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.

---

**Is This Agile Testing?**

**9.8.1 To verify Power Accuracy**
- a. Connect the components according to the General Setup document.
- b. Power on and connect test jig (instead of electrodes)
- c. Power on the Zapper Box.
- d. Power on the Control Box.
- e. Set default settings of temperature and power for the Zapper Box.
- f. Set test jig load to nominal value
- g. Select nominal duration and nominal power setting
- h. Press the Start button
- i. Verify Zapper reports the power setting value ±10% on display.
Is This Agile Testing?

Scenario: Power Accuracy

**GIVEN** that the components are connected according to the General Setup document.

**AND** the Zapper Box is on.

**AND** the Control Box is on.

**WHEN** I set the default settings of temperature and power

**AND** I select nominal duration and nominal power setting

**AND** I press the Start button

**THEN** the Zapper should report the power setting value ±10% on display.

Is This Agile Testing?

George is now an employee at Spacely's Space Sprockets, a manufacturer of "sprockets" and other high tech equipment. His job title is "digital index operator."

George's job primarily requires him to **repeatedly push a single button** (or on occasion a series of buttons) on a computer. In one of the episodes, George complained of his heavy work load—having to **push a button for one hour, two days a week.**

—Wikipedia
Q. How I do Agile Testing?

Start with this: testing is testing. Agile is context.

Thank you, Scott Barber.

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

Business Facing

Support Programming

Critique Product

Technology Facing


Crispin & Gregory’s (Earlier) Version

Agile Testing Quadrants

Automated & Manual

Functional Tests
Examples
Story Tests
Prototypes
Simulations

Q2

Q3

Q4

Exploratory Testing
Scenarios
Usability Testing
UA7 (User Acceptance Testing)
Alpha / Beta

Manual

Critique Product

Supporting the Team

Automated

Unit Tests
Component Tests

Performance & Load Testing
Security Testing
“My” Testing

Tools

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
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?

Why you might like the quadrants:

Because they represent a generic diversified test strategy! This is a Good Thing. But after 10 years or so of learning and experience with them, we can refactor.
Let’s refactor those in terms of what the business wants.

This is how the business plans to keep customers happy and make money.

This helps make the business sustainable.

This is how the business gets something it 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!
“Traditional” Development Cycle

Discover something worth building.

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.

Get it right the first time.

Study what we built.

Build some of it.

Refactoring the Agile Testing Quadrants - 23

Plus: testing as an assembly line?

Understand Context (TP1)

Test Planning Process

From ISO 29119

Still partying like it’s 1999!

Brought to you by certificationists!
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

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

http://www.agilemanifesto.org
Agile Development Cycle

Discover something worth building.

Discover problems and fix them right away.

Lots of short loops so we don’t get ahead of ourselves.

Almost NONE of it.

Recognize that we’ll make SOME mistakes.

Build with change in mind.

Build some of it.

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. (At the time, this was much more a feature, and much less a bug.)
• An emergent 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.
• As we’re developing the product, we don’t know where those problems are... and that’s where risk lives.
Call This “Checking”, Not Testing

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*.

Think “spelling checker”

means

Observe
Evaluate
Report

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” here 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 understand our products and the risk of problems that matter to people, we must do more than output checking; we must test.

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 mechanistically to check specific facts about it...
- ...to help our clients to make informed decisions about risk.

Why it’s important to distinguish testing and checking

• Because checking is mechanistic. It can be made completely explicit and automated. It is inside testing. It is a tactic of testing.

• Because testing involves tacit and social skills that cannot be encoded. Testing skills and must be developed through socialization, practice, and increasingly challenging work, not via rote procedures.

• Because talk about efficiency and effectiveness makes for very different conversations when we’re talking about explicit vs. tacit skills.

• Because for checking to be truly excellent, it must be embedded in excellent testing. Developing valuable checks requires skill!

• Programmers have resisted marginalization for years! (They no longer call compilers “autocoders” and programming languages are no longer called “autocodes”.)
Testing is not about correctness, or passing or failing test cases. Skilled testers focus on two questions:

To themselves, referring to the product: Is there a problem here?

To the team and to management, referring to issues and obstacles: Are you okay with this?

What’s Unique About Real Testing Specialists?

We test more deeply and reliably.

(Why? Because we aspire to do so; we enjoy doing it; and we learn how.)
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."

- **Defocusing**
  - Discover something worth building.
  - Study what we built.
- **Envisioning Success**
  - Build some of it.
- **Anticipating Failure**
  - Build with change in mind.
- **Focusing**
  - "Continuous attention to technical excellence and good design enhances agility."
### Four Testing Questions

<table>
<thead>
<tr>
<th>Studying it</th>
<th>Building it</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do we know about <em>every</em> important bug?</td>
<td>Do we know how this thing <em>should</em> work?</td>
</tr>
<tr>
<td>What must we do to be ready to test <em>efficiently</em>?</td>
<td>Are we building what we <em>think</em> we’re building?</td>
</tr>
</tbody>
</table>

### Four Frames for Testing

<table>
<thead>
<tr>
<th>High Value of Product</th>
<th>Low Cost of Development</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Realization</strong></td>
<td><strong>Intention</strong></td>
</tr>
<tr>
<td>Deep testing can help us discover subtle, rare, or unanticipated problems that frustrate the intent of the product.</td>
<td>Design-focused testing can help us discover what the customer really wants.</td>
</tr>
<tr>
<td><strong>Preparation</strong></td>
<td><strong>Discipline</strong></td>
</tr>
<tr>
<td>Advocating to make the product reasonably testable and organizing the project and systems to facilitate testing work.</td>
<td>Shallow testing can help us discover if what we just did is reasonably close to what we intended to do.</td>
</tr>
</tbody>
</table>
**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

The whole team is involved here

---

**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

Mostly developer work... but testers can certainly assist
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

Strong developer-tester collaboration

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

Deep testing work that (probably) requires some dedicated testers.
And although these dimensions have a roughly clockwise sequence...

...so that we can...

...experiment imaginatively and suspiciously...

As we do so, we...

Discover something worth building.

...develop the design...

...so that we can...

Build some of it.

...build cleanly and simply...

As we do so, we...

Build with change in mind.

...so that we can...

...foster testability...

As we do so, we...

Study what we built.

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

RST’s Agile Quadrants in Detail
Q. I’m a manual tester. How do I fit in on an Agile team?

A. You’re not a manual tester.
   He’s not a manual programmer, either.
   And she’s not a manual product owner.

Testing is neither manual nor automated!
Q. I mean, I’m not a programmer. How am I suppose to fit in? (They want me to automate the test cases, you see.)
**Why to resist framing testing as test cases:**

- Testing is about exploration, experimentation, discovery, investigation, learning, and reporting.
- Test cases *tend to* focus on output checking, confirmation and demonstration; showing that something *works*, rather than prompting a search for problems that matter.
- Plus, when people turn testing into test cases, they start counting them.
- When people turn testing into *counting*, rather than assessment, the information loss, dysfunction and distortion begin.

---

**Testing Is *Social Science***

“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.

Testing Is Social Science

“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.”


In other words...

Is the product we’ve got good enough for people to be happy with it?
Reminder:
Management wants
an expert answer to this question:

Are there problems
that threaten the value of the product,
or the on-time successful completion of the project?

To TEST is to develop 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.
Better Talking and Thinking about Testing

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 misuse cases AND abuse cases
AND obtuse cases...
KPIs and KLOCs Learning from every bug

Talking More Clearly About Testing

Try replacing... with...
“The environment’s down. We’re stuck. We can’t test.” “What can we test, review, or analyze now... and are you OK with this situation, dear client?”
“They didn’t give us good requirements documents!” “Let’s write down what we know—and then they’ll tell us when they think it’s wrong.”
“It’s too hard to test this!” “What can we do in the product and the project to things more testable?”
“We don’t have enough time to test!” “What testing shall we do—what shall we cover—in the time we do have?”
“We have to...!” “We choose to...”
Q. What am I supposed to do at those planning meetings?

Learn.
Ask questions.
Ask for testability.
Challenge reckless and risky assumptions.
Establish your role as a tester.
Q. What questions should I ask in planning meetings?

What are we building?
For whom are we building it?
What could go wrong?
When something goes wrong, how would we know (quickly)?

*Focus on risk.*

---

Q. How should I help the developers?

Be careful not to *inflict* help on them.
Ask how you can help them.
Offer to test anything they’re willing to give you right away.
Offer to help with investigating mysteries.
Be a service, not a bottleneck.
Q. You forgot my question.
I’m not a programmer.
How do I fit in?

You can learn to code reasonably well; it’s not *that* hard.
Or you can be charming to people who can code.
But your role as a tester is to *test*.

---

Q. But… there aren’t supposed to be roles on an Agile team!

There doesn’t have to be somebody *called* tester.
But if you are producing something important or risky,
there had better be people *in a testing role*
to *perform testing* and to *tell the testing story*. 
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 *testing* role.
- A tester’s role is to
  - to develop one’s self as a tester
  - connect with the clients of testing
  - prepare for testing and to advocate for testability
  - perform testing
  - aid supporting testers in performing testing
  - report the results of testing.

The Testing Role Heuristic

The testing specialist role is a heuristic for *developing, operating, and expressing good testing mentality* across the organization.

If testing is nobody’s focus, it will not be anybody’s competence.
Why not just eliminate roles?
You have probably experienced this.

Why do some people think of a role as a prison or a fortress?

This happens when “role” is defined as the only things you do and what no one else does.

HIGH SOCIAL DISTANCE
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

What a Role Is NOT

• NOT a declaration of the only things you are allowed to do (not a prison)
• NOT a declaration of the things that you and you only are allowed to do (not a fortress)
• NOT permanent and unchanging
• NOT like a tattoo
We prefer to think of roles this way.

A role is like a villa. It is a semi-private space. Someone dwells in it. Someone is responsible. But visitors may come and help.

- Developers help testers.
- Testers help developers.
- But testers are RESPONSIBLE and ACCOUNTABLE for testing.

FLEXIBLE SOCIAL DISTANCE
USEFUL CRITICAL DISTANCE

Deeper testing, more like users, or targeted towards more diversified risk, tends to require or create more distance from the builder’s mindset.

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

Shallow testing is easy at close critical distance.

Discover something worth building; build some of it.

Build with change in mind.

Study what we build.

Testing Mindset

Building Mindset
Why roles? Because changing mindsets is HARD.

NOTE: We do NOT claim that different kinds of work must be done by different people, or that the people must have different titles.

We DO claim that having skilled people focused on testing is a powerful heuristic for addressing the mindset switching problem.

A Healthy Role Institutionalizes...

• **Commitment:** Making sure important work gets done.
• **Competence:** Increasing skill over time.
• **Focus:** Marshalling energy and concentration to solve difficult problems; taking advantage of economies of scale.
• **Anticipation:** Identifying future needs and potential problems when they’re cheap and easy to deal with, before it’s too late.
• **Accountability:** Accepting responsibility for outcomes within scope of the role.
• **Coordination:** Minding the interfaces with other roles.

See “On a Role” http://www.developsense.com/blog/2015/06/on-a-role/
Career Path: Test Jumper!

- may spend weeks on one project, acting as an ordinary responsible tester.
- may spend a few days on one project, organizing and leading testing events, coaching people, and helping to evaluate the results.
- may spend as little as 90 minutes on one project, reviewing a test strategy and giving suggestions to a local tester or developer.
- may attend a sprint planning meeting to assure that testing issues are discussed.
- may design, write, or configure a tool to help perform a certain special kind of testing.
- may coach another tester about how to create a test strategy, use a tool, or otherwise learn to be a better tester.
- may make sense of test coverage.
- may work with designers to foster better testability in the product.
- may help improve relations between testers and developers, or if there are no other testers help the developers think productively about testing.

Test Jumper Requirements

- Ability and enthusiasm for plunging in and doing testing right now.
- Ability to defocus out of specific test tasks and see the big picture.
- Ability to **recruit, coach and train testers and helpers**.
- A wide knowledge of tools, and ability to develop tools.
- A good respectful relationship with developers.
- Ability to speak up in planning meetings about testing-related issues.
- A keen understanding of testability, and how to advocate for it.
- Ability to lead ad hoc groups of people with challenging personalities during occasional test events.
- Ability to speak in front of people and product useful and **concise documentation** as necessary.
- Ability to manage many threads of work at once.
- Ability to evaluate, describe, and explain testing in general and in particular forms.
Q. But... but... what about regression?

Consistent Regression Problems Are **Symptoms of Trouble**

If you see a consistent pattern of regression
- failing checks or tests probably aren’t your biggest problem
- more likely, the issue is that you’ve got favourable conditions for regression to happen
- testing cannot fix this problem; at best testing can only detect some regression bugs
- the programmers are probably working too quickly to understand what’s happening

**THAT’S A SEVERITY-ONE ISSUE. REPORT IT.**
Q. There’s too much work! We can’t meet the “definition of done” by the end of the sprint!

Congratulations; that’s a test result. Tell the testing story. If your team is committing to get something specific done by the end of the sprint, I’d say you’re not on an Agile team. The only sane commitment is to try to get things done, and have a retrospective on what happens.

Foreseeable Problems

- Problem:
  - Programming time and testing time are asymmetrical.
  - Deep testing may take time, effort, tools, skills to perform.
- Answers:
  - Focus on testability.
  - DO treat sprints as appointments for a review.
  - DON’T treat sprints as commitment to finishing something, but as commitments to try finishing them—and to see what we learn in the process.
  - Notice when pace becomes unsustainable.
Foreseeable Problems

- Problem:
  - Agile (and especially) DevOps means that we’re getting new builds all the time—no time to test!

- Answers:
  - “No time to test” literally means “no time to study the product and investigate it for potential problems”. Is everybody OK with that?
  - DO treat continuous integration as an opportunity to do quick checks on incremental builds and deep testing on periodic stable builds and.
  - DON’T let frequent, frantic building mislead you into thinking that a checked product is a tested product.

Q. How should we measure quality?

You can measure attributes that might have a bearing on quality, but you can’t measure quality itself.

You can’t measure quality.

But you can report on it, and you can discuss it.
Technical Suggestions for Testers

• Resist test cases and scripts; focus on test activities and on telling the testing story.
• Let risk guide your testing.
• 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!

Technical Suggestions About Tools

• DON’T use them to “do” the testing. Tools don’t do testing; YOU do testing, and tools help you to do that.
• DON’T become fixated on tools.
• DO prefer lightweight tools, in general.
• DO use tools 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 for Testers

- Practice explaining testing.
- Declare your role and commitments.
- Don’t accept responsibility for the quality of the product.
- Embed yourself in 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.

Some Caveats
Q. Can’t I focus on preventing problems instead of finding problems?

Sure. But when you’re doing that, you are (at least temporarily) abandoning the testing role, and adopting a building role.

Development is about preventing problems. Testing is about discovering problems development didn’t prevent.

Early in development, testers can help by anticipating risks and problems in a way that helps developers and others to prevent them.
**Why it’s important for testers NOT to say “I prevent problems”**

As a tester, when you say “I prevent problems”, there’s a strong possibility you are over-reaching.

- It’s important to honour the roles of the developers and the designers, and to remain humble about our own. We don’t put the quality in directly. They do.
- Problems occur because of a variety of contributing factors. The same is true of the prevention of problems. It’s a team, right?
- It’s perfectly okay to say “I help people to prevent problems.” We’re here to help! We help!
- Plus... how will you KNOW that you’ve prevented problems successfully unless you try to find them?

**Why it’s important NOT to say “I break the software”**

When you say “I break the software”, you set yourself up for a potential public relations problem. Others may “repair” “I break the software.”

- “The software was fine until the testers broke it.”
- “We could ship our wonderful product on time if only the testers would stop breaking it.”
- “Normal customers wouldn’t have problems our wonderful product; it’s just that the testers break it.”
- “There are no systemic management or development problems that have been leading to problems in the product. Nuh-uh. No way. The testers broke it.”
Alternatives to Test Cases: Parallel Oracles in Tables

Alternatives to Test Cases: Coverage Outlines
Alternatives to Test Cases: Open Procedures

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 on each box and VERIFY conformance to the requirements specifications. Remain alert for the presence of any behaviour or attribute that could mislead or confuse the operator, or 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.

Alternatives to Test Cases: More Specific Procedures

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

3.5.2.8 The equation for calculating the tolerance interval $k$ is:

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

where $\chi^2_{N-1, \gamma}$ is the critical value of the chi-square distribution with degrees of freedom, $N-1$, that is exceeded with probability $\gamma$ 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.
Alternatives to Test Cases: Concise Learning 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 SFDIPO 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.”

Alternatives to Test Cases: Concise Testing Charters

• ...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.”
Alternatives to Test Cases: More Formalized Charters

PROCHAIN ENTERPRISE

SCENARIO TEST CHARTER

UP1: “Check tasks and update”

<table>
<thead>
<tr>
<th>Theme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>You are an individual contributor on a project. You have tasks assigned to you. Check your tasks and update them. Check the status of tasks that gate the ones you are responsible for.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Setup</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assure that your user account(s) are set up with rights to access a project that has many tasks assigned to it.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Go to Tasks panel and filter tasks for ones assigned to you. (Alternatively, filter in other ways such as by project or by incomplete tasks, and choose a way to sort.)</td>
<td></td>
</tr>
<tr>
<td>Select one of the task list views and view each task. Set the task filter to show at least: actual start, total duration, and remaining duration.</td>
<td></td>
</tr>
<tr>
<td>For some tasks, view details, checklists, and attachments.</td>
<td></td>
</tr>
<tr>
<td>Update each task in some way, including:</td>
<td></td>
</tr>
<tr>
<td>- No update</td>
<td></td>
</tr>
<tr>
<td>- “Mark as Updated”</td>
<td></td>
</tr>
<tr>
<td>- Shorten duration remaining</td>
<td></td>
</tr>
<tr>
<td>- Set remaining duration to zero, or “Mark as Completed”</td>
<td></td>
</tr>
<tr>
<td>- Increase duration remaining</td>
<td></td>
</tr>
<tr>
<td>- Provide comments, update checklist</td>
<td></td>
</tr>
<tr>
<td>- Undo some updates</td>
<td></td>
</tr>
<tr>
<td>Refilter to see more tasks. Find tasks that feed into or lead from your tasks. Update some of those.</td>
<td></td>
</tr>
</tbody>
</table>

Alternatives to Test-Case-Based Reporting:

Session-Based Test Management Debriefs: PROOF!

- **Past**
  - What happened during the session?
- **Results**
  - What was achieved? What was covered?
- **Obstacles**
  - What got in the way or slowed things down?
- **Outlook**
  - What’s next? What remains to be done?
- **Feelings**
  - How does the tester feel about all this?
“They” will forget about empty “test case” documentation when we give them something more credible.