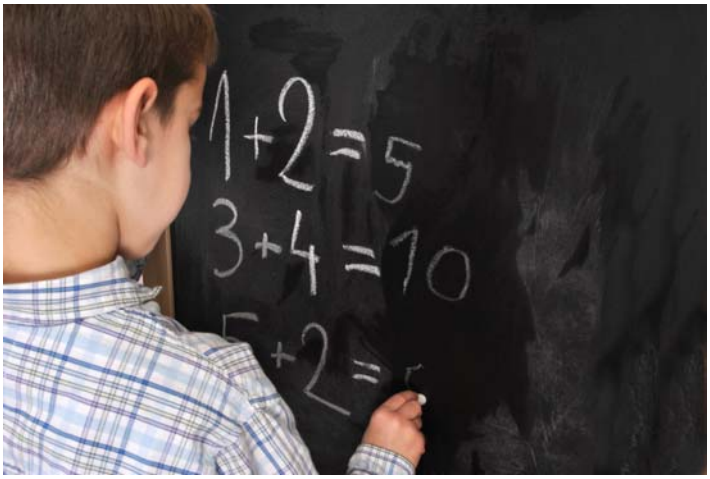


image credit: istockphoto.com

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

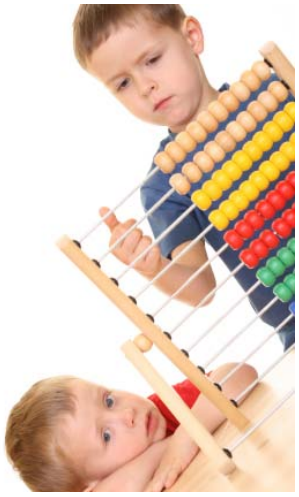


image credit: istockphoto.com

Do pilots use "piloting cases"?

image credit: istockphoto.com

Do parents use "parenting cases"?

image credit: istockphoto.com

Do scientists use "science cases"?

image credit: istockphoto.com

Do managers use "management cases"?

image credit: istockphoto.com

Then why do testers use test cases?

One Answer

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

But smart people can do MUCH better than that.

Premises of Rapid Testing

1. Software projects and products are relationships between people.
2. Each project occurs under conditions of uncertainty and time pressure.
3. Despite our best hopes and intentions, some degree of inexperience, carelessness, and incompetence is normal.
4. A test is an activity; it is performance, not artifacts.

35

Premises of Rapid Testing

5. 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.
6. We commit to performing credible, cost-effective testing, and we will inform our clients of anything that threatens that commitment.
7. We will not knowingly or negligently mislead our clients and colleagues or ourselves.
8. Testers accept responsibility for the quality of their work, although they cannot control the quality of the product.

36

The Themes of Rapid Testing

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

Important words!

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

Four Kinds of Risk Drivers

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

Bugs

Issues

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

Tell a Three-Part Testing Story

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…
…how you operated and observed it…
…how you recognized problems…
…what you have and *have not* tested *yet*…
…what you won't test *at all* (unless the client objects)…

Oracles

Coverage

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

image credit: istockphoto.com

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.

Call this "Checking" not Testing

operating a product to check specific facts about it…

means

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

Apply algorithmic decision rules to those observations.

Report any failed checks.

Three Parts to a Check

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

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

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

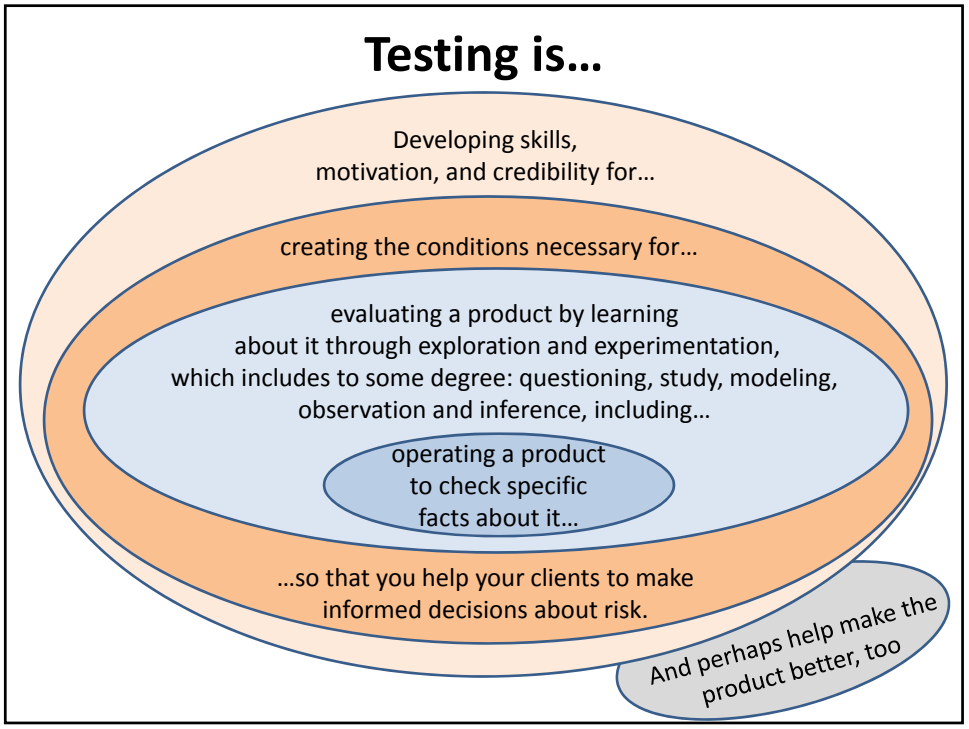
by a human who has been
programmed *not* to think
(and who is slow and variable)

image credit: istockphoto.com

Testing Is *More* Than Checking

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

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



Putting things together without learning about them is assembly, not development. Testing is intrinsic to development.

Harry Collins on Software Testing

"Computers and their software are two things. As collections of interacting cogs they must be 'checked' to make sure there are no missing teeth and the wheels spin together nicely. Machines are also 'social prostheses', fitting into social life where a human once fitted. It is a characteristic of medical prostheses, like replacement hearts, that they do not do exactly the same job as the thing they replace; the surrounding body compensates.

Abstract, "Machines as Social Prostheses", EuroSTAR 2013

Harry Collins on Software Testing

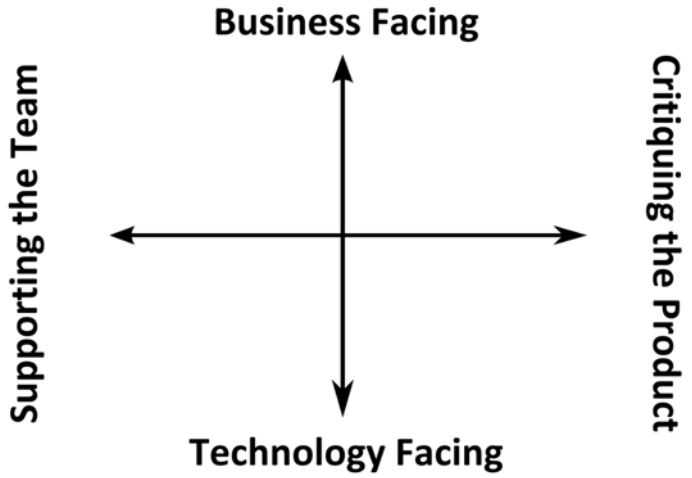
"Contemporary computers cannot do just the same thing as humans because they do not fit into society as humans do, so the surrounding society must compensate for the way the computer fails to reproduce what it replaces. This means that a complex judgment is needed to test whether software fits well enough for the surrounding humans to happily 'repair' the differences between humans and machines. This is much more than a matter of deciding whether the cogs spin right."

Abstract, "Machines as Social Prostheses", EuroSTAR 2013

Core Ideas of Agile Development

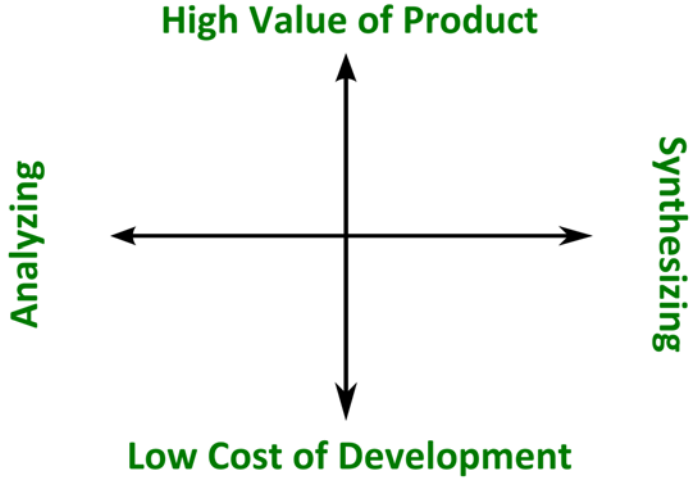
- "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."
  - so, keep refocusing on **increasing value**
  - and, in the testing role, refocus on *threats* to value
- "Welcome changing requirements, even late in development. Agile processes harness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

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



First, refactor those dimensions…

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



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

(This version avoids alienating professional testers and more directly addresses the tension between business and technology "facings.")

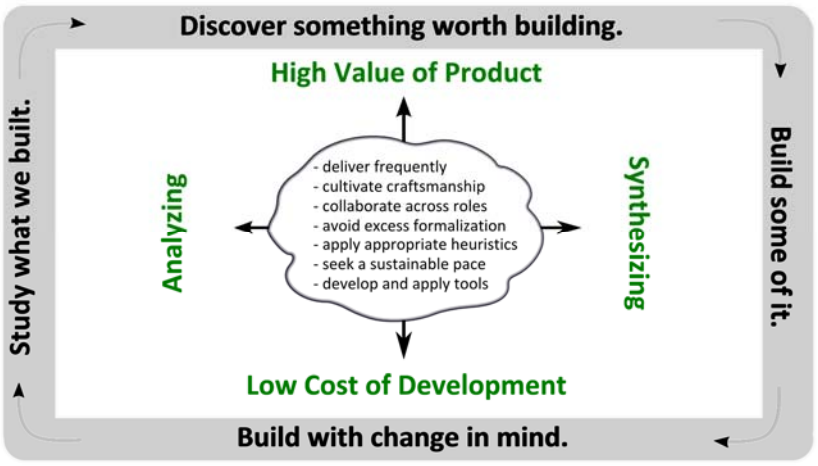
HOW do we do "Agile Development"?

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

What does it mean to do "Agile Development"?

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

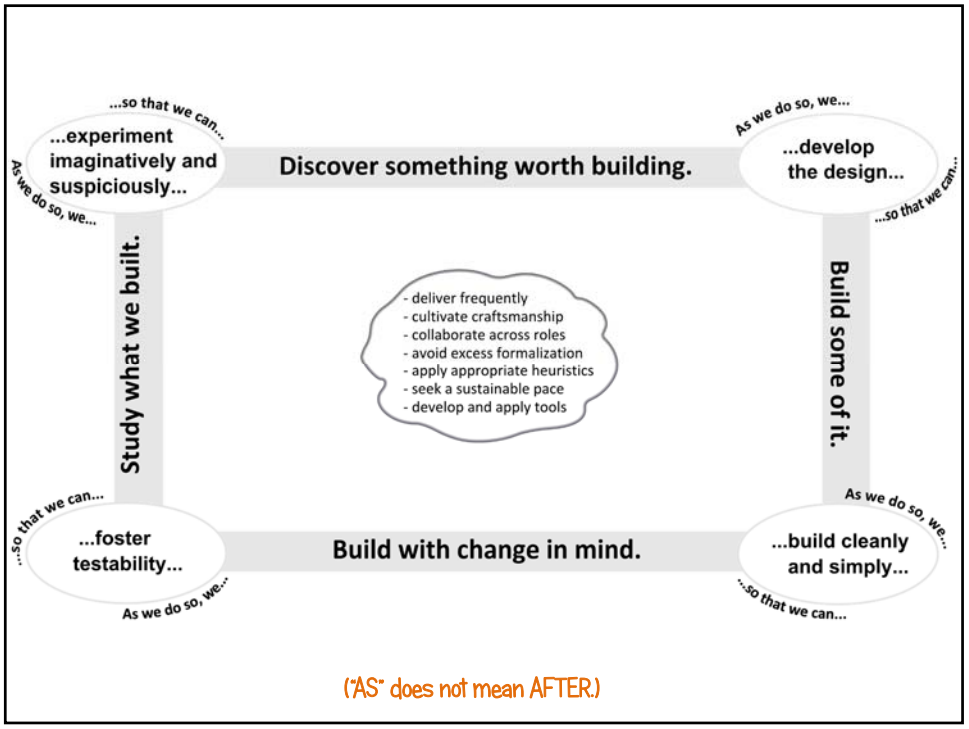
This suggests a clockwise cycle…



Important words!

- **Craftsmanship:** skill, expertise, mastery that is learned from knowledge, practice, and a culture
  - important for new designs and new problems
- **Heuristic**: a way of solving a problem that *often works* but that *might fail*; also, something that helps us to learn
  - we will often fail when we develop something new, but we only *really* fail when we don't learn
- **Excessive formalization**: something that is *too* fancy, or rigid, or controlled, or standardized
- **Sustainable pace**: not going so fast that we get into trouble

…and the corners representing enabling patterns.



("AS" does not mean AFTER.)

What is the testing role?

- *To test* is to evaluate a product by learning about it through exploration and experimentation.
- *A tester's role* is to develop himself as a tester, connect with the clients of testing, prepare for testing, perform testing, and report the results of testing.

Why is this the testing role?

- *Because that's the meaning and history of "testing".*
- Because to add quality policing or improvement to testing creates responsibility without authority; usurps management's role; and sets testers up as scapegoats.
- Because this is already a very challenging portfolio and skill set. Adding anything to it distracts and dilutes testing effort.

But wait! Does that mean that testers should ONLY look for bugs?

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.

Some roles are more permanent than others.

- If I am a father, do I HAVE to do fathering?

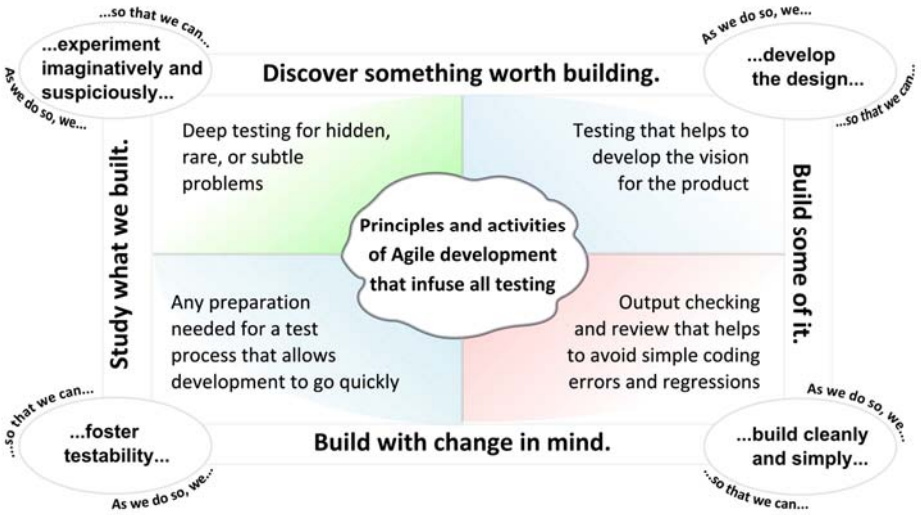
Well, yes.

Testing in an Agile Context

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

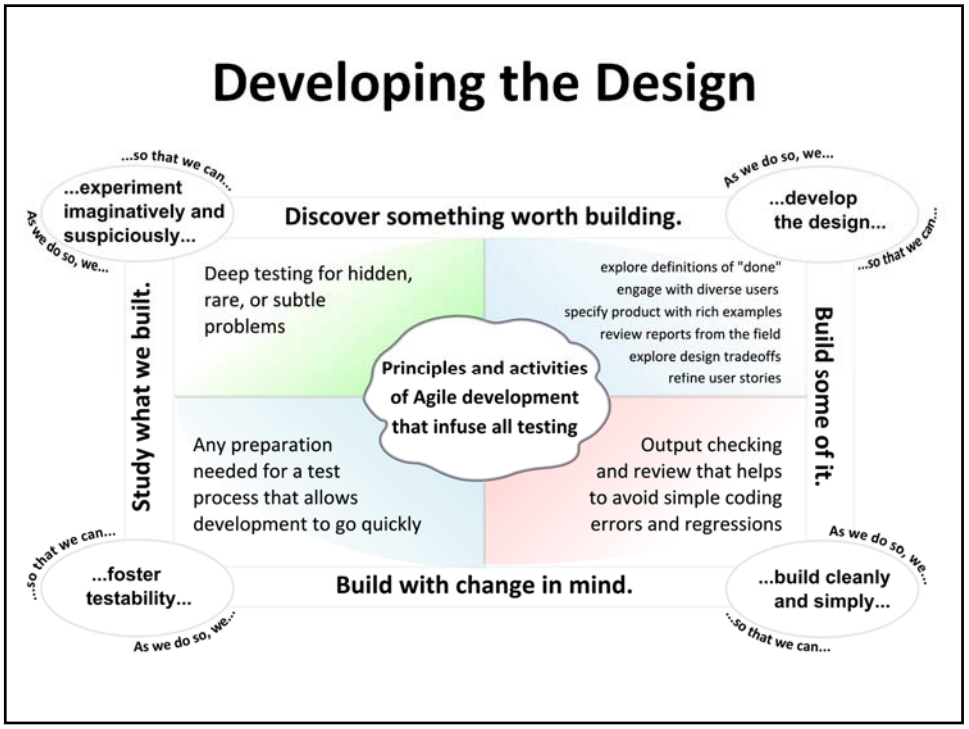
* See "Heuristics of Software Testability", http://www.satisfice.com/tools/testable.pdf

Testing in an Agile Context



Developing the Design

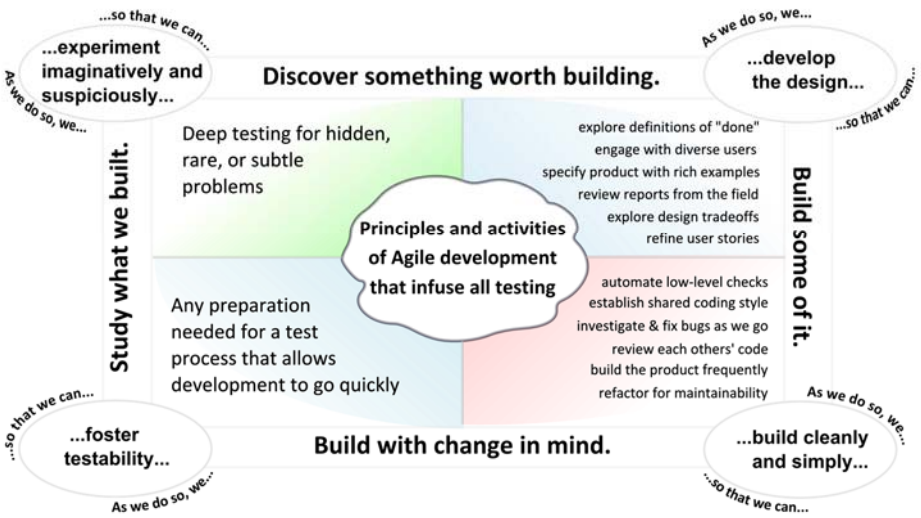
- Exploring definitions of done
  - we will change our definition of done when we learn more about the product and the project
- Work with diverse users
  - there are many different kinds of “user”
- Rich examples
  - examples ARE NOT tests, but examples DO help
- Review reports from the field
  - real-world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



Building Cleanly and Simply

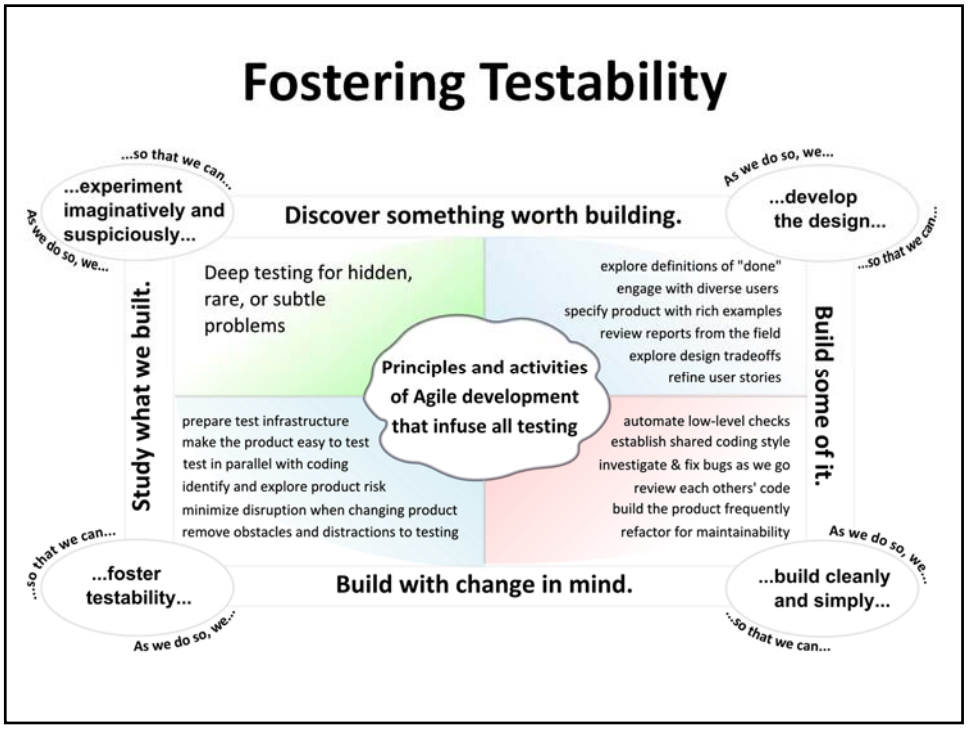
- Automate low-level checks
  - producing a much more testable product
- Establishing shared coding style
  - make interpretation faster and easier
- Reviewing each other's code
  - destroy bugs before they even get into the house
- Building the product frequently
  - make constant feedback easy
- Re-factoring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

Building with Change in Mind



Fostering Testability

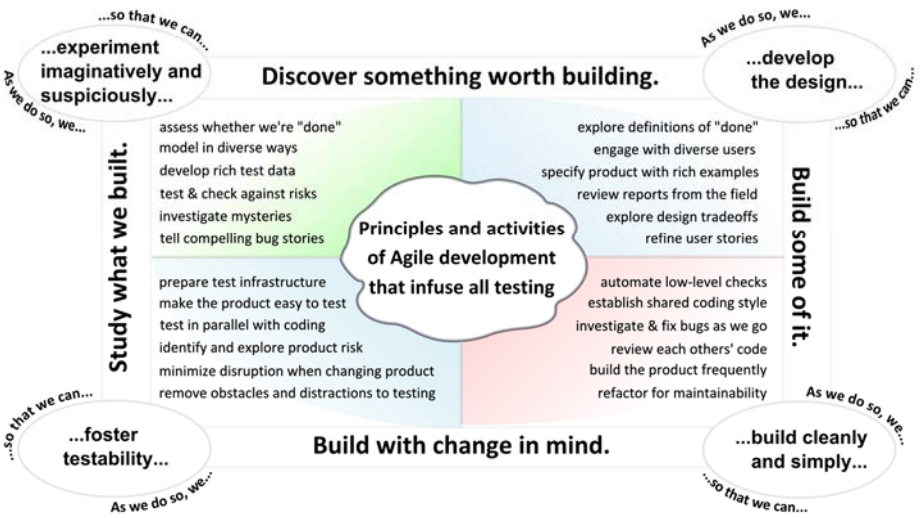
- Preparing test infrastructure
  - choose and develop tools and environments
- Making the product easy to test
  - ask developers for visibility and controllability
- Identifying and exploring product risk
  - digging up buried assumptions and problems
- Minimizing disruption when changing product
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



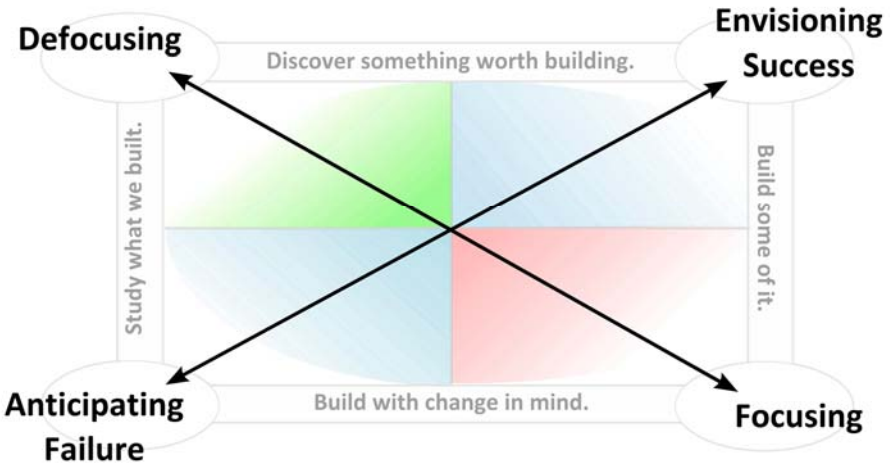
Experimenting Imaginatively and Suspiciously

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

RST's Agile Quadrants in Detail

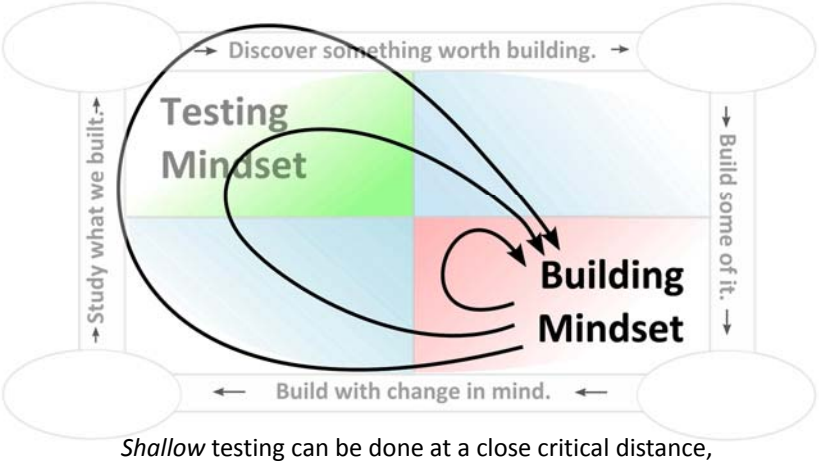


Helpful Polarities



Critical Distance

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



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

Trading Zone

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

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

Some Caveats