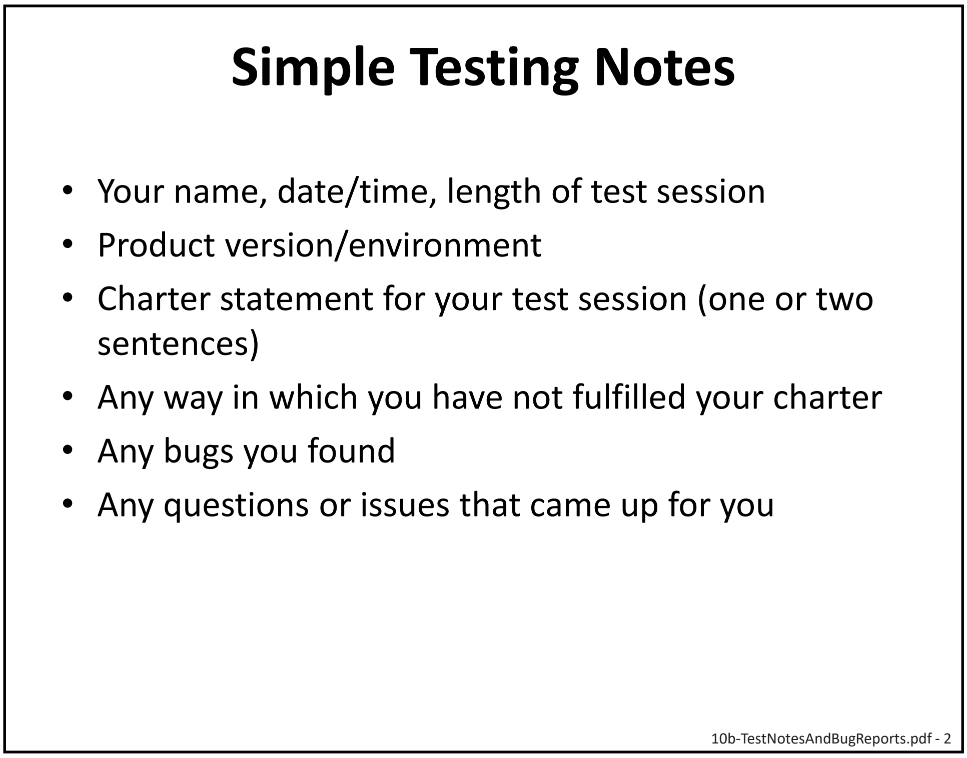
RST Recording and Reporting

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

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

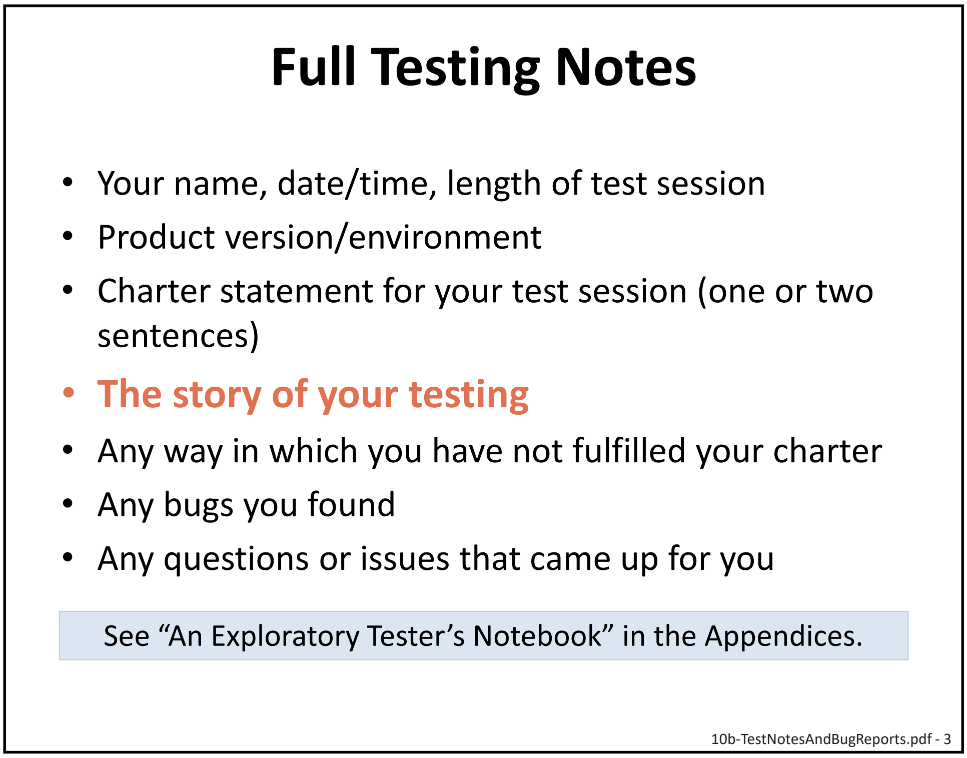
RST Recording and Reporting

How do you record your work?
How do you report bugs?
How do you report test status?



Simple Testing Notes

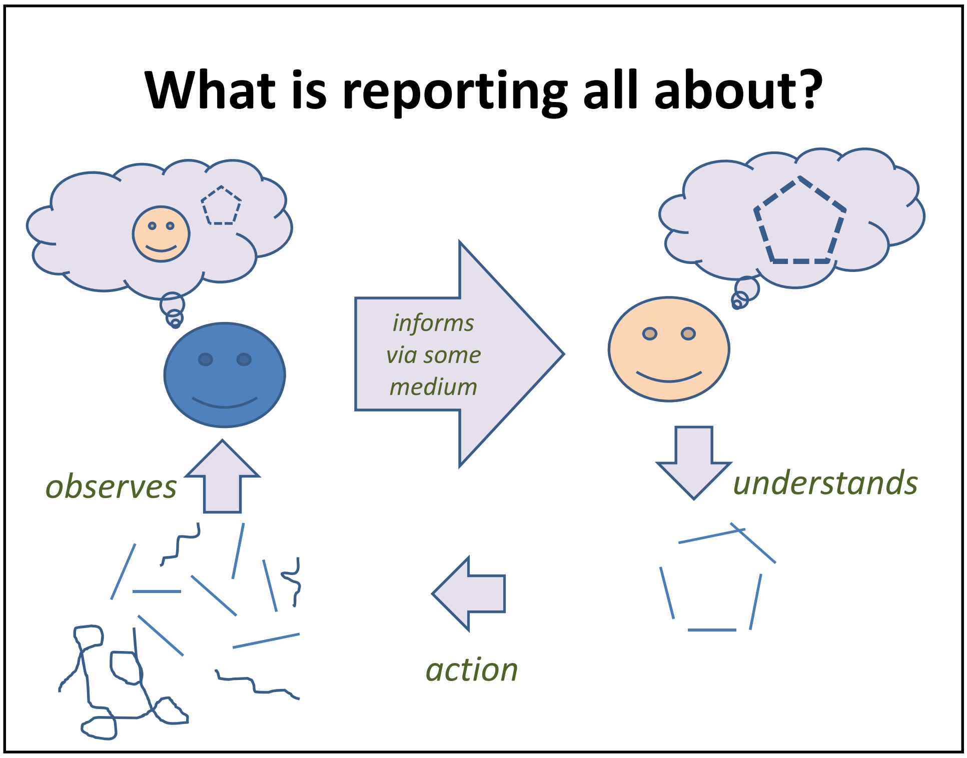
- Your name, date/time, length of test session
- Product version/environment
- Charter statement for your test session (one or two sentences)
- Any way in which you have not fulfilled your charter
- Any bugs you found
- Any questions or issues that came up for you



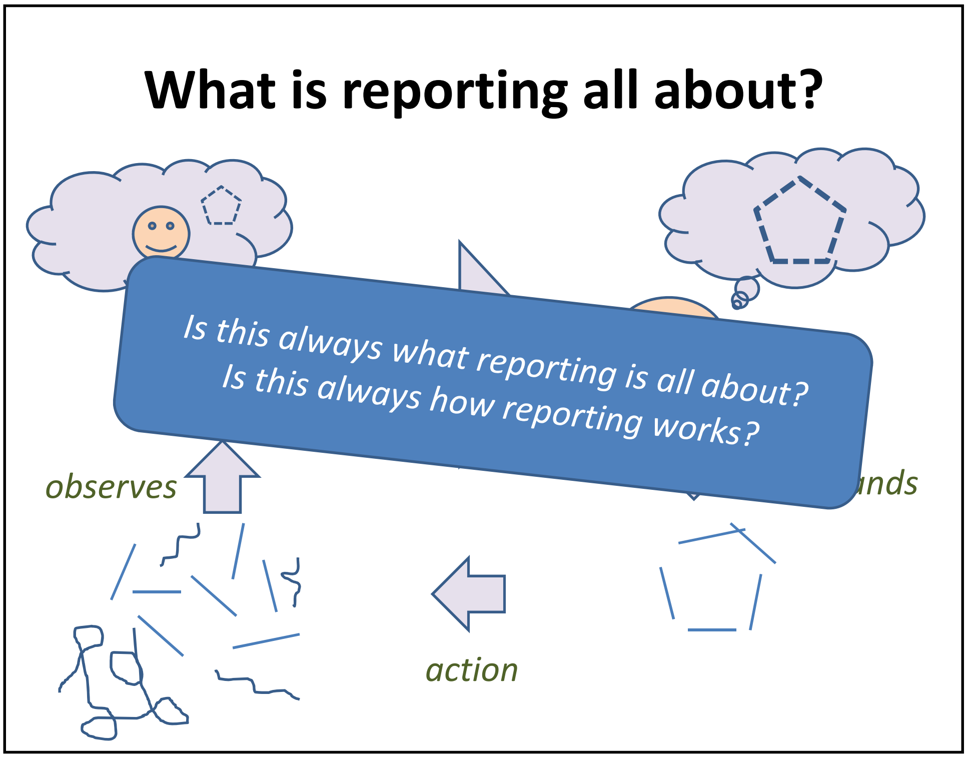
Full Testing Notes

- Your name, date/time, length of test session
- Product version/environment
- Charter statement for your test session (one or two sentences)
- **The story of your testing**
- Any way in which you have not fulfilled your charter
- Any bugs you found
- Any questions or issues that came up for you

See “An Exploratory Tester’s Notebook” in the Appendices.

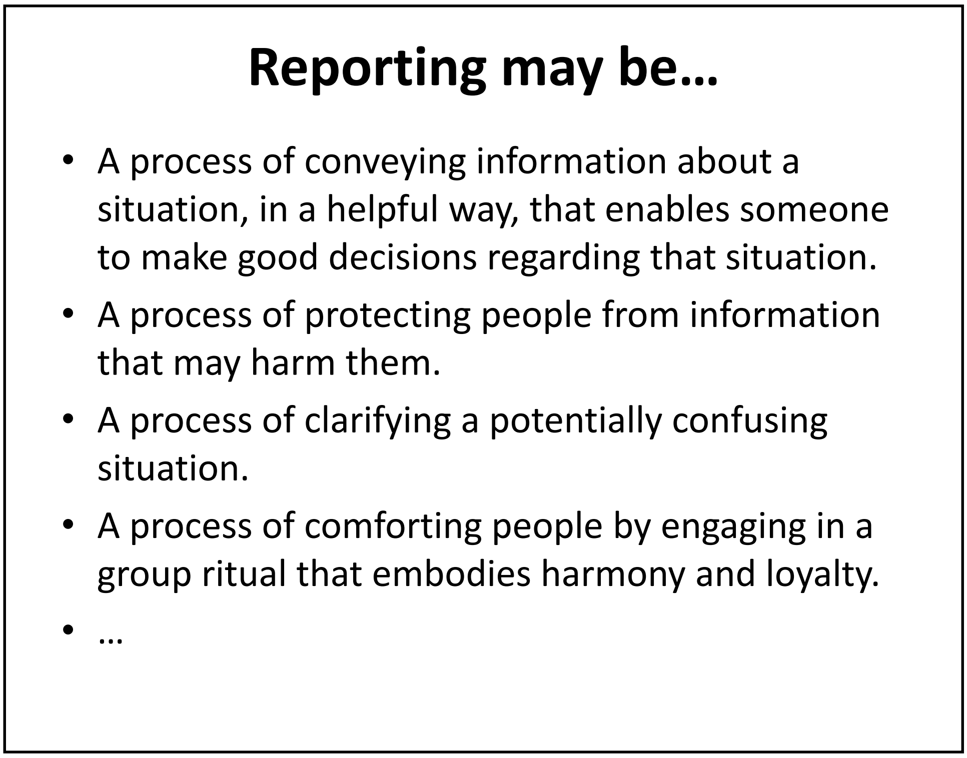


What is reporting all about?



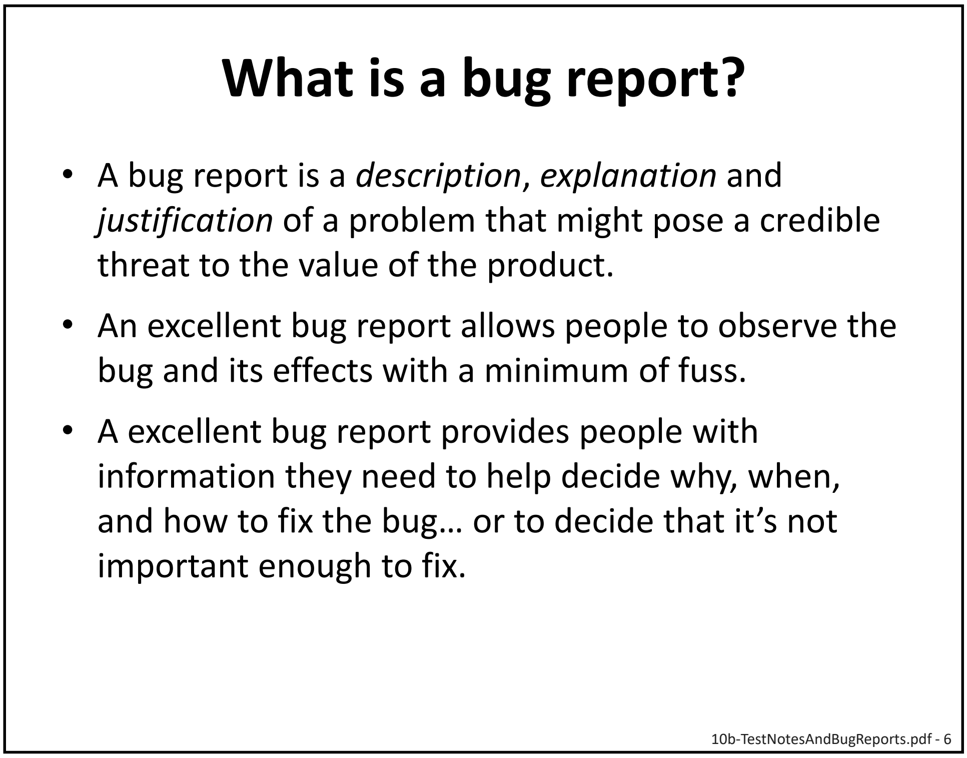
What is reporting all about?

Is this always what reporting is all about?
Is this always how reporting works?



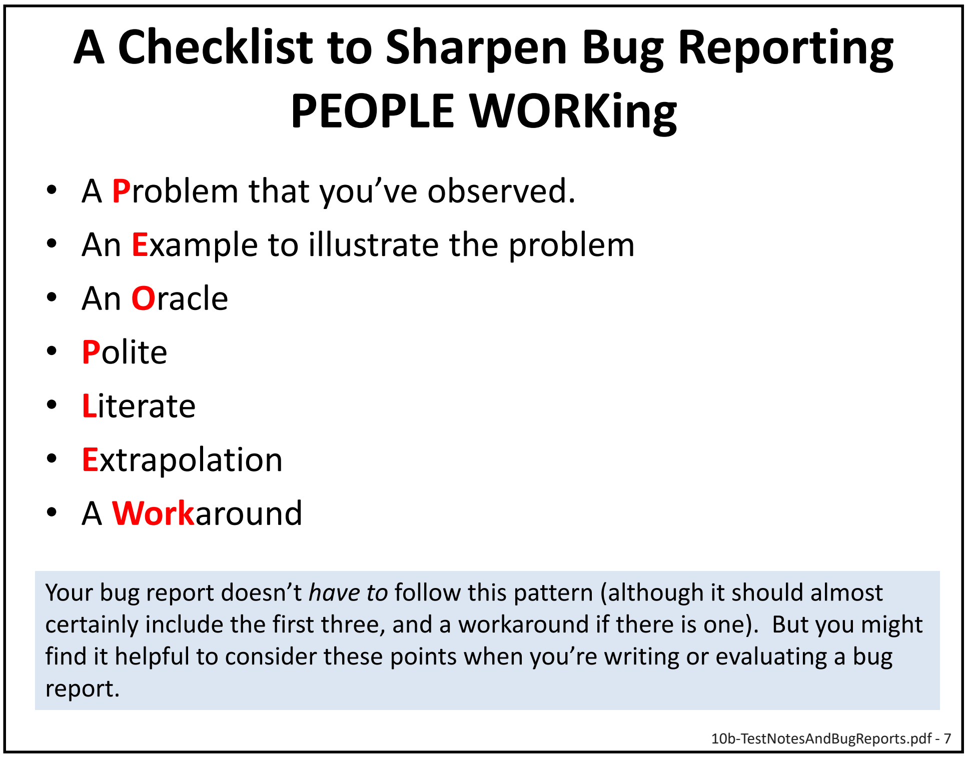
Reporting may be…

- A process of conveying information about a situation, in a helpful way, that enables someone to make good decisions regarding that situation.
- A process of protecting people from information that may harm them.
- A process of clarifying a potentially confusing situation.
- A process of comforting people by engaging in a group ritual that embodies harmony and loyalty.
- …



What is a bug report?

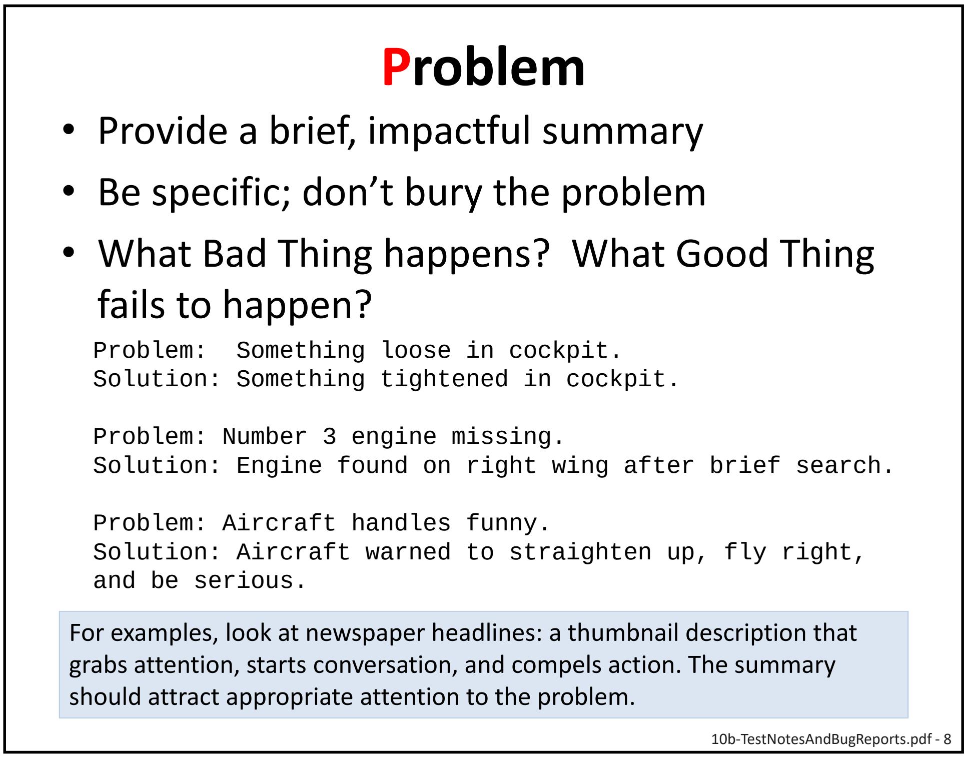
- A bug report is a *description*, *explanation* and *justification* of a problem that might pose a credible threat to the value of the product.
- An excellent bug report allows people to observe the bug and its effects with a minimum of fuss.
- A excellent bug report provides people with information they need to help decide why, when, and how to fix the bug… or to decide that it’s not important enough to fix.



A Checklist to Sharpen Bug Reporting PEOPLE WORKing

- A **P**roblem that you've observed.
- An **E**xample to illustrate the problem
- An **O**racle
- **P**olite
- **L**iterate
- **E**xtrapolation
- A **Work**around

Your bug report doesn't *have to* follow this pattern (although it should almost certainly include the first three, and a workaround if there is one). But you might find it helpful to consider these points when you're writing or evaluating a bug report.



Problem

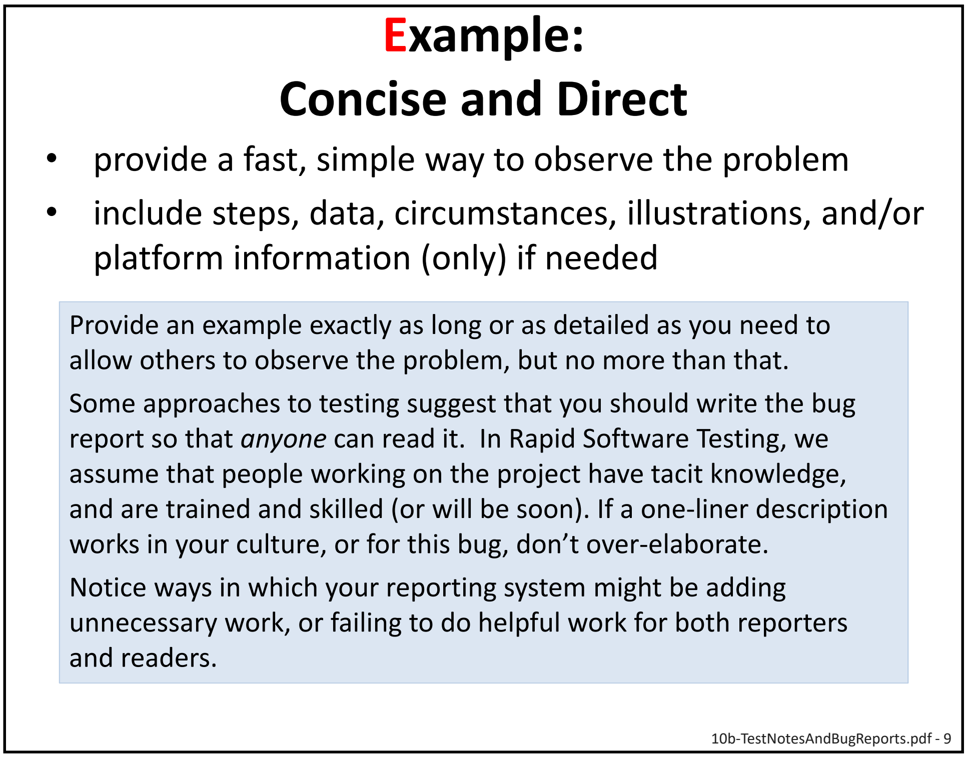
- Provide a brief, impactful summary
- Be specific; don't bury the problem
- What Bad Thing happens? What Good Thing fails to happen?

```
Problem:  Something loose in cockpit.
Solution: Something tightened in cockpit.

Problem: Number 3 engine missing.
Solution: Engine found on right wing after brief search.

Problem: Aircraft handles funny.
Solution: Aircraft warned to straighten up, fly right,
and be serious.
```

For examples, look at newspaper headlines: a thumbnail description that grabs attention, starts conversation, and compels action. The summary should attract appropriate attention to the problem.



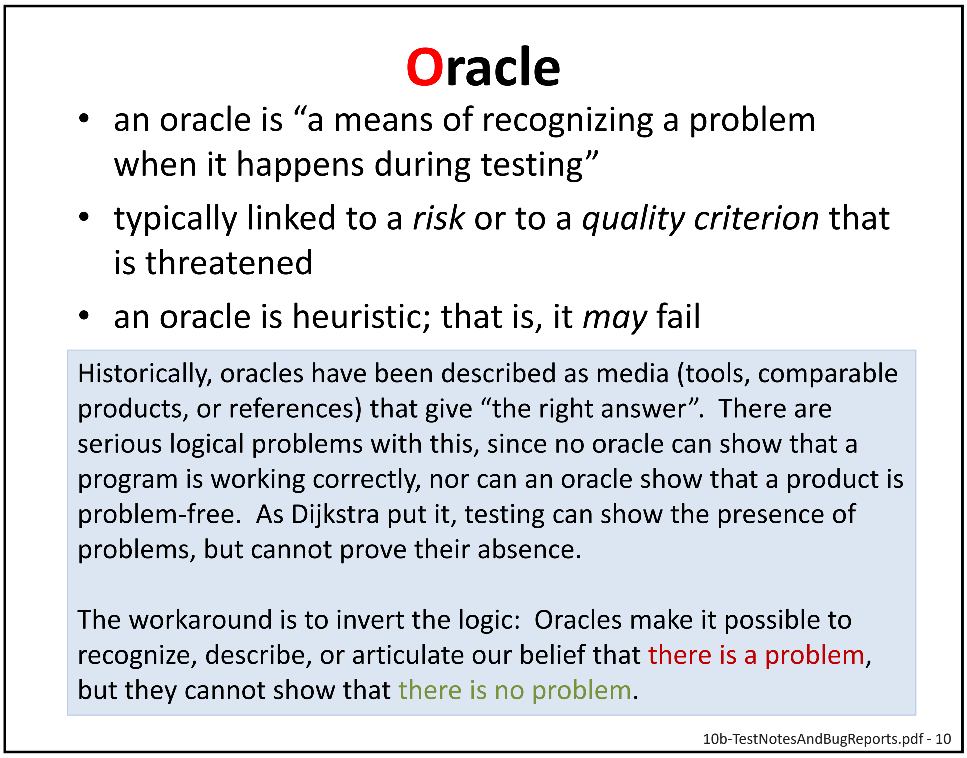
Example: Concise and Direct

- provide a fast, simple way to observe the problem
- include steps, data, circumstances, illustrations, and/or platform information (only) if needed

Provide an example exactly as long or as detailed as you need to allow others to observe the problem, but no more than that.

Some approaches to testing suggest that you should write the bug report so that *anyone* can read it. In Rapid Software Testing, we assume that people working on the project have tacit knowledge, and are trained and skilled (or will be soon). If a one-liner description works in your culture, or for this bug, don’t over-elaborate.

Notice ways in which your reporting system might be adding unnecessary work, or failing to do helpful work for both reporters and readers.

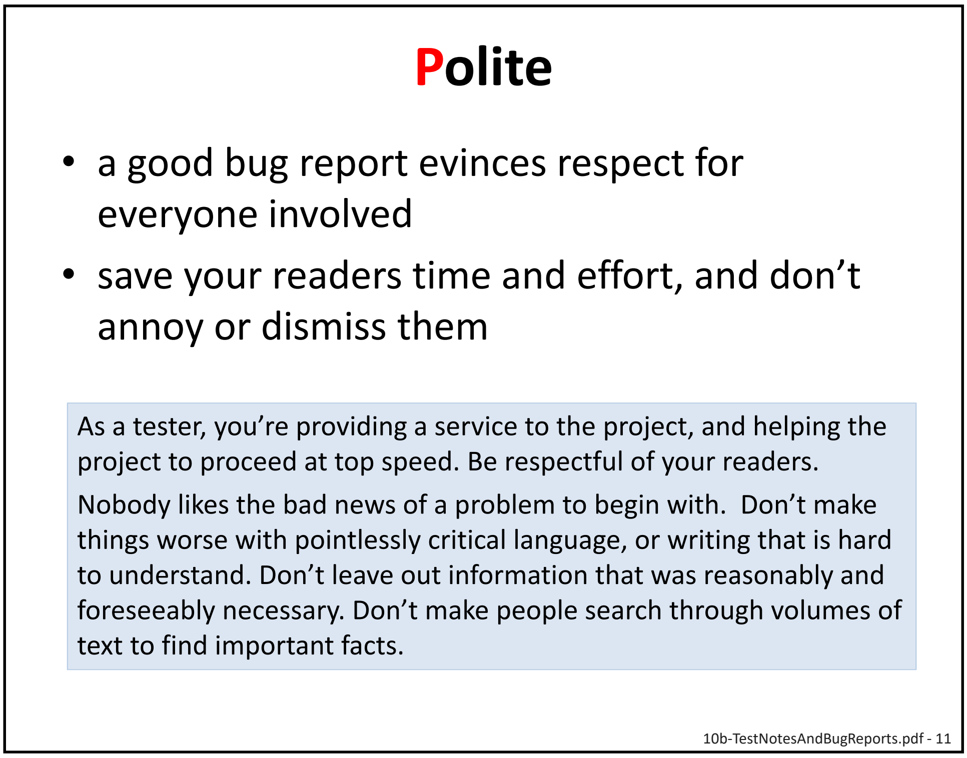


Oracle

- an oracle is “a means of recognizing a problem when it happens during testing”
- typically linked to a *risk* or to a *quality criterion* that is threatened
- an oracle is heuristic; that is, it *may* fail

Historically, oracles have been described as media (tools, comparable products, or references) that give “the right answer”. There are serious logical problems with this, since no oracle can show that a program is working correctly, nor can an oracle show that a product is problem-free. As Dijkstra put it, testing can show the presence of problems, but cannot prove their absence.

The workaround is to invert the logic: Oracles make it possible to recognize, describe, or articulate our belief that there is a problem, but they cannot show that there is no problem.

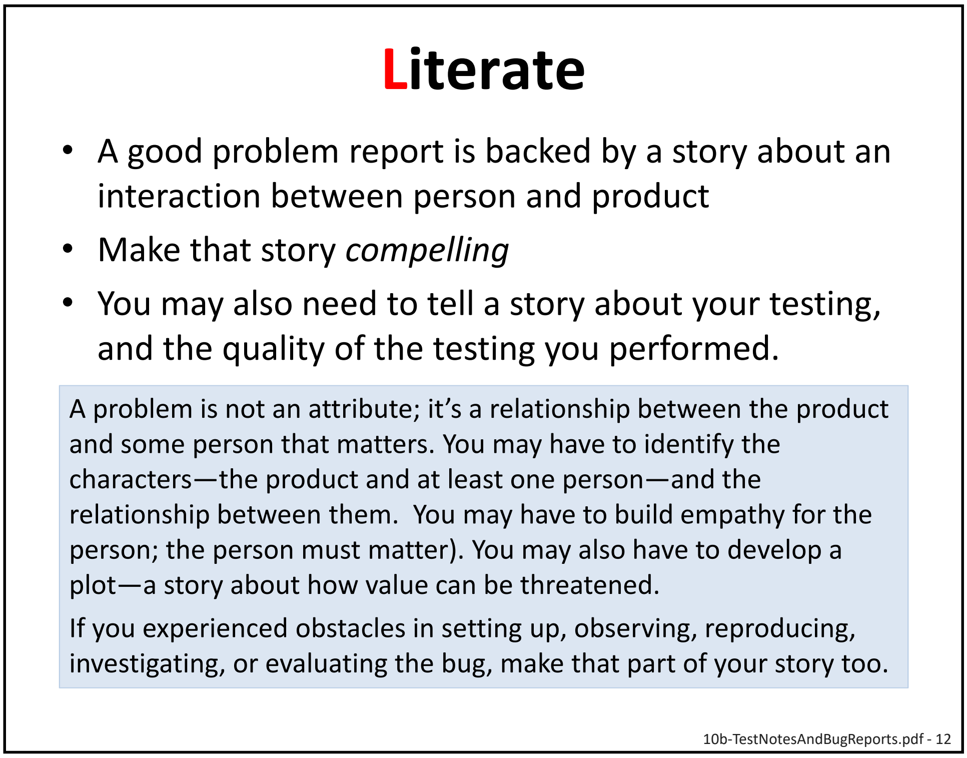


Polite

- a good bug report evinces respect for everyone involved
- save your readers time and effort, and don’t annoy or dismiss them

As a tester, you’re providing a service to the project, and helping the project to proceed at top speed. Be respectful of your readers.

Nobody likes the bad news of a problem to begin with. Don’t make things worse with pointlessly critical language, or writing that is hard to understand. Don’t leave out information that was reasonably and foreseeably necessary. Don’t make people search through volumes of text to find important facts.

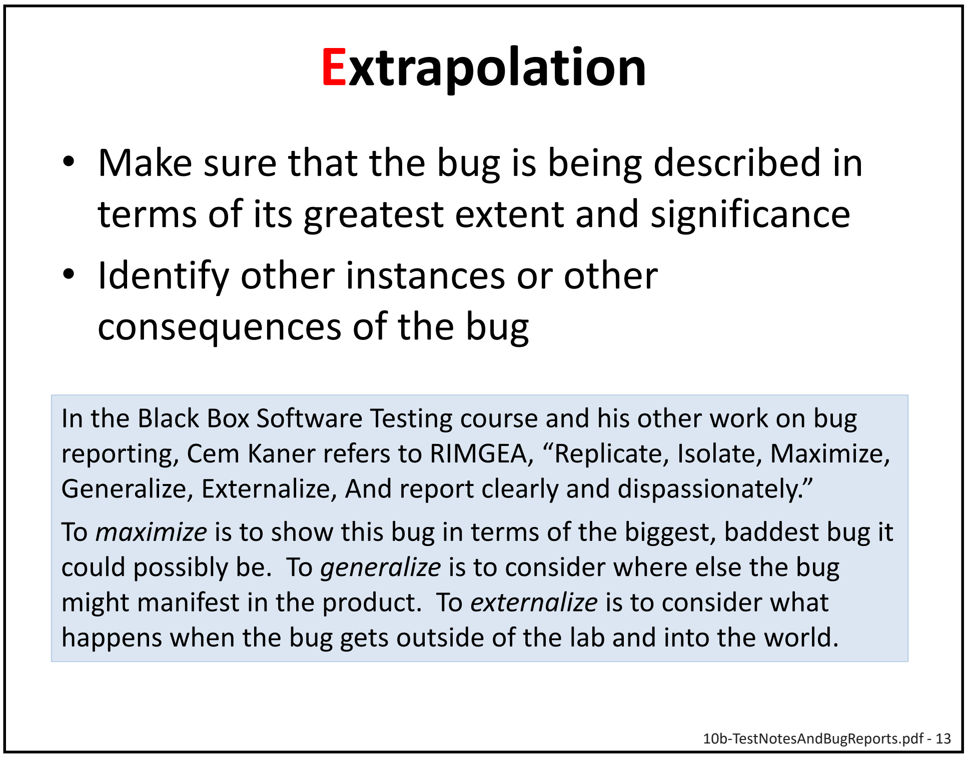


Literate

- A good problem report is backed by a story about an interaction between person and product
- Make that story *compelling*
- You may also need to tell a story about your testing, and the quality of the testing you performed.

A problem is not an attribute; it’s a relationship between the product and some person that matters. You may have to identify the characters—the product and at least one person—and the relationship between them. You may have to build empathy for the person; the person must matter). You may also have to develop a plot—a story about how value can be threatened.

If you experienced obstacles in setting up, observing, reproducing, investigating, or evaluating the bug, make that part of your story too.

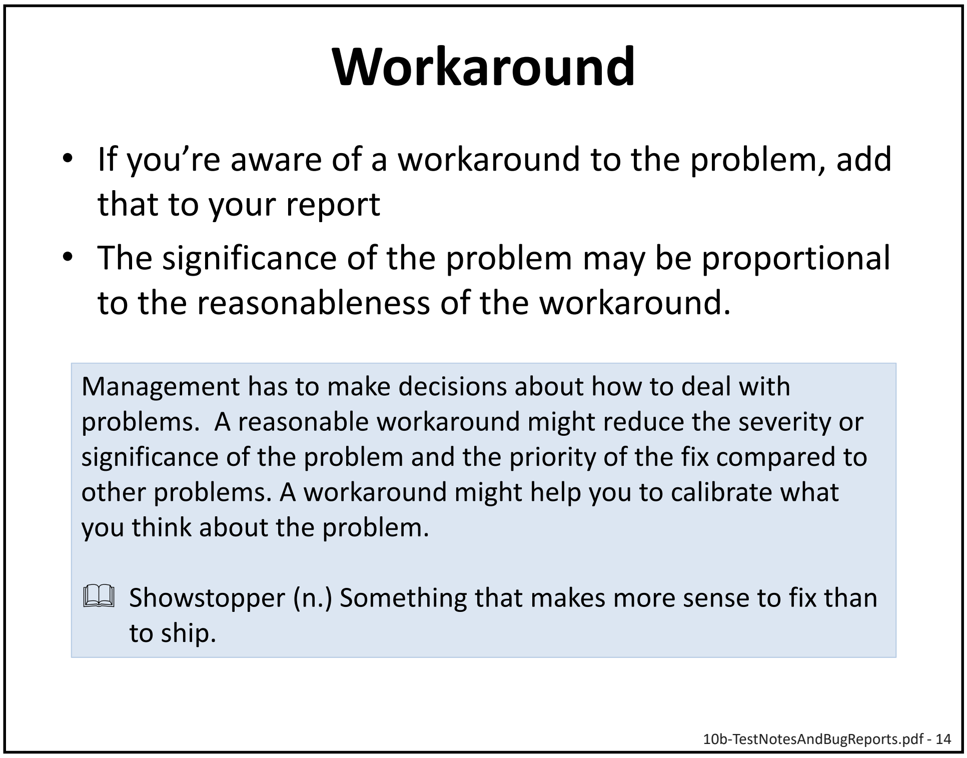


Extrapolation

- Make sure that the bug is being described in terms of its greatest extent and significance
- Identify other instances or other consequences of the bug

In the Black Box Software Testing course and his other work on bug reporting, Cem Kaner refers to RIMGEA, “Replicate, Isolate, Maximize, Generalize, Externalize, And report clearly and dispassionately.”

To *maximize* is to show this bug in terms of the biggest, baddest bug it could possibly be. To *generalize* is to consider where else the bug might manifest in the product. To *externalize* is to consider what happens when the bug gets outside of the lab and into the world.

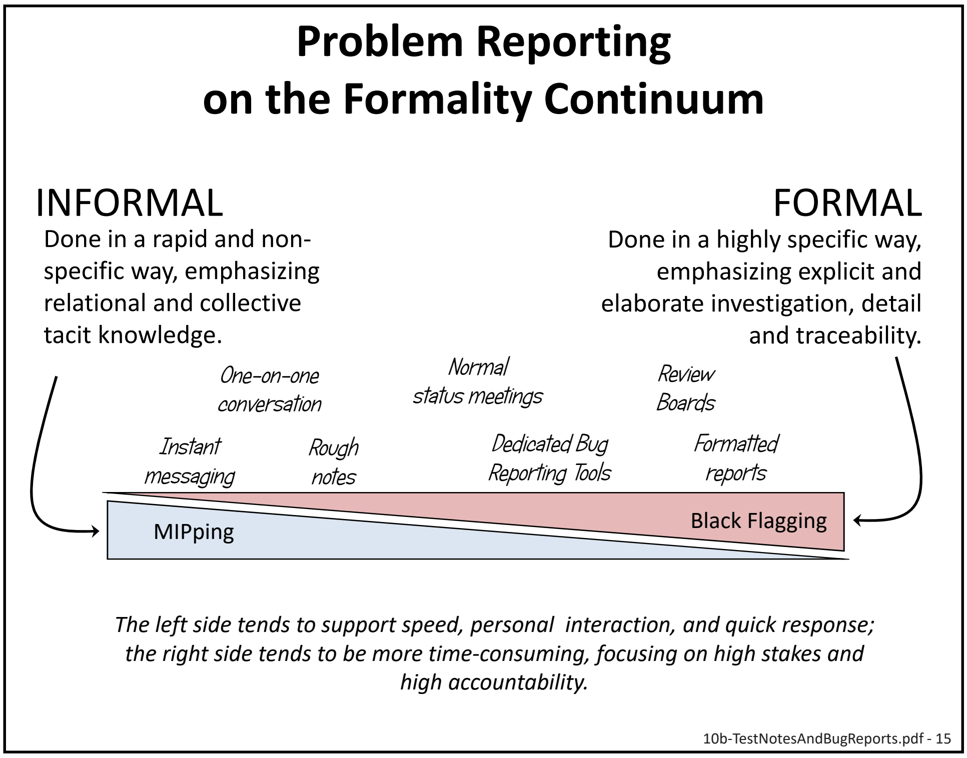


Workaround

- If you’re aware of a workaround to the problem, add that to your report
- The significance of the problem may be proportional to the reasonableness of the workaround.

Management has to make decisions about how to deal with problems. A reasonable workaround might reduce the severity or significance of the problem and the priority of the fix compared to other problems. A workaround might help you to calibrate what you think about the problem.

📖 Showstopper (n.) Something that makes more sense to fix than to ship.



Problem Reporting on the Formality Continuum

INFORMAL

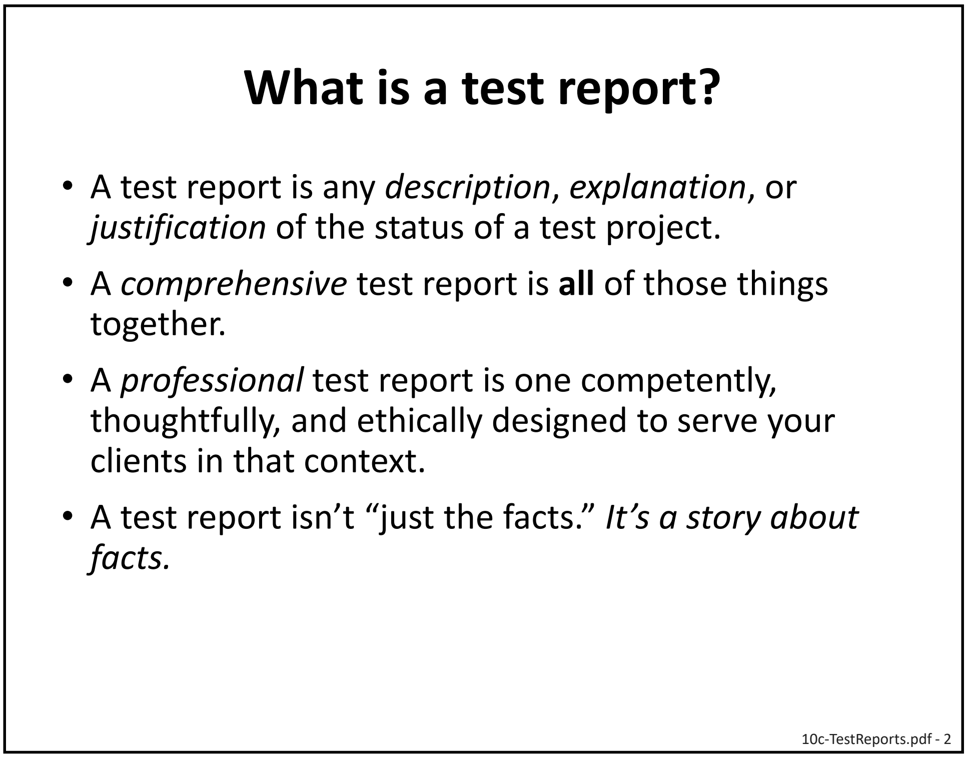
Done in a rapid and non-specific way, emphasizing relational and collective tacit knowledge.

FORMAL

Done in a highly specific way, emphasizing explicit and elaborate investigation, detail and traceability.

The left side tends to support speed, personal interaction, and quick response; the right side tends to be more time-consuming, focusing on high stakes and high accountability.

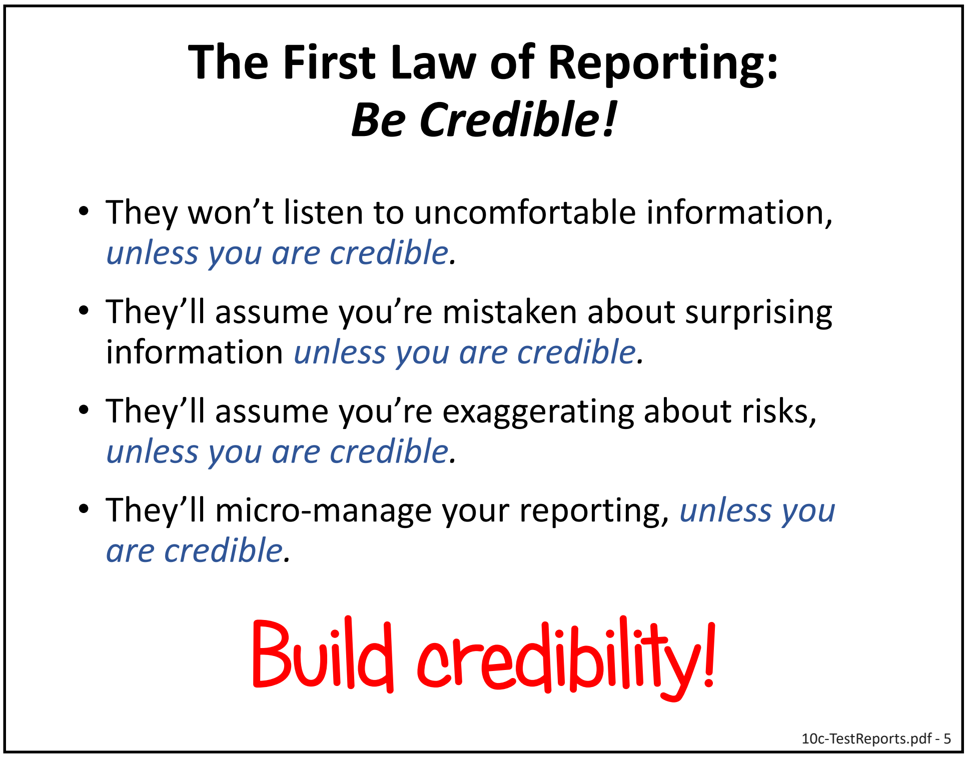
10b-TestNotesAndBugReports.pdf - 15



What is a test report?

- A test report is any *description*, *explanation*, or *justification* of the status of a test project.
- A *comprehensive* test report is **all** of those things together.
- A *professional* test report is one competently, thoughtfully, and ethically designed to serve your clients in that context.
- A test report isn't "just the facts." *It's a story about facts.*

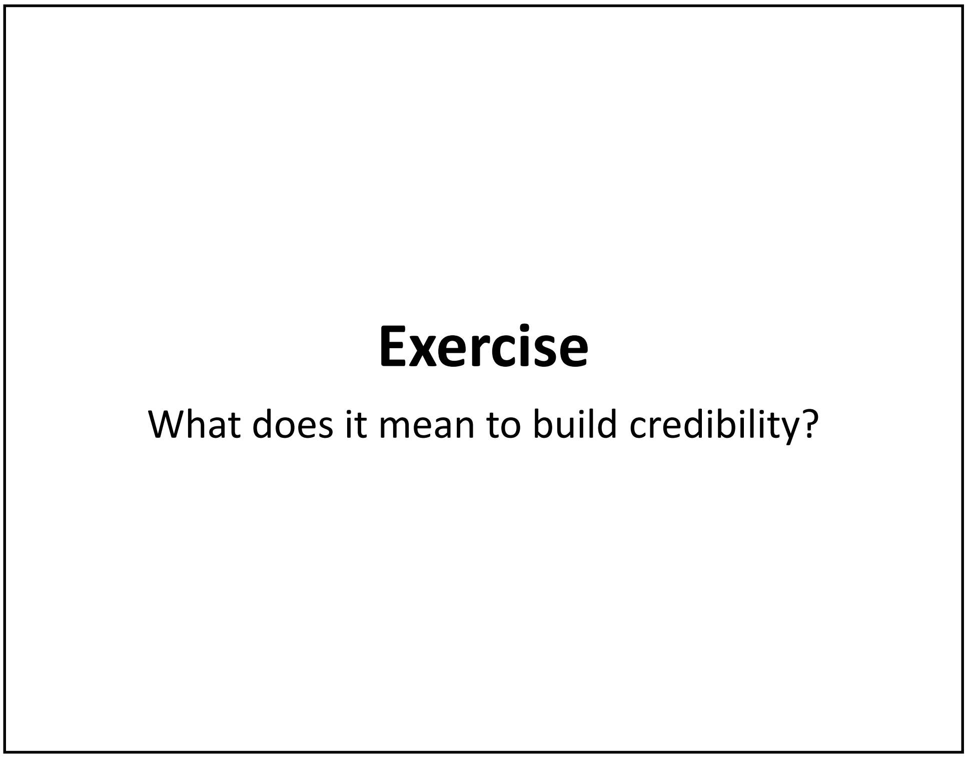
10c-TestReports.pdf - 2



The First Law of Reporting: *Be Credible!*

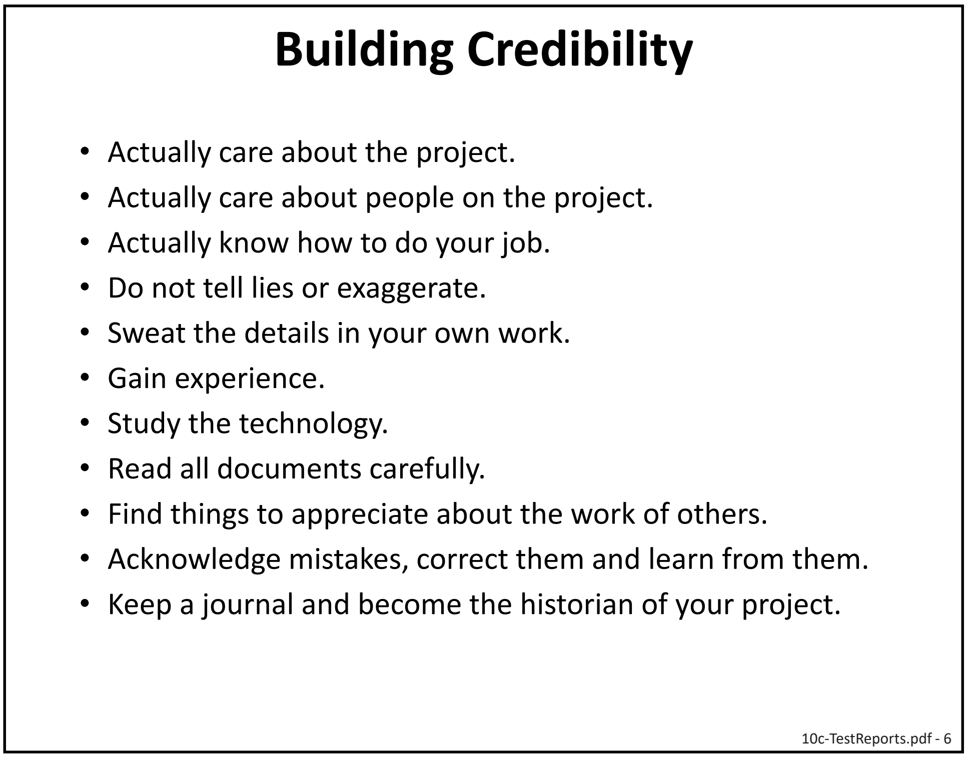
- They won’t listen to uncomfortable information, *unless you are credible.*
- They’ll assume you’re mistaken about surprising information *unless you are credible.*
- They’ll assume you’re exaggerating about risks, *unless you are credible.*
- They’ll micro-manage your reporting, *unless you are credible.*

Build credibility!



Exercise

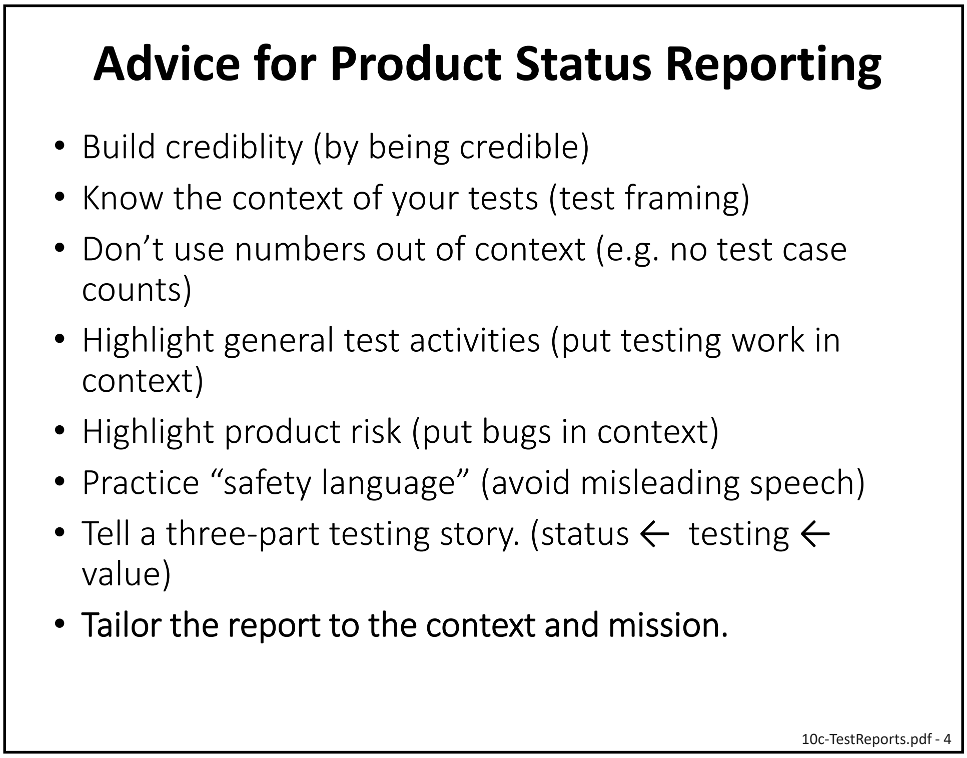
What does it mean to build credibility?



Building Credibility

- Actually care about the project.
- Actually care about people on the project.
- Actually know how to do your job.
- Do not tell lies or exaggerate.
- Sweat the details in your own work.
- Gain experience.
- Study the technology.
- Read all documents carefully.
- Find things to appreciate about the work of others.
- Acknowledge mistakes, correct them and learn from them.
- Keep a journal and become the historian of your project.

10c-TestReports.pdf - 6



Advice for Product Status Reporting

- Build crediblity (by being credible)
- Know the context of your tests (test framing)
- Don’t use numbers out of context (e.g. no test case counts)
- Highlight general test activities (put testing work in context)
- Highlight product risk (put bugs in context)
- Practice “safety language” (avoid misleading speech)
- Tell a three-part testing story. (status ← testing ← value)
- Tailor the report to the context and mission.

10c-TestReports.pdf - 4

Test Case Counting and Unicorns!

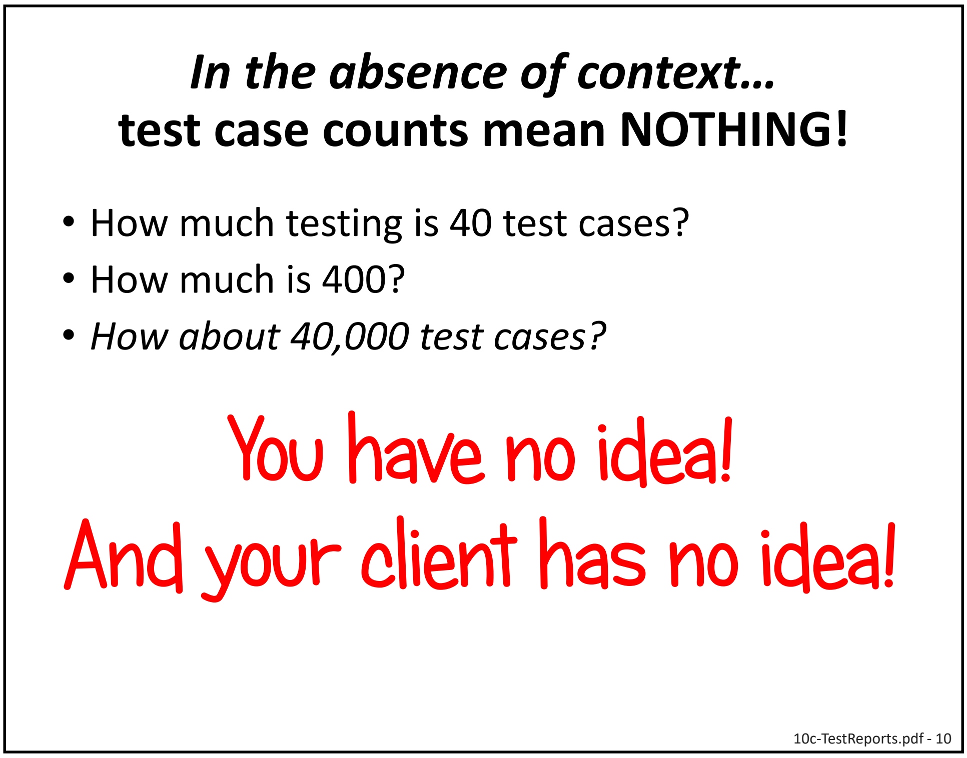
Do you know what a Unicorn is? Okay.

Answer this question:
How many unicorns will fit into your cubicle?

One answer: At Least 15 Quadrillion

- The word “UNICORN” repeated 100,000 times, compressed, and packaged into an archive with 9 other such files, takes up 4096 bytes on my disk.
  - *one million unicorns*
- About 750,000 of those files would fit on a 32gb flash drive.
  - *7.5 trillion unicorns*
- 2,000 32gb flash drives fit tightly in a cubic meter
  - *15 quadrillion unicorns*

That’s fifteen PETACORNS, people!!



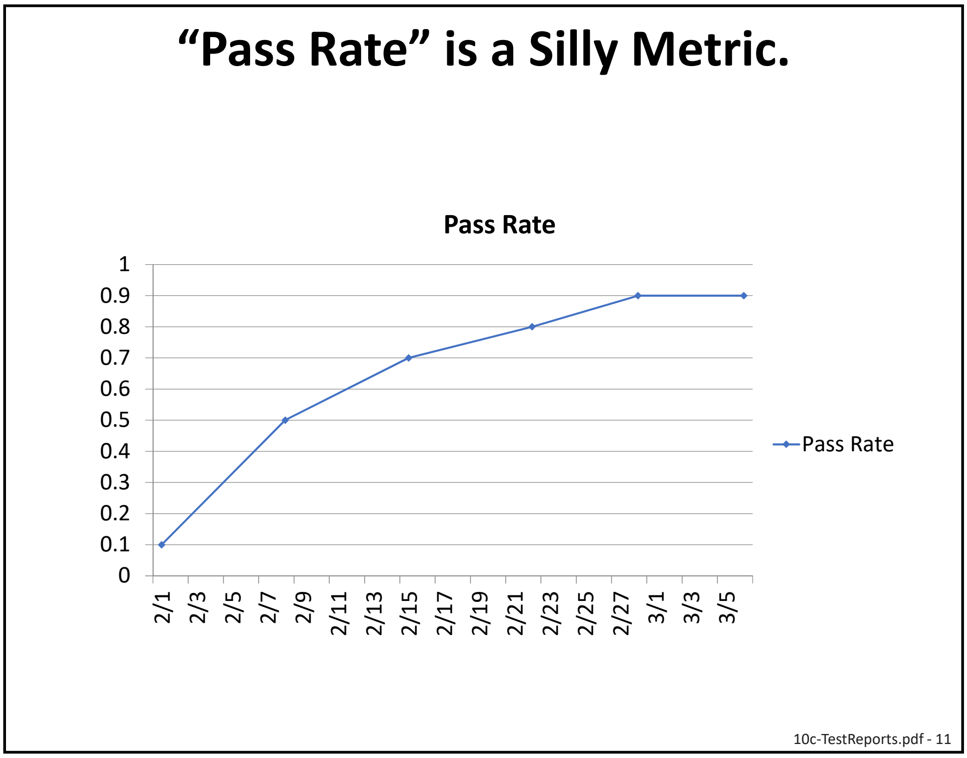
In the absence of context…
test case counts mean NOTHING!

- How much testing is 40 test cases?
- How much is 400?
- *How about 40,000 test cases?*

You have no idea!

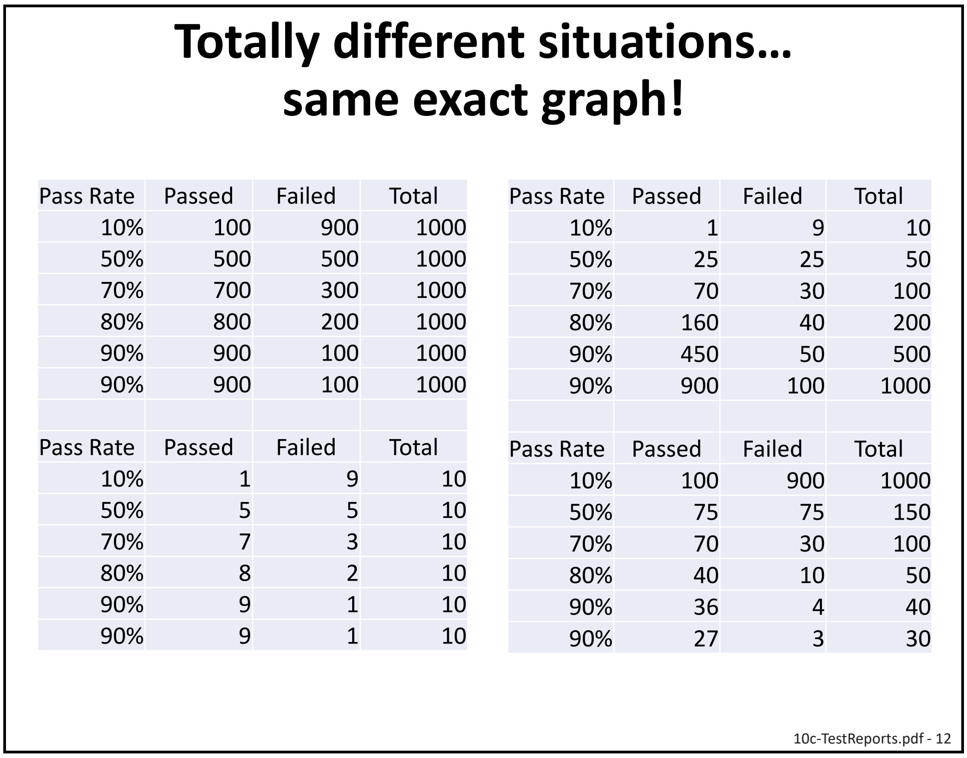
And your client has no idea!

10c-TestReports.pdf - 10



“Pass Rate” is a Silly Metric.

10c-TestReports.pdf - 11



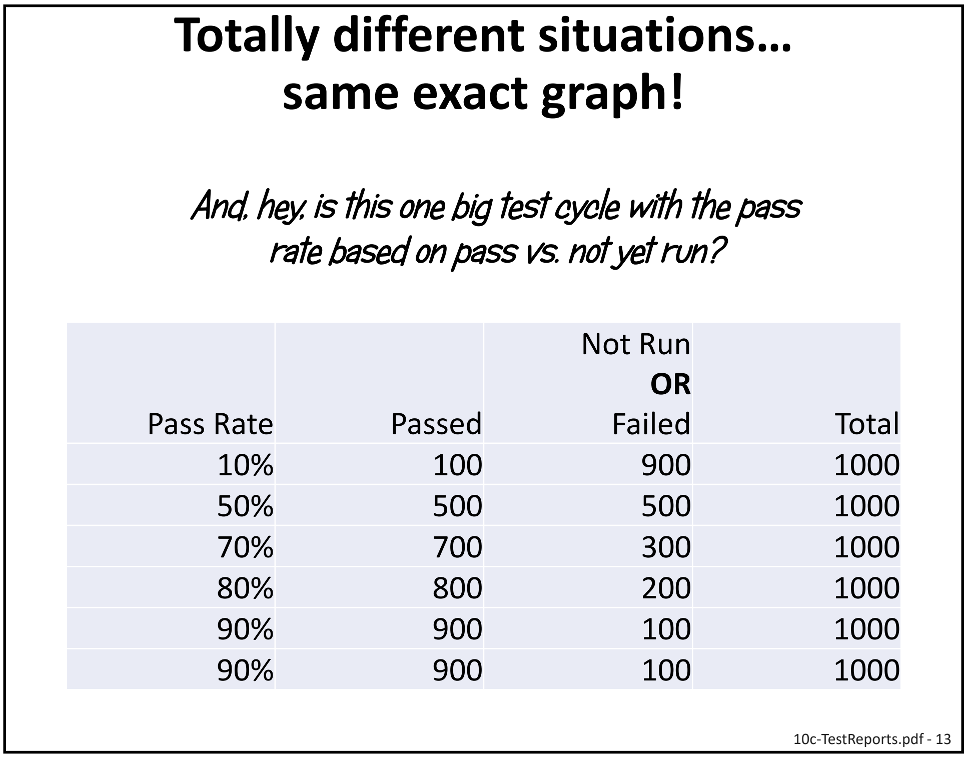
Totally different situations… same exact graph!

Pass Rate	Passed	Failed	Total
10%	100	900	1000
50%	500	500	1000
70%	700	300	1000
80%	800	200	1000
90%	900	100	1000
90%	900	100	1000

Pass Rate	Passed	Failed	Total
10%	1	9	10
50%	25	25	50
70%	70	30	100
80%	160	40	200
90%	450	50	500
90%	900	100	1000

Pass Rate	Passed	Failed	Total
10%	1	9	10
50%	5	5	10
70%	7	3	10
80%	8	2	10
90%	9	1	10
90%	9	1	10

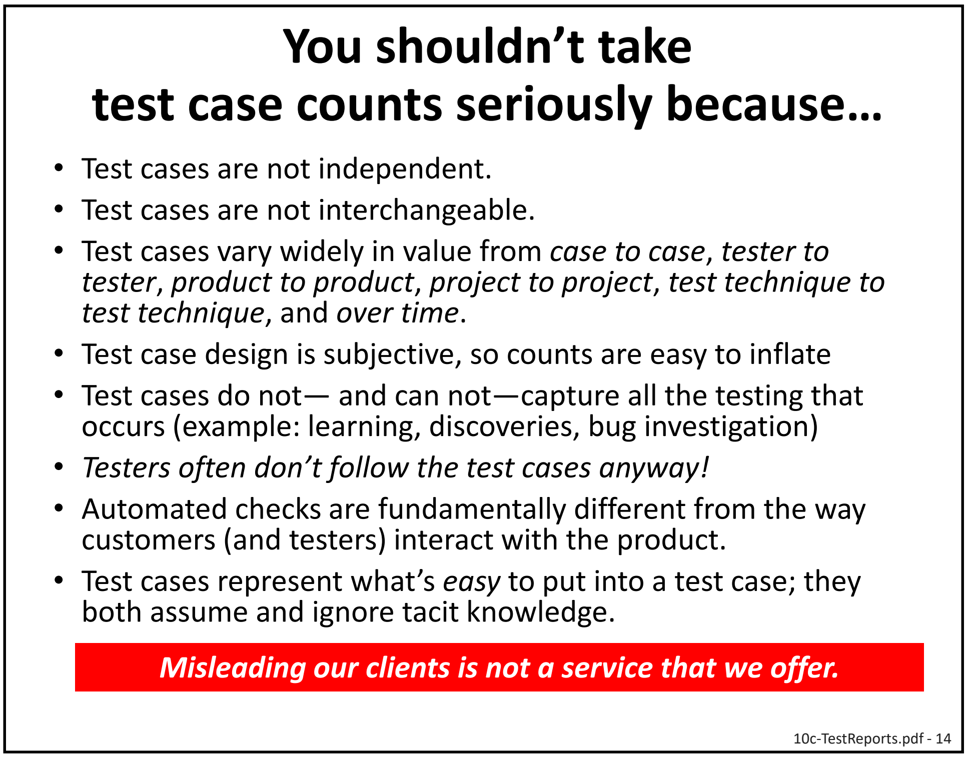
Pass Rate	Passed	Failed	Total
10%	100	900	1000
50%	75	75	150
70%	70	30	100
80%	40	10	50
90%	36	4	40
90%	27	3	30



Totally different situations… same exact graph!

And, hey, is this one big test cycle with the pass rate based on pass vs. not yet run?

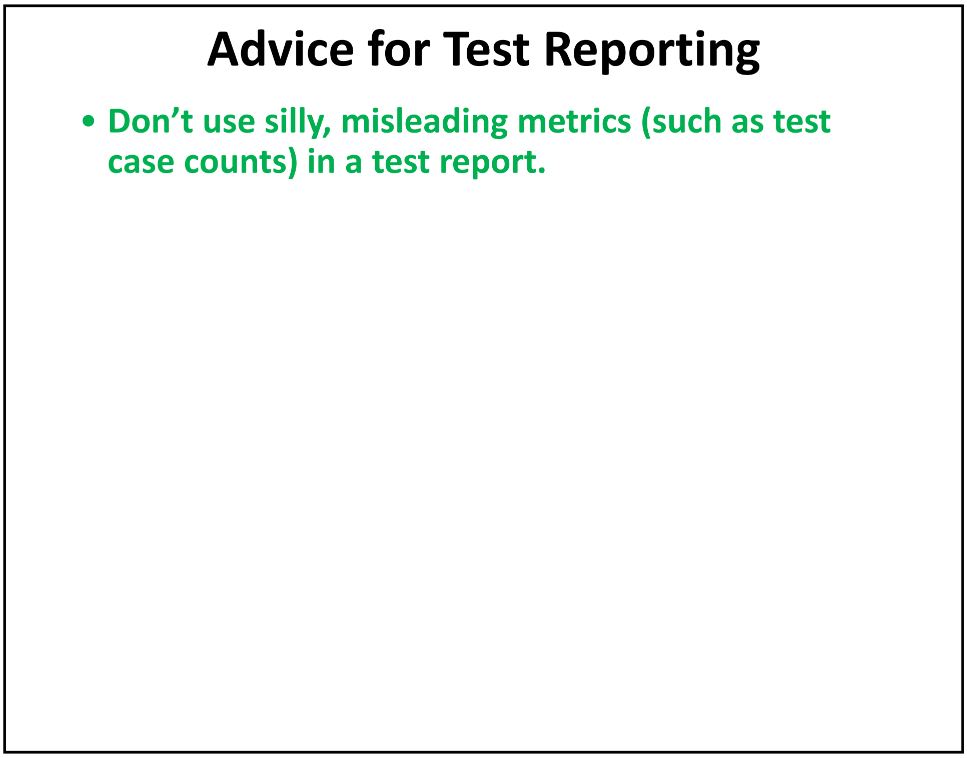
Pass Rate	Passed	Not Run **OR** Failed	Total
10%	100	900	1000
50%	500	500	1000
70%	700	300	1000
80%	800	200	1000
90%	900	100	1000
90%	900	100	1000



You shouldn't take test case counts seriously because…

- Test cases are not independent.
- Test cases are not interchangeable.
- Test cases vary widely in value from *case to case*, *tester to tester*, *product to product*, *project to project*, *test technique to test technique*, and *over time*.
- Test case design is subjective, so counts are easy to inflate
- Test cases do not— and can not—capture all the testing that occurs (example: learning, discoveries, bug investigation)
- *Testers often don't follow the test cases anyway!*
- Automated checks are fundamentally different from the way customers (and testers) interact with the product.
- Test cases represent what's *easy* to put into a test case; they both assume and ignore tacit knowledge.

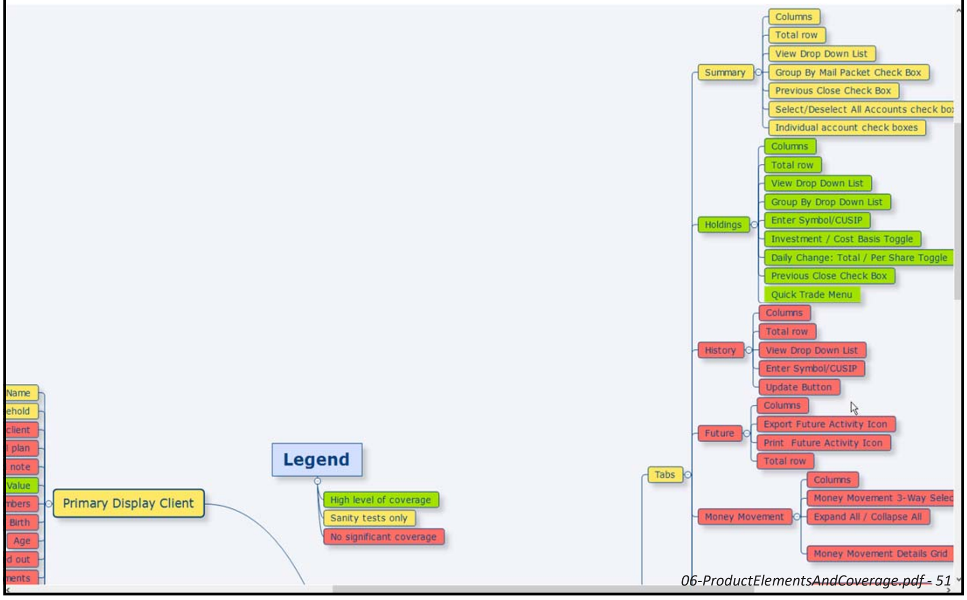
Misleading our clients is not a service that we offer.



Advice for Test Reporting

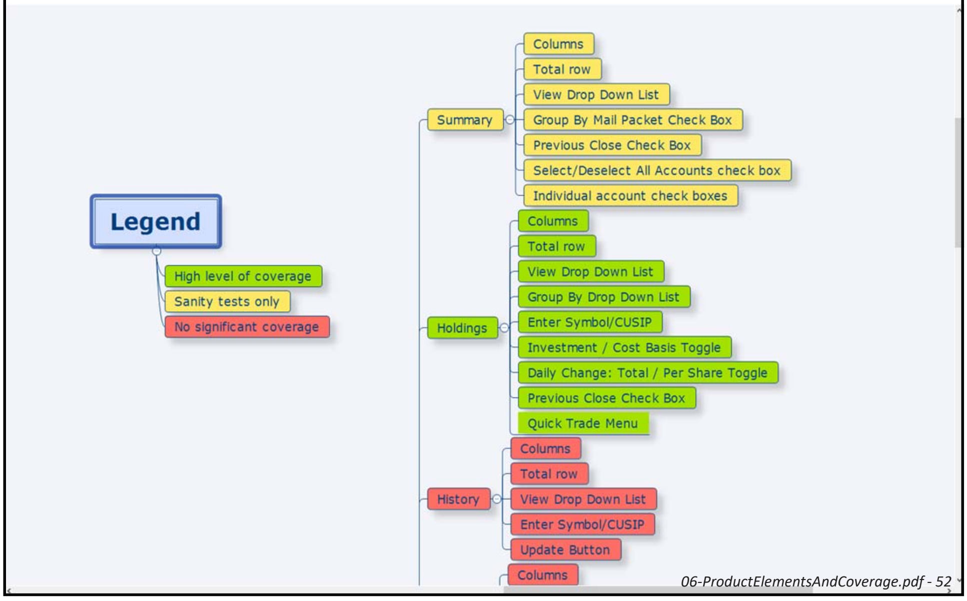
- **Don't use silly, misleading metrics (such as test case counts) in a test report.**

Focus on the Risky Stuff



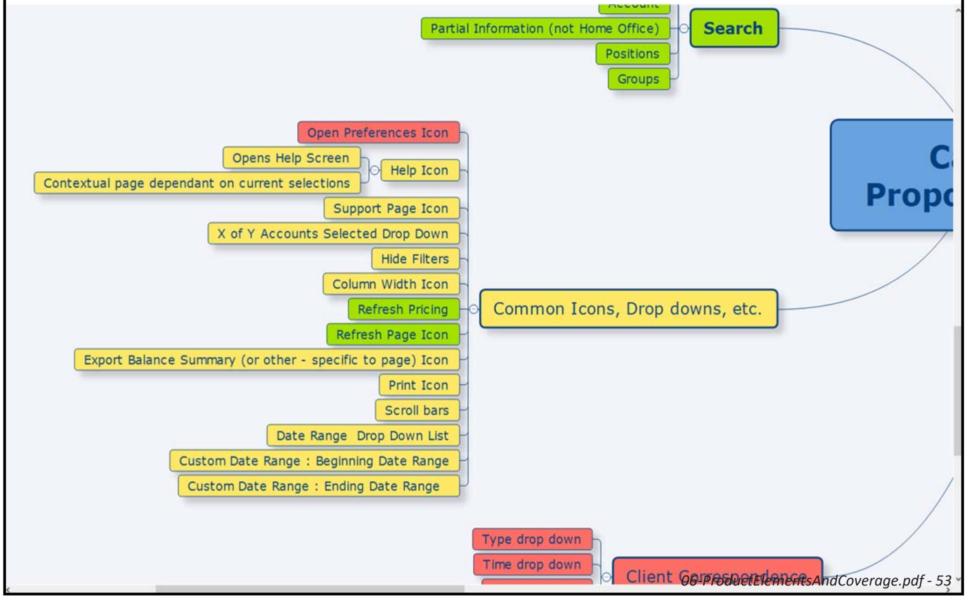
06-ProductElementsAndCoverage.pdf - 51

Feature Area Coverage



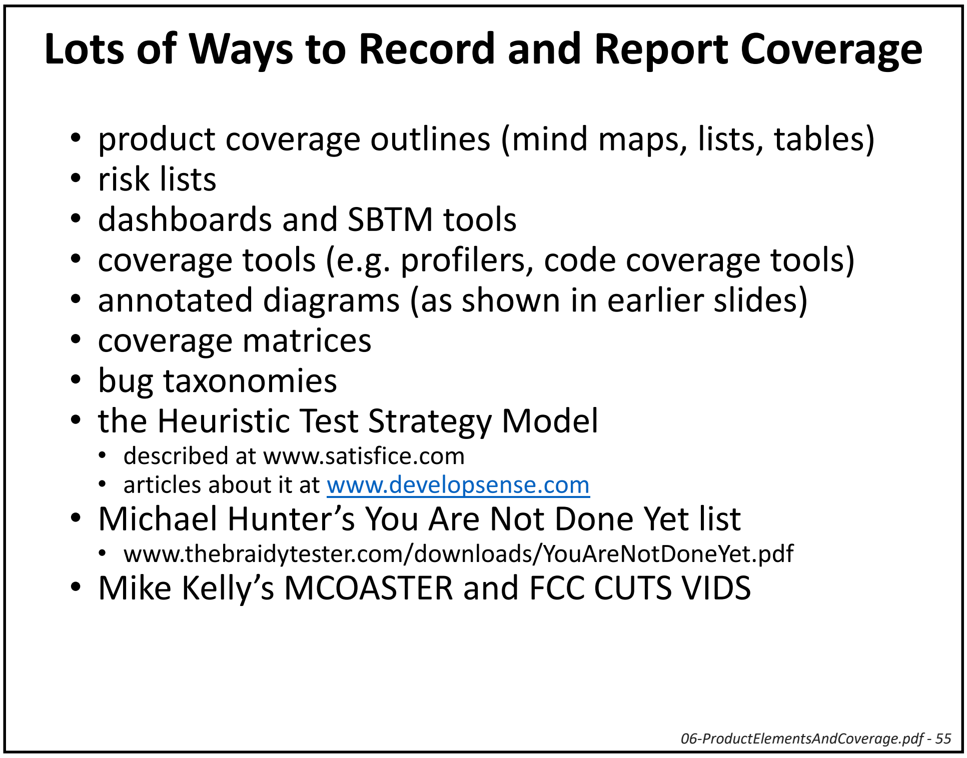
06-ProductElementsAndCoverage.pdf - 52

Interface Coverage


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

Degrees of Coverage

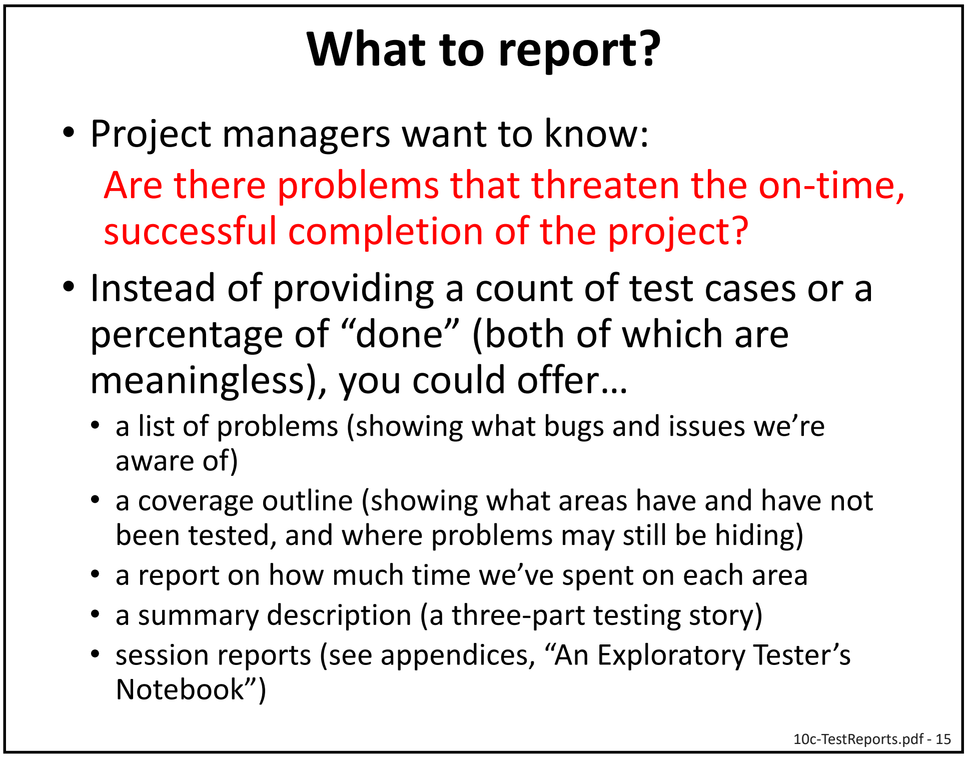
Level 0	**We don't really know anything about this area.** We're aware that this area exists, but it's a black box to us, so far.
Level 1	**We're just getting to know this area.** We've done basic reconnaissance; surveyed it; we've done smoke and sanity testing. We may have some artifacts that represent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



Lots of Ways to Record and Report Coverage

- product coverage outlines (mind maps, lists, tables)
- risk lists
- dashboards and SBTM tools
- coverage tools (e.g. profilers, code coverage tools)
- annotated diagrams (as shown in earlier slides)
- coverage matrices
- bug taxonomies
- the Heuristic Test Strategy Model
  - described at www.satisfice.com
  - articles about it at www.developsense.com
- Michael Hunter's You Are Not Done Yet list
  - www.thebraidytester.com/downloads/YouAreNotDoneYet.pdf
- Mike Kelly's MCOASTER and FCC CUTS VIDS

06-ProductElementsAndCoverage.pdf - 55

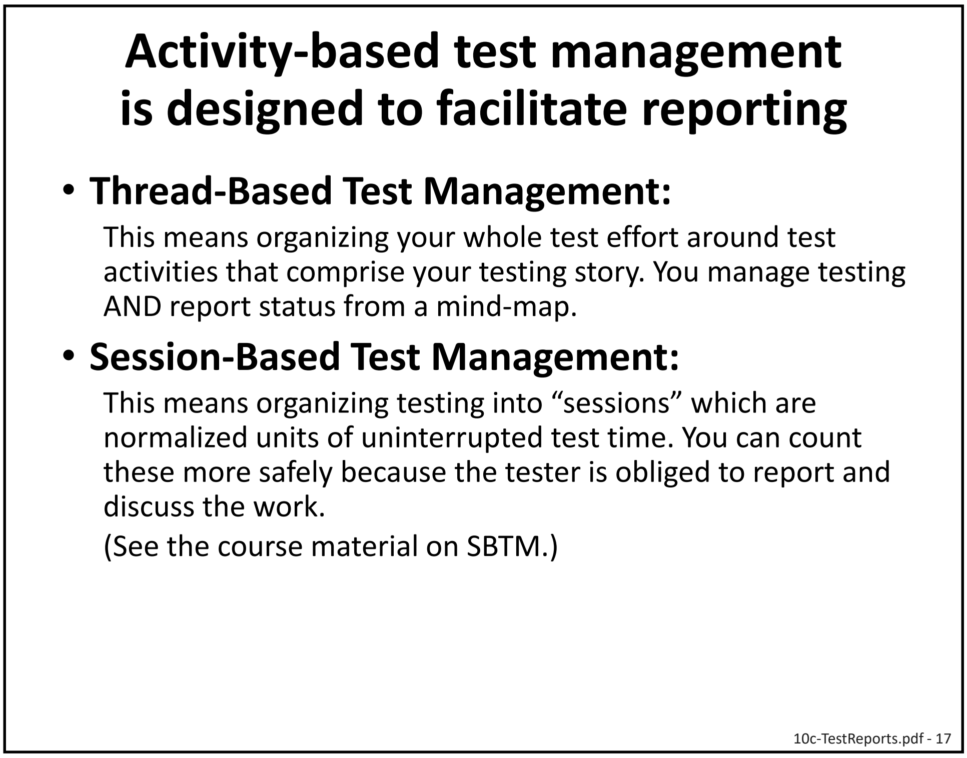


What to report?

- Project managers want to know:

  Are there problems that threaten the on-time, successful completion of the project?
- Instead of providing a count of test cases or a percentage of "done" (both of which are meaningless), you could offer…
  - a list of problems (showing what bugs and issues we're aware of)
  - a coverage outline (showing what areas have and have not been tested, and where problems may still be hiding)
  - a report on how much time we've spent on each area
  - a summary description (a three-part testing story)
  - session reports (see appendices, "An Exploratory Tester's Notebook")

10c-TestReports.pdf - 15



Activity-based test management is designed to facilitate reporting

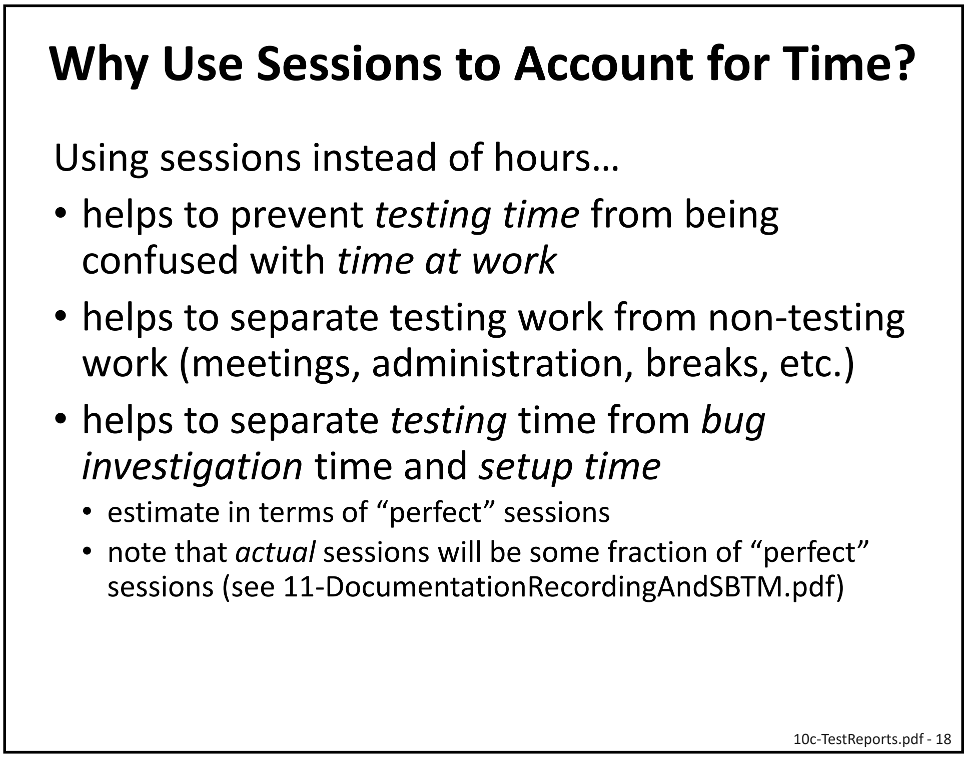
- **Thread-Based Test Management:**

  This means organizing your whole test effort around test activities that comprise your testing story. You manage testing AND report status from a mind-map.

- **Session-Based Test Management:**

  This means organizing testing into "sessions" which are normalized units of uninterrupted test time. You can count these more safely because the tester is obliged to report and discuss the work.

  (See the course material on SBTM.)



Why Use Sessions to Account for Time?

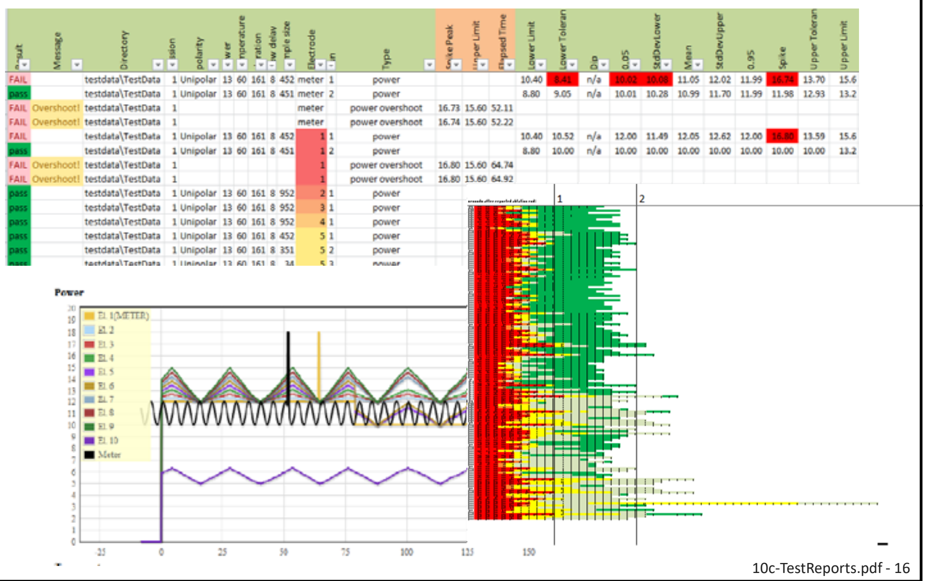
Using sessions instead of hours…

- helps to prevent *testing time* from being confused with *time at work*
- helps to separate testing work from non-testing work (meetings, administration, breaks, etc.)
- helps to separate *testing* time from *bug investigation* time and *setup time*
  - estimate in terms of "perfect" sessions
  - note that *actual* sessions will be some fraction of "perfect" sessions (see 11-DocumentationRecordingAndSBTM.pdf)

Advice for Test Reporting

- Don’t use silly, misleading metrics (such as test case counts) in a test report.
- **Tailor your reporting to the situation at hand.**

Illustrating the product story by visualizing the data



Result	Message	Directory	sion	polarity	wer	nperature	ration	w delay	mple size	Electrode	n	Type	Spike Peak	Upper Limit	Elapsed Time	Lower Limit	Lower Toleran	Dip	0.05	StdDevLower	Mean	StdDevUpper	0.95	Spike	Upper Toleran	Upper Limit
FAIL		testdata\TestData	1	Unipolar	13	60	161	8	452	meter	1	power				10.40	8.41	n/a	10.02	10.08	11.05	12.02	11.99	16.74	13.70	15.6
pass		testdata\TestData	1	Unipolar	13	60	161	8	451	meter	2	power				8.80	9.05	n/a	10.01	10.28	10.99	11.70	11.99	11.98	12.93	13.2
FAIL	Overshoot!	testdata\TestData	1							meter		power overshoot	16.73	15.60	52.11											
FAIL	Overshoot!	testdata\TestData	1							meter		power overshoot	16.74	15.60	52.22											
FAIL		testdata\TestData	1	Unipolar	13	60	161	8	452	1	1	power				10.40	10.52	n/a	12.00	11.49	12.05	12.62	12.00	16.80	13.59	15.6
pass		testdata\TestData	1	Unipolar	13	60	161	8	451	1	2	power				8.80	10.00	n/a	10.00	10.00	10.00	10.00	10.00	10.00	10.00	13.2
FAIL	Overshoot!	testdata\TestData	1							1		power overshoot	16.80	15.60	64.74											
FAIL	Overshoot!	testdata\TestData	1							1		power overshoot	16.80	15.60	64.92											
pass		testdata\TestData	1	Unipolar	13	60	161	8	952	2	1	power														
pass		testdata\TestData	1	Unipolar	13	60	161	8	952	3	1	power														
pass		testdata\TestData	1	Unipolar	13	60	161	8	952	4	1	power														
pass		testdata\TestData	1	Unipolar	13	60	161	8	452	5	1	power														
pass		testdata\TestData	1	Unipolar	13	60	161	8	351	5	2	power														
pass		testdata\TestData	1	Unipolar	13	60	161	8	34	5	3	power														

Safety Language

- "Safety language" in software testing, means to qualify or otherwise draft statements of fact so as to avoid false confidence.
- Examples:

I think...

So far...

It appears...

I infer...

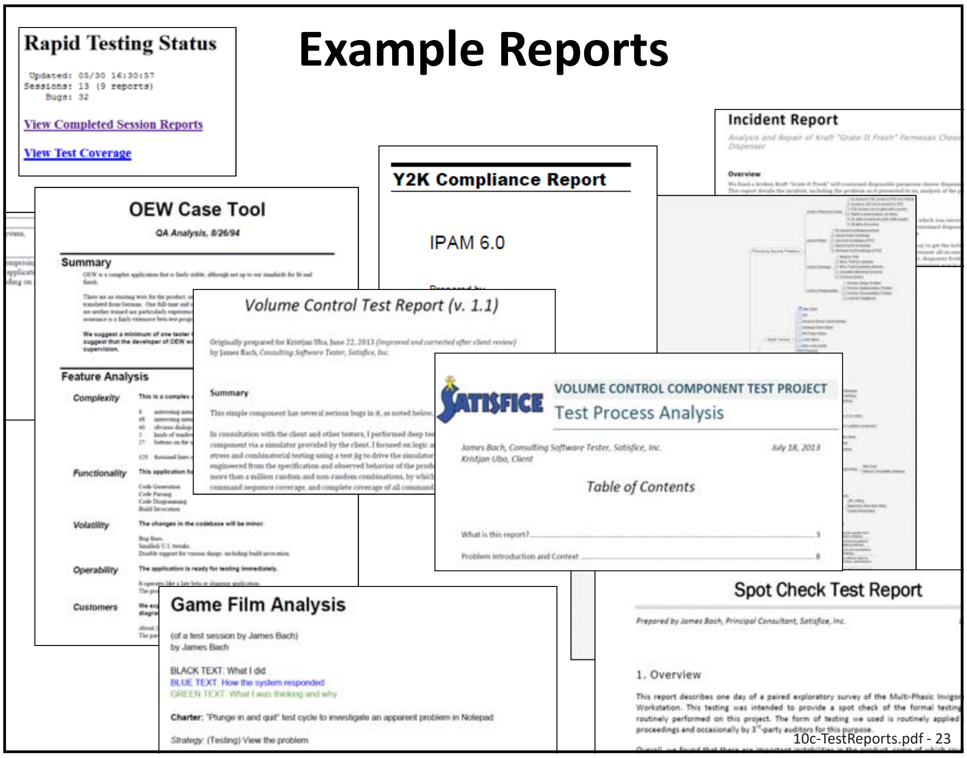
apparently...

I assumed...

It seems...

~~*The feature worked*~~ ⇨ *I have not yet seen any failures in the feature...*

Example Reports



Rapid Testing Status

```
 Updated: 05/30 16:30:57
Sessions: 13 (9 reports)
    Bugs: 32
```

View Completed Session Reports

View Test Coverage

OEW Case Tool

QA Analysis, 8/26/94

Summary

Feature Analysis

Complexity

Functionality

Volatility — The changes in the codebase will be minor.

Operability — The application is ready for testing immediately.

Customers

Y2K Compliance Report

IPAM 6.0

Volume Control Test Report (v. 1.1)

Originally prepared for Kristjan Uba, June 22, 2013 *(improved and corrected after client review)* by James Bach, *Consulting Software Tester, Satisfice, Inc.*

Summary

This simple component has several serious bugs in it, as noted below,

In consultation with the client and other testers, I performed deep te… component via a simulator provided by the client. I focused on logic a… stress and combinatorial testing using a test jig to drive the simulator… engineered from the specification and observed behavior of the produ… more than a million random and non-random combinations, by which… command sequence coverage, and complete coverage of all command…

Incident Report

Analysis and Repair of Kraft "Grate-It Fresh" Parmesan Chees… Dispenser

Overview

SATISFICE

VOLUME CONTROL COMPONENT TEST PROJECT

Test Process Analysis

James Bach, Consulting Software Tester, Satisfice, Inc. *July 18, 2013*

Kristjan Uba, Client

Table of Contents

What is this report? 3

Problem Introduction and Context 8

Game Film Analysis

(of a test session by James Bach)
by James Bach

BLACK TEXT: What I did
BLUE TEXT: How the system responded
GREEN TEXT: What I was thinking and why

Charter: "Plunge in and quit" test cycle to investigate an apparent problem in Notepad

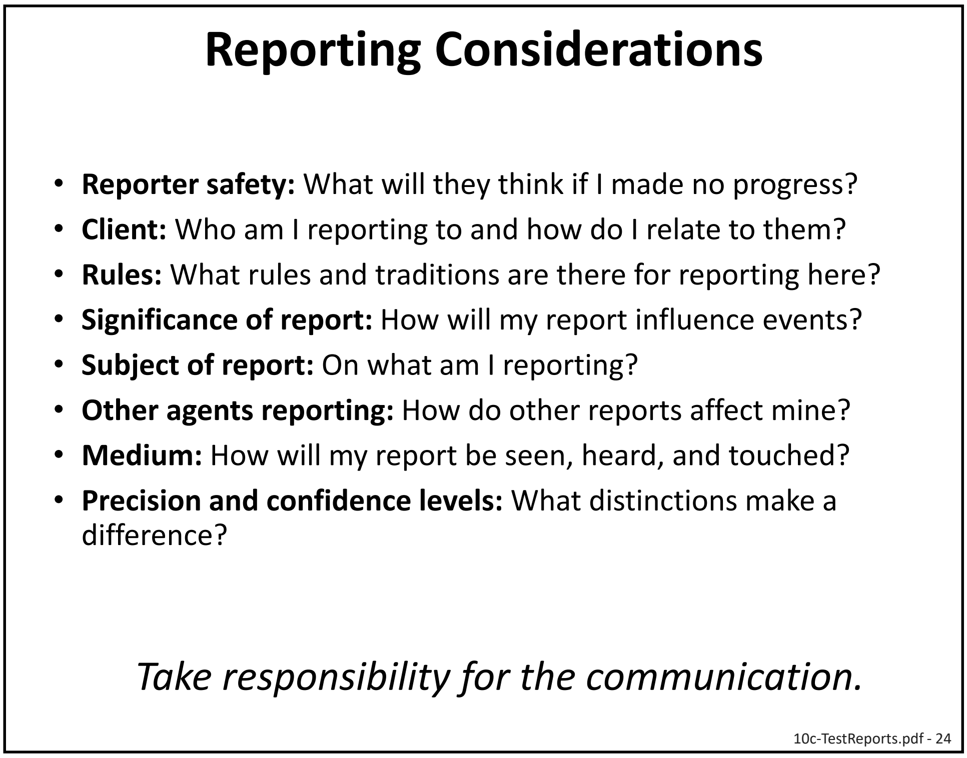
Strategy: (Testing) View the problem

Spot Check Test Report

Prepared by James Bach, Principal Consultant, Satisfice, Inc.

1. Overview

This report describes one day of a paired exploratory survey of the Multi-Phasic Invigor… Workstation. This testing was intended to provide a spot check of the formal testing… routinely performed on this project. The form of testing we used is routinely applied… proceedings and occasionally by 3rd-party auditors for this purpose.

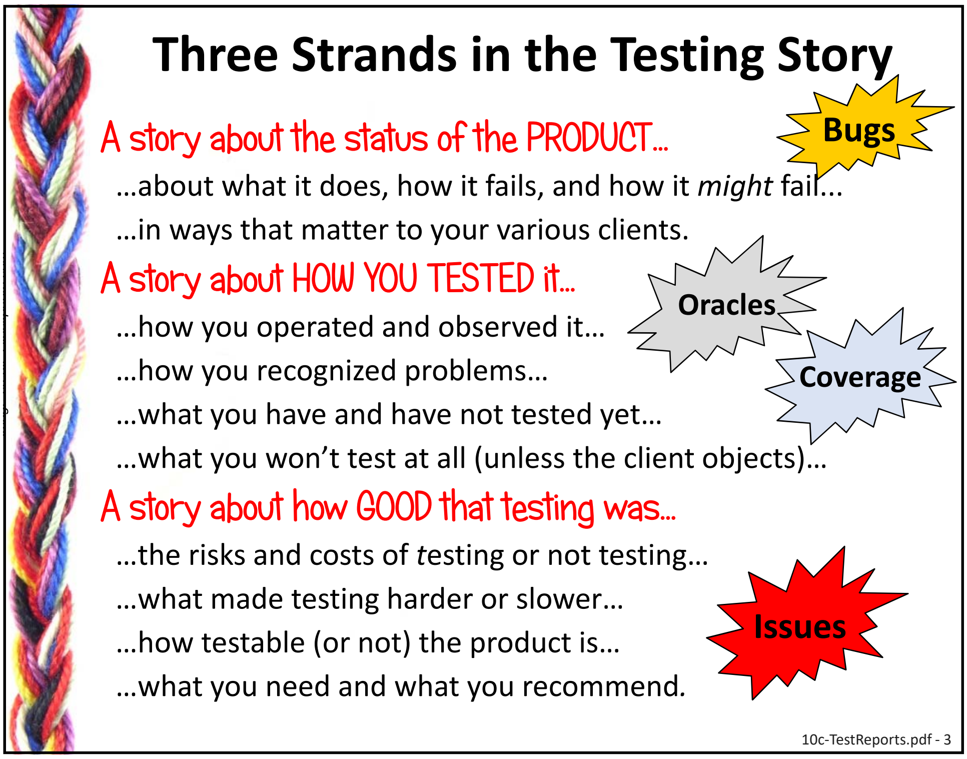


Reporting Considerations

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

Take responsibility for the communication.

10c-TestReports.pdf - 24



Three Strands in the Testing Story

A story about the status of the PRODUCT…

Bugs

…about what it does, how it fails, and how it *might* fail…

…in ways that matter to your various clients.

A story about HOW YOU TESTED it…

Oracles

Coverage

…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)…

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

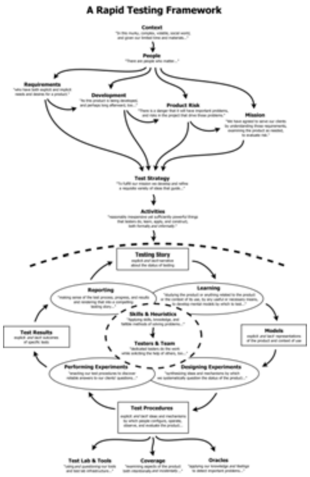
10c-TestReports.pdf - 3

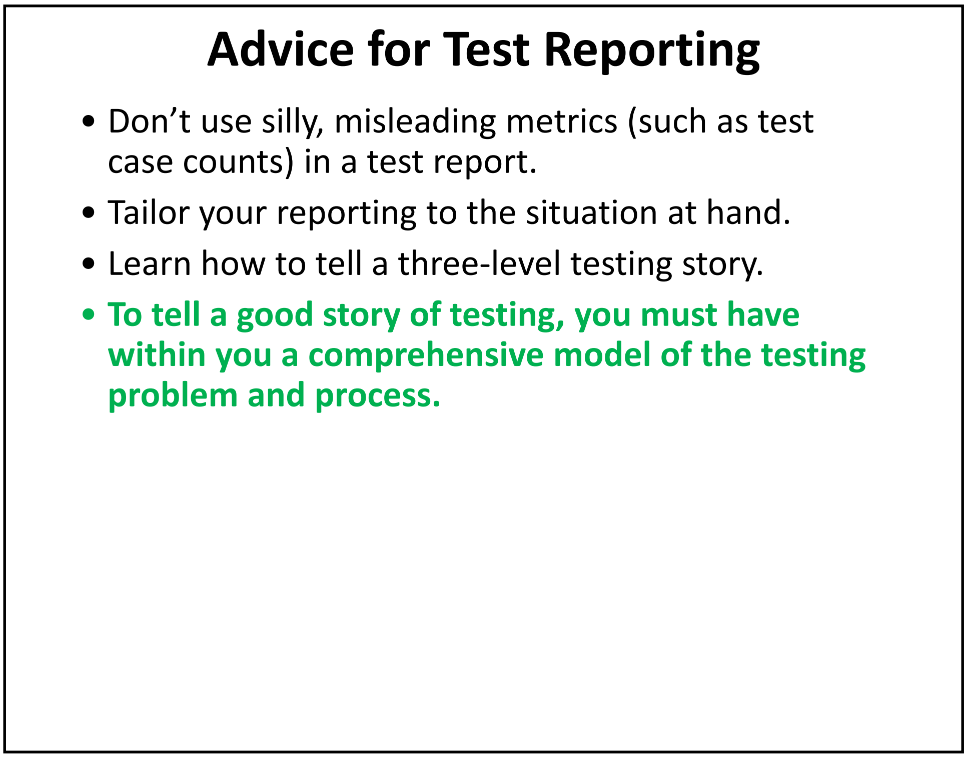
Advice for Test Reporting

- Don’t use silly, misleading metrics (such as test case counts) in a test report.
- Tailor your reporting to the situation at hand.
- **Learn how to tell a three-level testing story.**

A Narrative Model of Testing

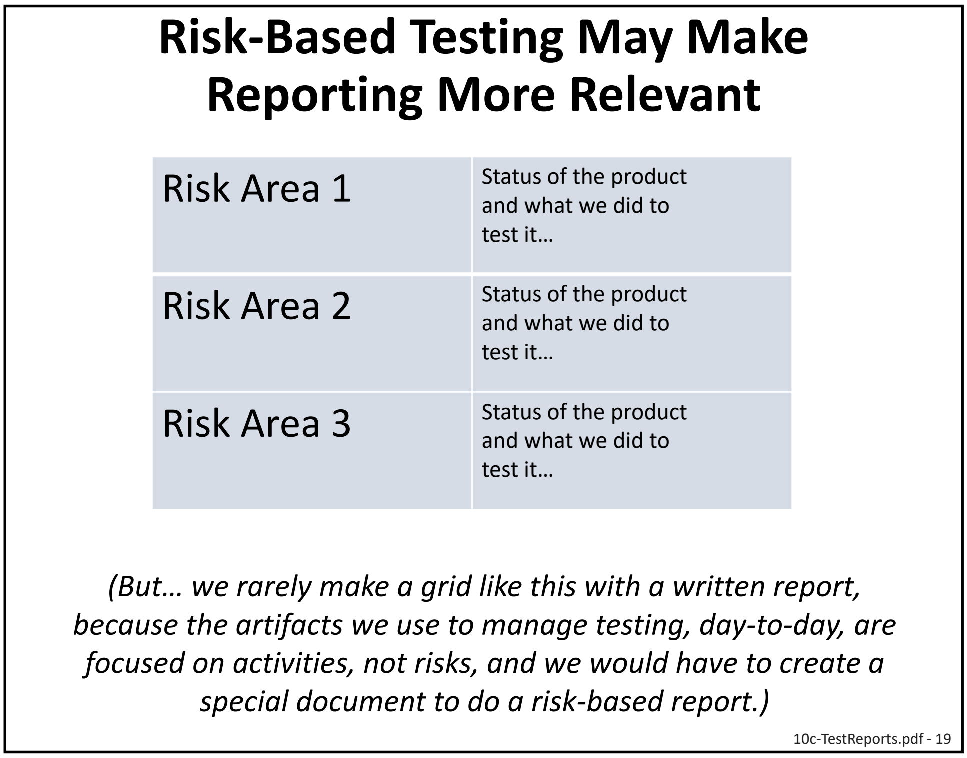
- This is a map of the Rapid Testing methodology that I teach.
- It is organized in the structure of a story, because story construction is at the heart of what it means to test.





Advice for Test Reporting

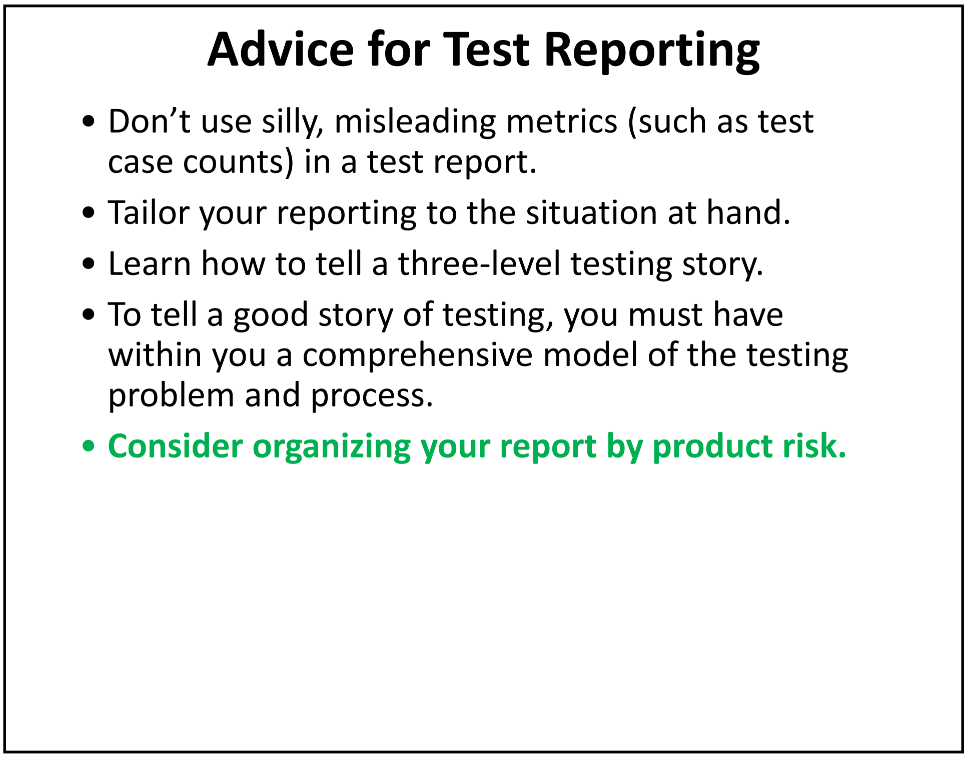
- Don’t use silly, misleading metrics (such as test case counts) in a test report.
- Tailor your reporting to the situation at hand.
- Learn how to tell a three-level testing story.
- **To tell a good story of testing, you must have within you a comprehensive model of the testing problem and process.**



Risk-Based Testing May Make Reporting More Relevant

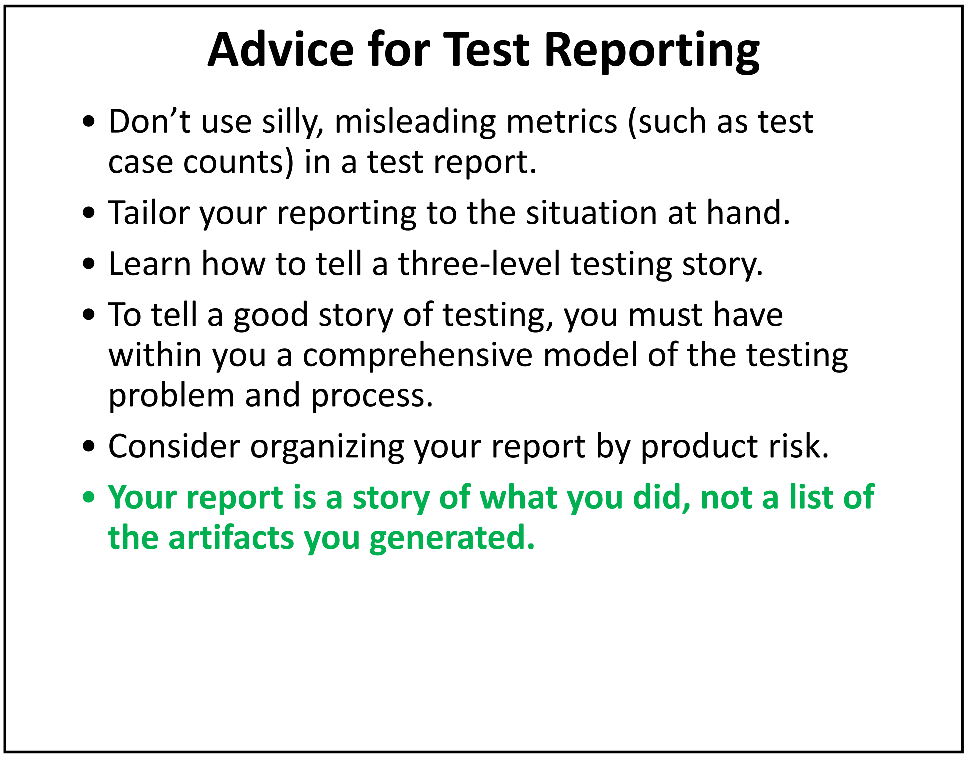
Risk Area 1	Status of the product and what we did to test it…
Risk Area 2	Status of the product and what we did to test it…
Risk Area 3	Status of the product and what we did to test it…

(But… we rarely make a grid like this with a written report, because the artifacts we use to manage testing, day-to-day, are focused on activities, not risks, and we would have to create a special document to do a risk-based report.)



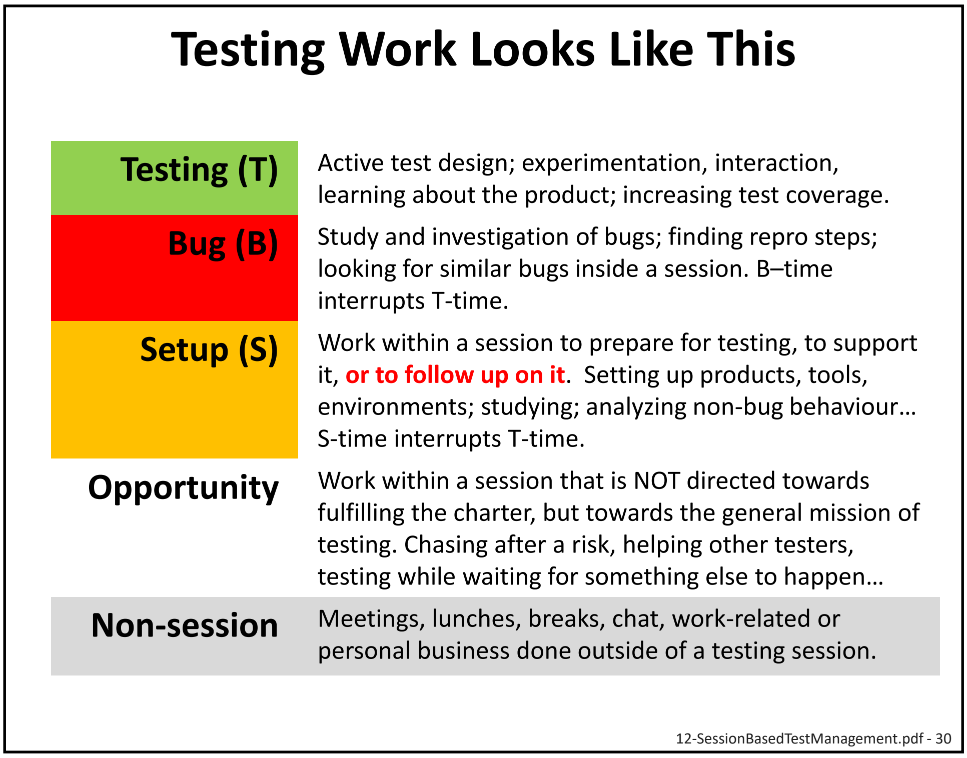
Advice for Test Reporting

- Don't use silly, misleading metrics (such as test case counts) in a test report.
- Tailor your reporting to the situation at hand.
- Learn how to tell a three-level testing story.
- To tell a good story of testing, you must have within you a comprehensive model of the testing problem and process.
- **Consider organizing your report by product risk.**



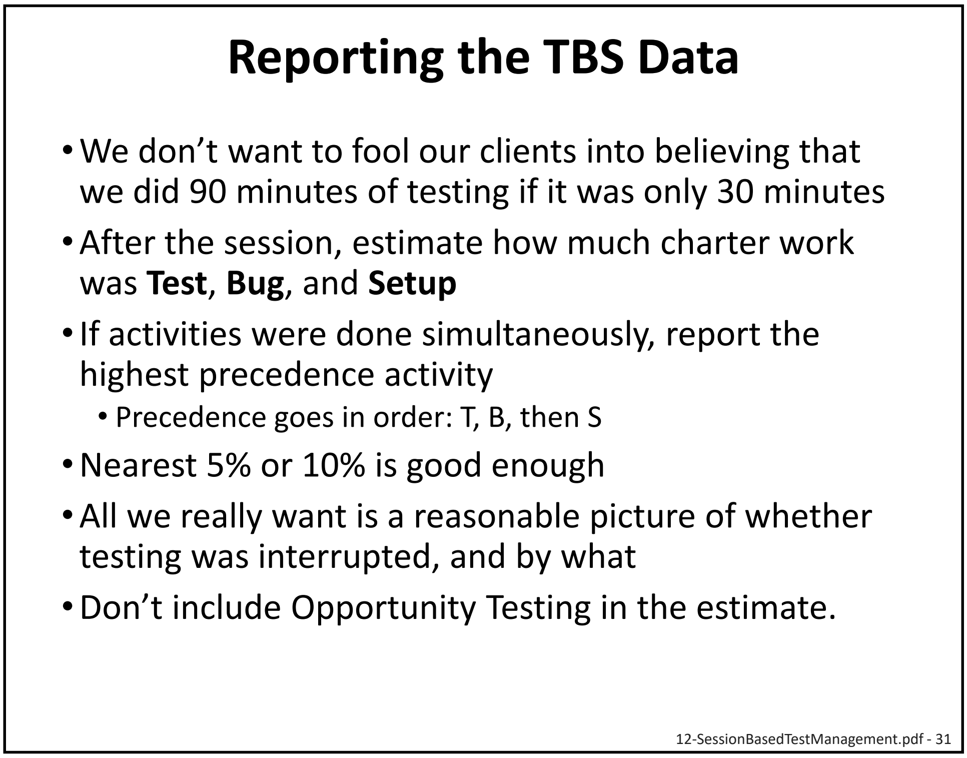
Advice for Test Reporting

- Don't use silly, misleading metrics (such as test case counts) in a test report.
- Tailor your reporting to the situation at hand.
- Learn how to tell a three-level testing story.
- To tell a good story of testing, you must have within you a comprehensive model of the testing problem and process.
- Consider organizing your report by product risk.
- **Your report is a story of what you did, not a list of the artifacts you generated.**



Testing Work Looks Like This

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



Reporting the TBS Data

- We don't want to fool our clients into believing that we did 90 minutes of testing if it was only 30 minutes
- After the session, estimate how much charter work was **Test**, **Bug**, and **Setup**
- If activities were done simultaneously, report the highest precedence activity
  - Precedence goes in order: T, B, then S
- Nearest 5% or 10% is good enough
- All we really want is a reasonable picture of whether testing was interrupted, and by what
- Don't include Opportunity Testing in the estimate.

We want PROOF!

Things to record in your notes:

- **P**ast
  - What has been happening during the session?
- **R**esults
  - What is being achieved?
- **O**bstacles
  - What's getting in the way or slowing things down?
- **O**utlook
  - What's next? What remains to be done?
- **F**eelings
  - How do you feel about all this?

Unless you want to forget a lot about what happened during the session, take notes as you test!

Consider Automatic Logging

- Note-taking is less disruptive when the product produces an automatic log of activities that were covered during testing.
- Automatic logging gives you some level of retrospective documentation on what was tested.
- You can also use external logging tools
  - BB Test Assistant
  - Camtasia
  - Spector (www.spectorsoft.com)
  - qTest Explorer

Ask for testability!

Safety Language
(aka “epistemic modalities”)

- “Safety language” in software testing, means to qualify or otherwise draft statements of fact so as to avoid false confidence.
- Examples:

I think...

So far...

It appears...

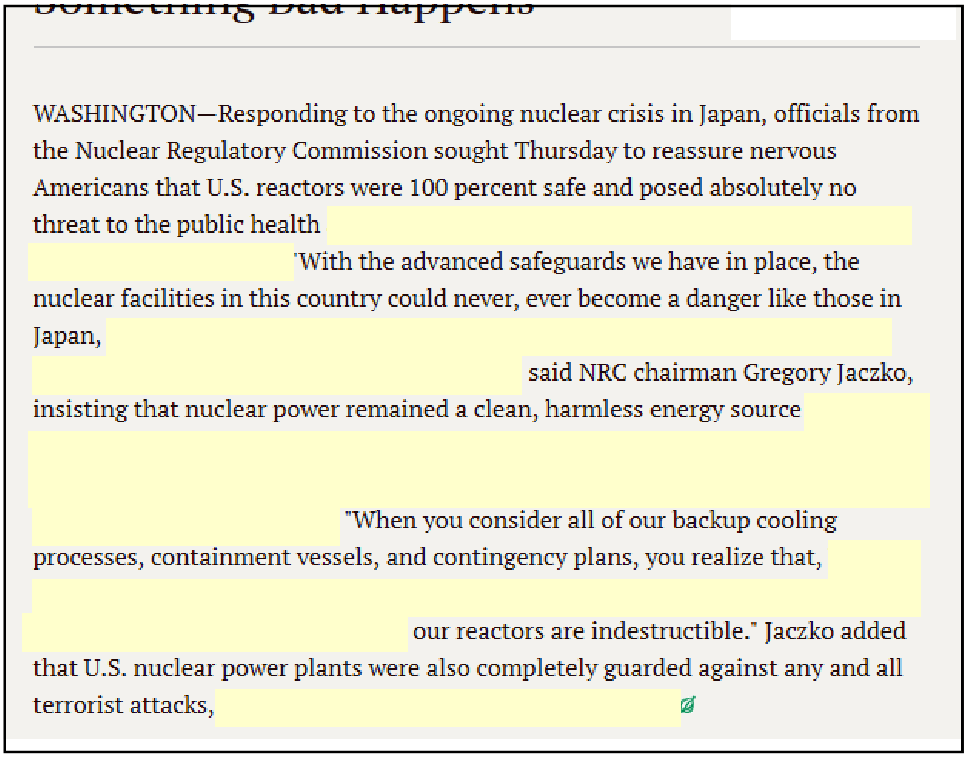
I infer...

apparently...

I assumed...

It seems...

~~*The feature worked*~~ → *I have not yet seen any failures in the feature...*



Something Bad Happens

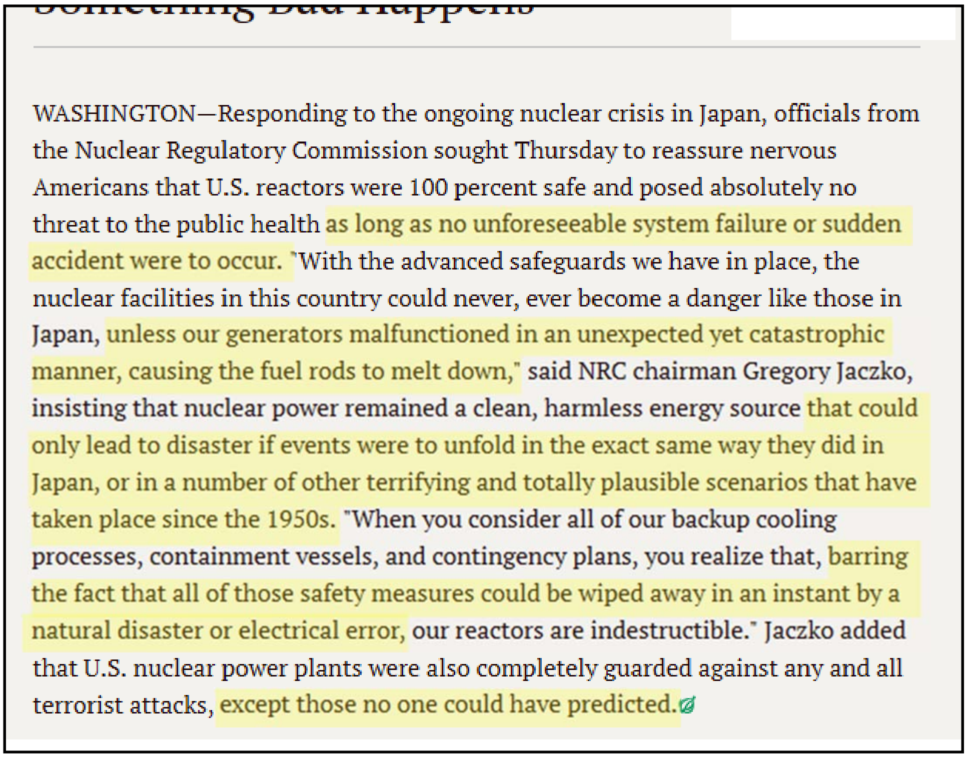
WASHINGTON—Responding to the ongoing nuclear crisis in Japan, officials from the Nuclear Regulatory Commission sought Thursday to reassure nervous Americans that U.S. reactors were 100 percent safe and posed absolutely no threat to the public health

"With the advanced safeguards we have in place, the nuclear facilities in this country could never, ever become a danger like those in Japan,

said NRC chairman Gregory Jaczko, insisting that nuclear power remained a clean, harmless energy source

"When you consider all of our backup cooling processes, containment vessels, and contingency plans, you realize that,

our reactors are indestructible." Jaczko added that U.S. nuclear power plants were also completely guarded against any and all terrorist attacks,



Something Bad Happens

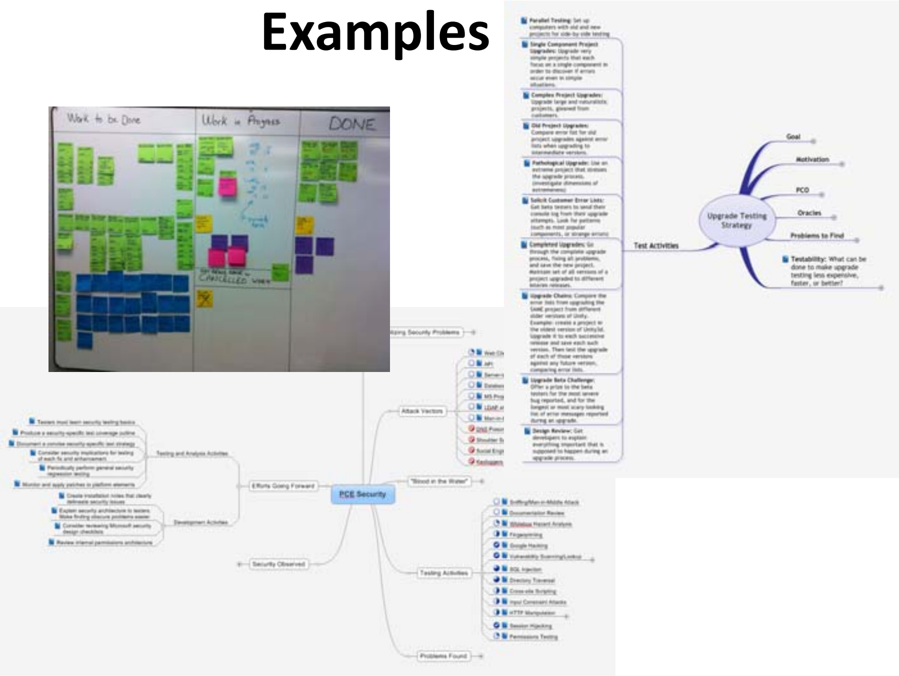
WASHINGTON—Responding to the ongoing nuclear crisis in Japan, officials from the Nuclear Regulatory Commission sought Thursday to reassure nervous Americans that U.S. reactors were 100 percent safe and posed absolutely no threat to the public health as long as no unforeseeable system failure or sudden accident were to occur. "With the advanced safeguards we have in place, the nuclear facilities in this country could never, ever become a danger like those in Japan, unless our generators malfunctioned in an unexpected yet catastrophic manner, causing the fuel rods to melt down," said NRC chairman Gregory Jaczko, insisting that nuclear power remained a clean, harmless energy source that could only lead to disaster if events were to unfold in the exact same way they did in Japan, or in a number of other terrifying and totally plausible scenarios that have taken place since the 1950s. "When you consider all of our backup cooling processes, containment vessels, and contingency plans, you realize that, barring the fact that all of those safety measures could be wiped away in an instant by a natural disaster or electrical error, our reactors are indestructible." Jaczko added that U.S. nuclear power plants were also completely guarded against any and all terrorist attacks, except those no one could have predicted.

Test Project Status

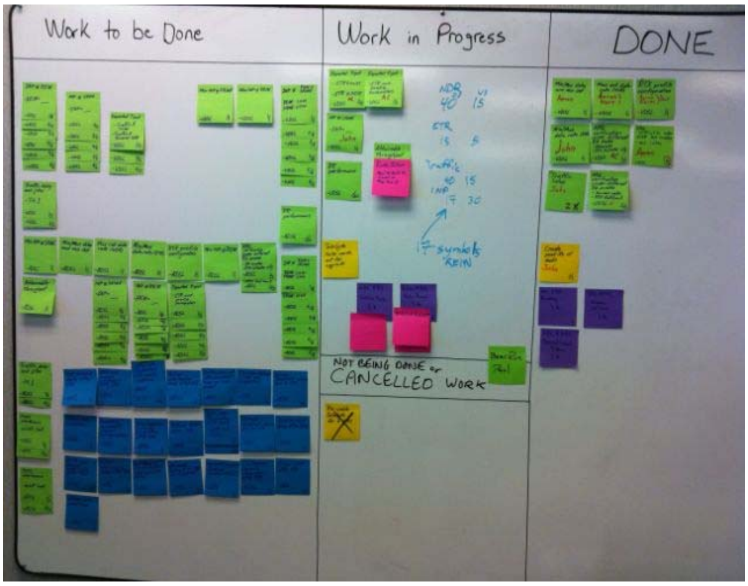
How do you report project status effectively?

Offer a compact summary that directs attention to risk.

Examples

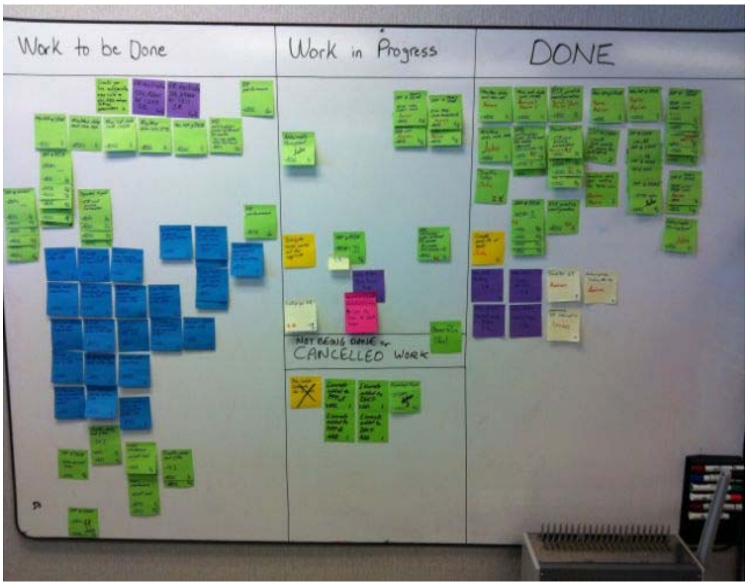

Work to be Done
Work in Progress
DONE
NOT BEING DONE or CANCELLED WORK
Parallel Testing: Set up computers with old and new projects for side-by-side testing
Single Component Project Upgrades: Upgrade very simple projects that each focus on a single component in order to discover if errors occur even in simple situations.
Complex Project Upgrades: Upgrade large and naturalistic projects, gleaned from customers.
Old Project Upgrades: Compare error list for old project upgrades against error lists when upgrading to intermediate versions.
Pathological Upgrade: Use an extreme project that stresses the upgrade process. (investigate dimensions of extremeness)
Solicit Customer Error Lists: Get beta testers to send their console log from their upgrade attempts. Look for patterns (such as most popular components, or strange errors)
Completed Upgrades: Go through the complete upgrade process, fixing all problems, and save the new project. Maintain set of all versions of a project upgraded to different Unity releases.
Upgrade Chains: Compare the error lists from upgrading the SAME project from different older versions of Unity. Example: create a project in the oldest version of Unity3d. Upgrade it to each successive release and save each such version. Then test the upgrade of each of those versions against any future version, comparing error lists.
Upgrade Beta Challenge: Offer a prize to the beta testers for the most severe bug reported, and for the longest or most scary-looking list of error messages reported during an upgrade.
Design Review: Get developers to explain everything important that is supposed to happen during an upgrade process.
Test Activities
Upgrade Testing Strategy
Goal
Motivation
PCO
Oracles
Problems to Find
Testability: What can be done to make upgrade testing less expensive, faster, or better?
PCE Security
Efforts Going Forward
Testing and Analysis Activities
Testers must learn security testing basics
Produce a security-specific test coverage outline
Document a concise security-specific test strategy
Consider security implications for testing of each fix and enhancement
Periodically perform general security regression testing
Monitor and apply patches to platform elements
Development Activities
Create installation notes that clearly delineate security issues
Explain security architecture to testers. Make finding obscure problems easier.
Consider reviewing Microsoft security design checklists
Review internal permissions architecture
Security Observed
Attack Vectors
"Blood in the Water"
Testing Activities
Sniffing/Man-in-Middle Attack
Documentation Review
Whitebox Hazard Analysis
Fingerprinting
Google Hacking
Vulnerability Scanning/Lookup
SQL Injection
Directory Traversal
Cross-site Scripting
Input Constraint Attacks
HTTP Manipulation
Session Hijacking
Permissions Testing
Problems Found

Visualizing Test Progress

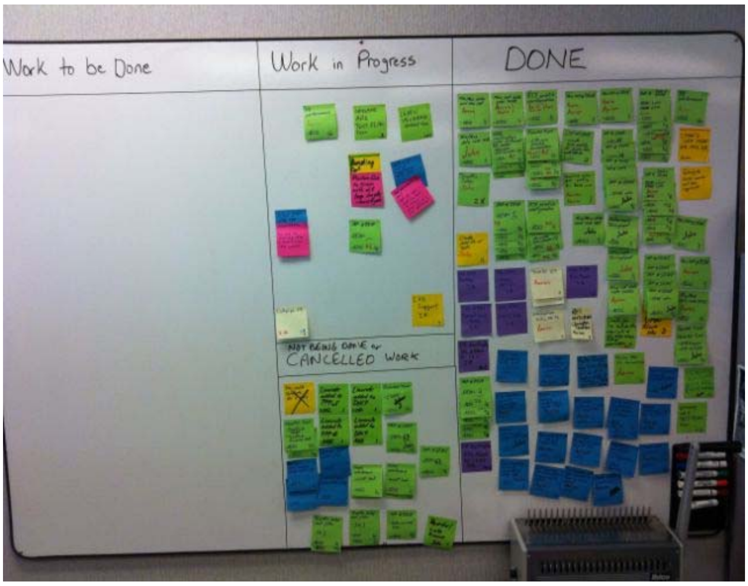


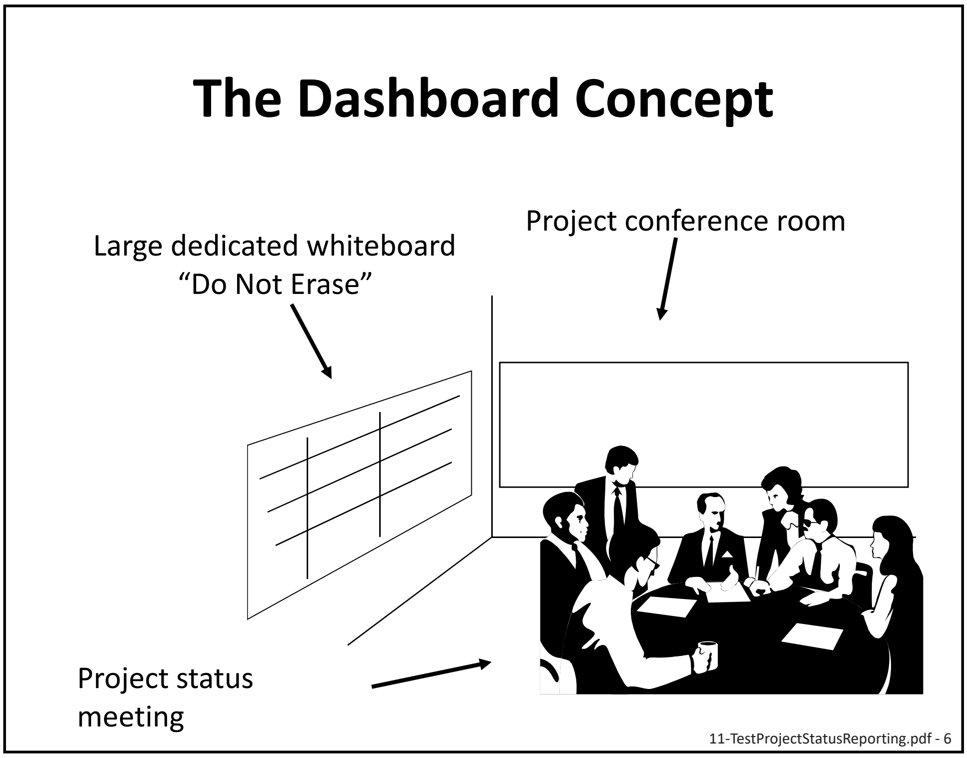
Google for "A Sticky Situation DevelopSense"

Visualizing Test Progress



Visualizing Test Progress





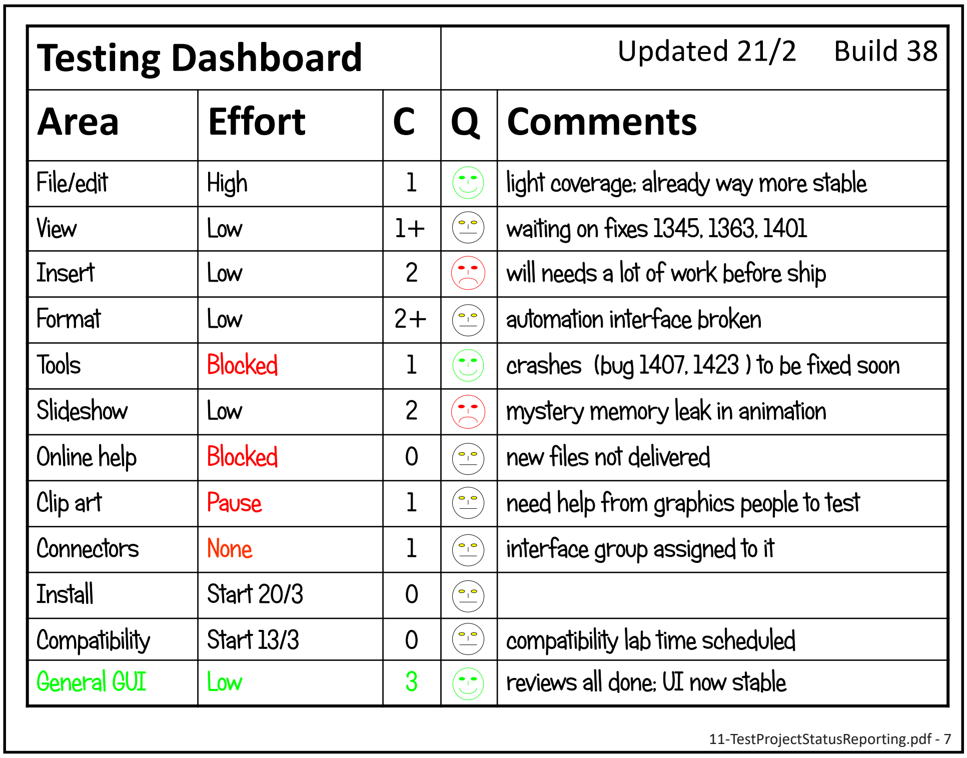
The Dashboard Concept

Large dedicated whiteboard
“Do Not Erase”

Project conference room

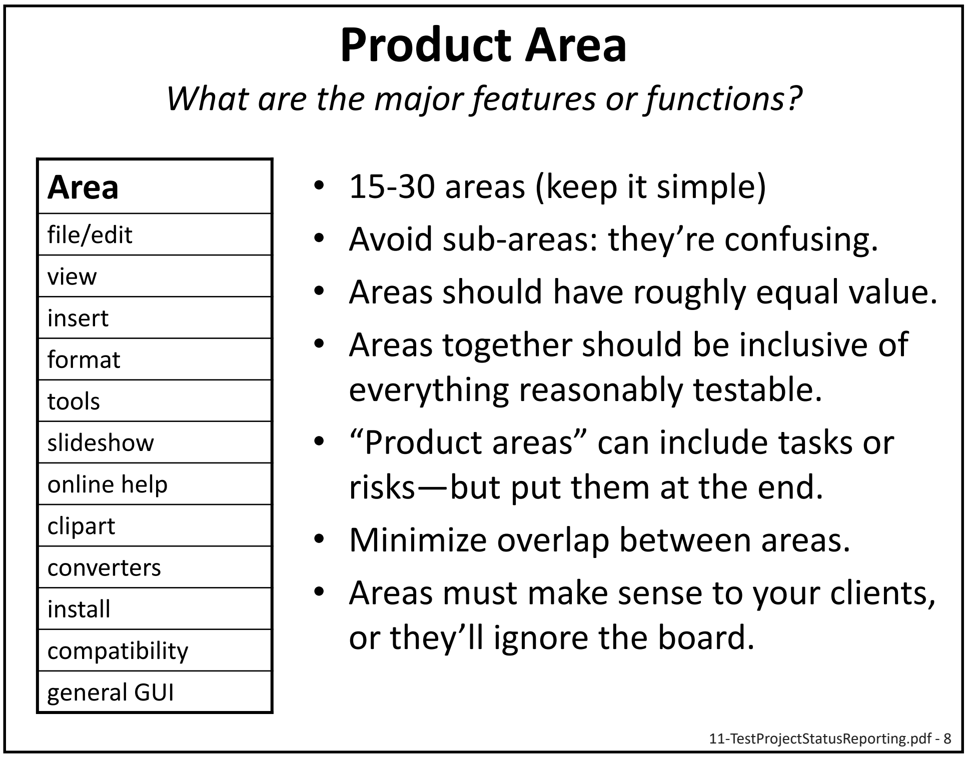
Project status
meeting

11-TestProjectStatusReporting.pdf - 6



Testing Dashboard			Updated 21/2	Build 38
Area	**Effort**	**C**	**Q**	**Comments**
File/edit	High	1	☺	light coverage: already way more stable
View	Low	1+	😐	waiting on fixes 1345, 1363, 1401
Insert	Low	2	☹	will needs a lot of work before ship
Format	Low	2+	😐	automation interface broken
Tools	Blocked	1	☺	crashes (bug 1407, 1423) to be fixed soon
Slideshow	Low	2	☹	mystery memory leak in animation
Online help	Blocked	0	😐	new files not delivered
Clip art	Pause	1	😐	need help from graphics people to test
Connectors	None	1	😐	interface group assigned to it
Install	Start 20/3	0	😐	
Compatibility	Start 13/3	0	😐	compatibility lab time scheduled
General GUI	Low	3	☺	reviews all done: UI now stable

11-TestProjectStatusReporting.pdf - 7

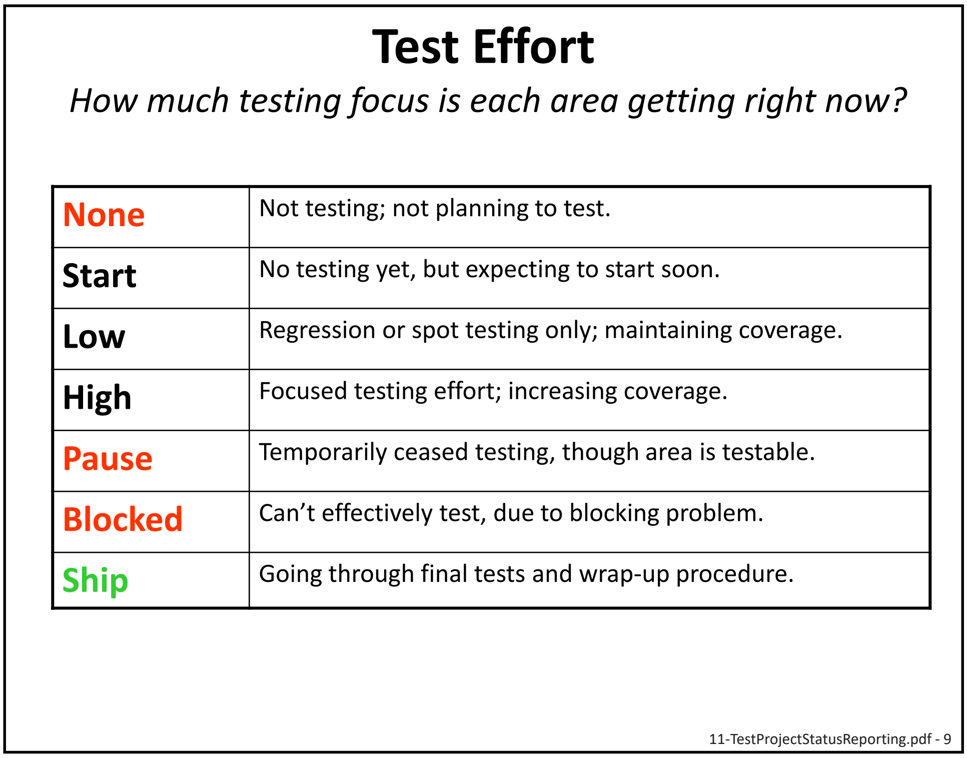


Product Area

What are the major features or functions?

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

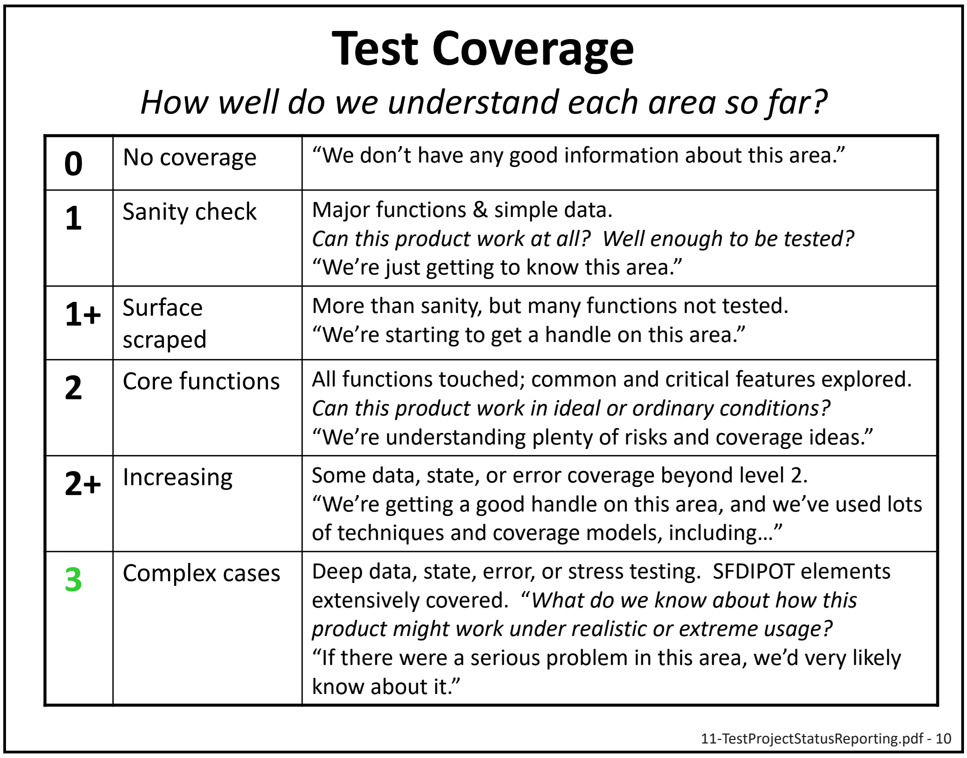
- 15-30 areas (keep it simple)
- Avoid sub-areas: they're confusing.
- Areas should have roughly equal value.
- Areas together should be inclusive of everything reasonably testable.
- "Product areas" can include tasks or risks—but put them at the end.
- Minimize overlap between areas.
- Areas must make sense to your clients, or they'll ignore the board.



Test Effort

How much testing focus is each area getting right now?

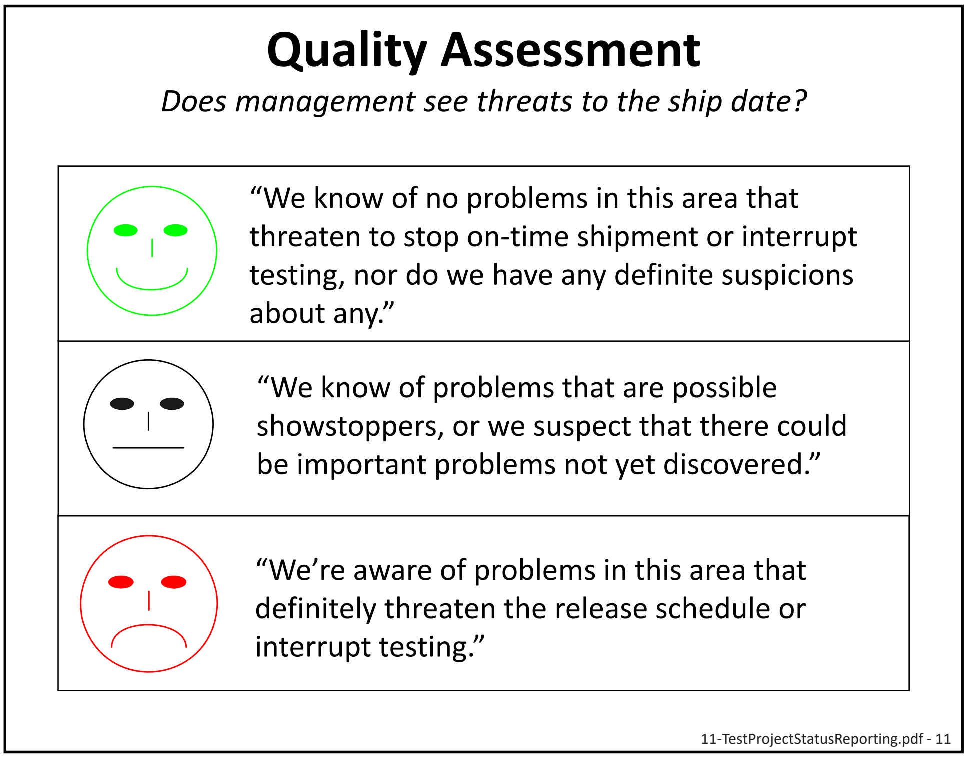
None	Not testing; not planning to test.
Start	No testing yet, but expecting to start soon.
Low	Regression or spot testing only; maintaining coverage.
High	Focused testing effort; increasing coverage.
Pause	Temporarily ceased testing, though area is testable.
Blocked	Can't effectively test, due to blocking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



Test Coverage

How well do we understand each area so far?

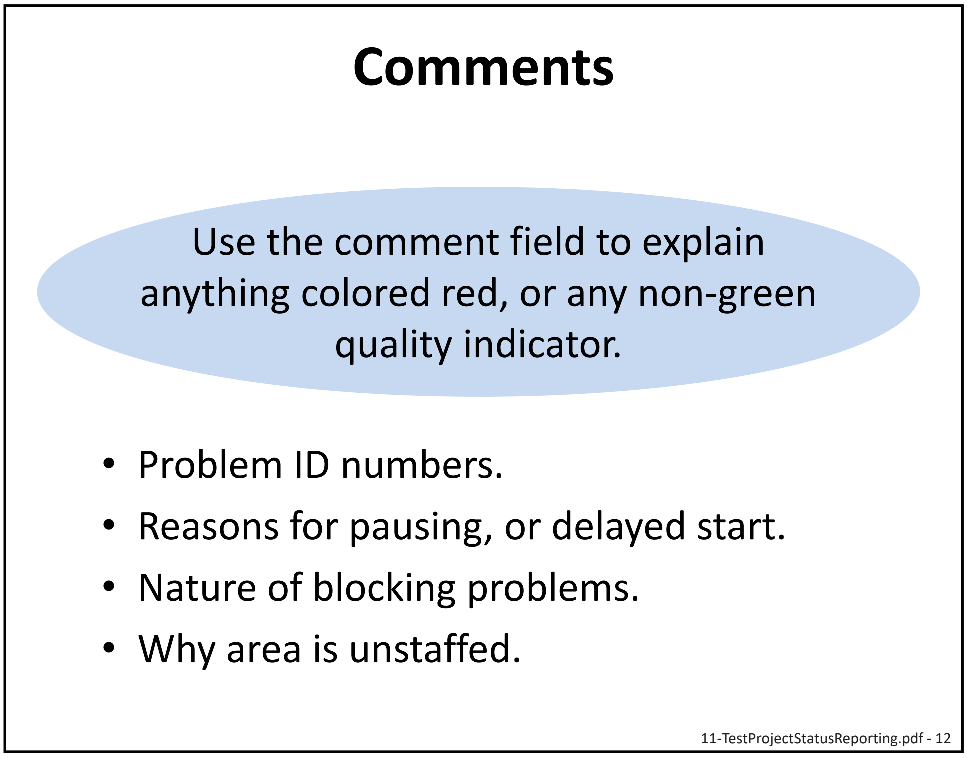
0	No coverage	"We don't have any good information about this area."
1	Sanity check	Major functions & simple data. *Can this product work at all? Well enough to be tested?* "We're just getting to know this area."
1+	Surface scraped	More than sanity, but many functions not tested. "We're starting to get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



Quality Assessment

Does management see threats to the ship date?

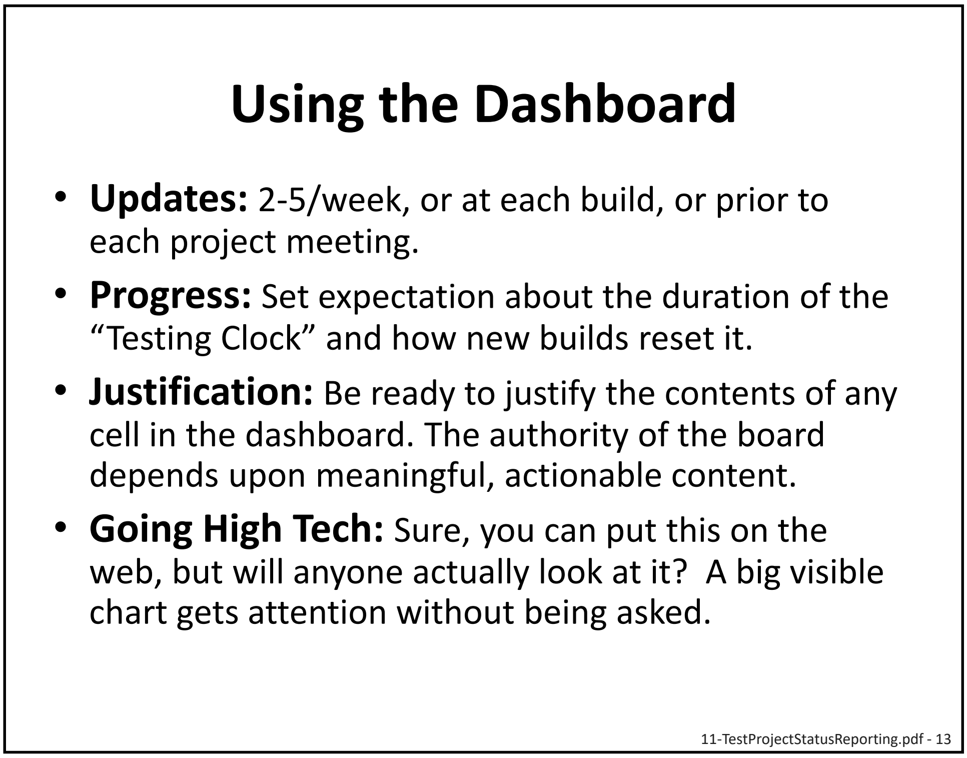
☺	"We know of no problems in this area that threaten to stop on-time shipment or interrupt testing, nor do we have any definite suspicions about any."
😐	"We know of problems that are possible showstoppers, or we suspect that there could be important problems not yet discovered."
☹	"We're aware of problems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



Comments

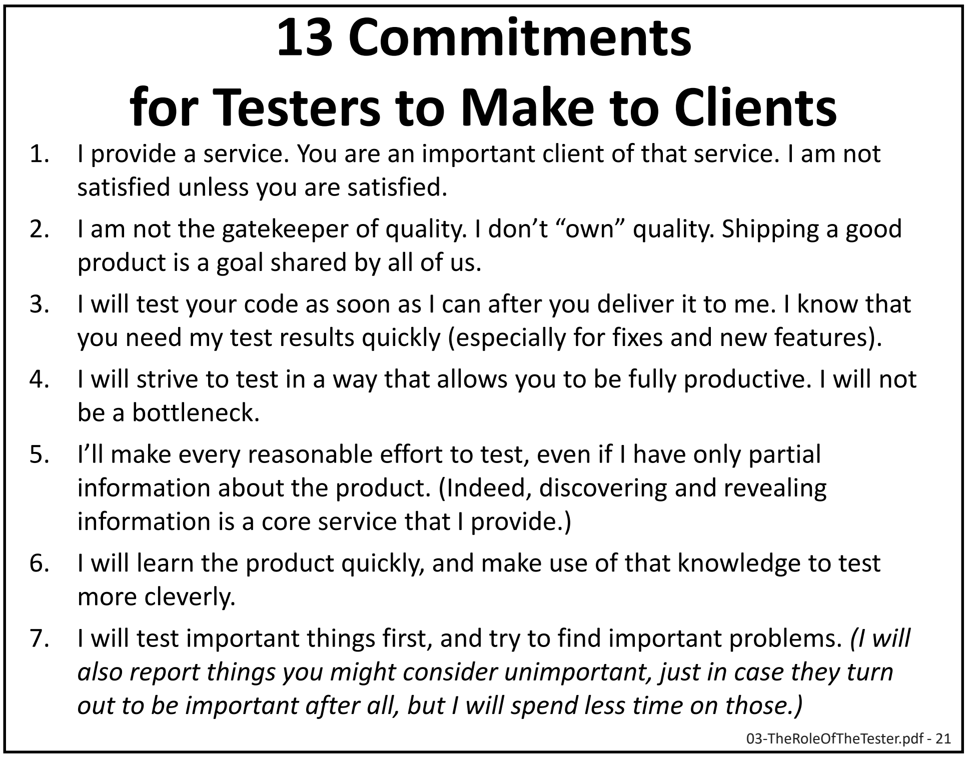
Use the comment field to explain anything colored red, or any non-green quality indicator.

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



Using the Dashboard

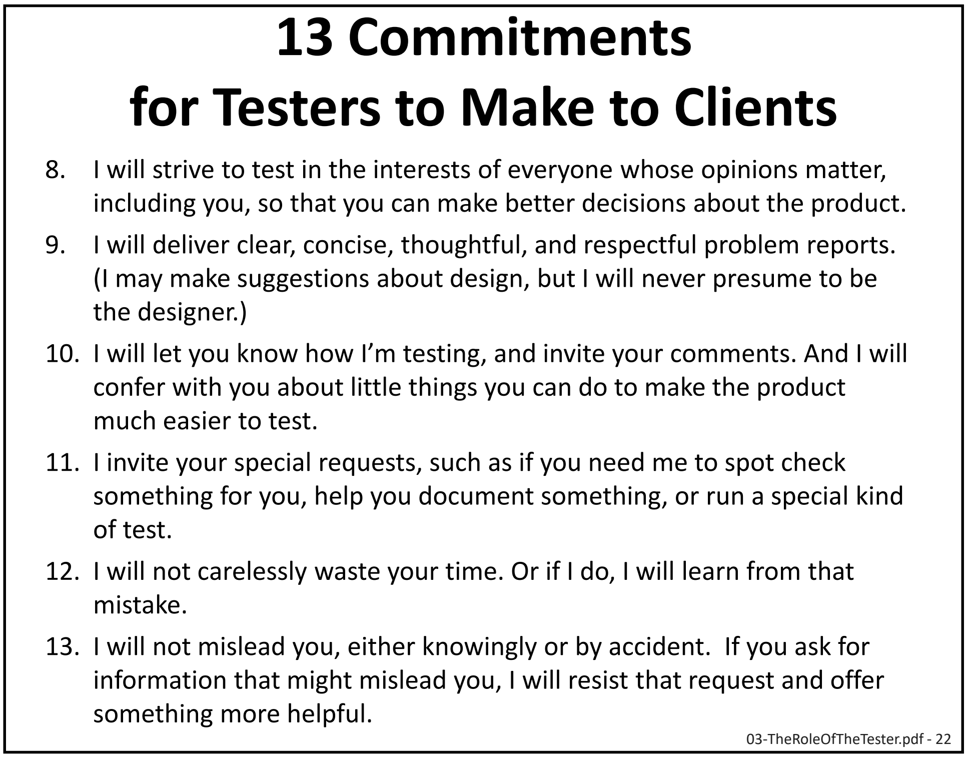
- **Updates:** 2-5/week, or at each build, or prior to each project meeting.
- **Progress:** Set expectation about the duration of the "Testing Clock" and how new builds reset it.
- **Justification:** Be ready to justify the contents of any cell in the dashboard. The authority of the board depends upon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



13 Commitments for Testers to Make to Clients

1. I provide a service. You are an important client of that service. I am not satisfied unless you are satisfied.
2. I am not the gatekeeper of quality. I don’t “own” quality. Shipping a good product is a goal shared by all of us.
3. I will test your code as soon as I can after you deliver it to me. I know that you need my test results quickly (especially for fixes and new features).
4. I will strive to test in a way that allows you to be fully productive. I will not be a bottleneck.
5. I’ll make every reasonable effort to test, even if I have only partial information about the product. (Indeed, discovering and revealing information is a core service that I provide.)
6. I will learn the product quickly, and make use of that knowledge to test more cleverly.
7. I will test important things first, and try to find important problems. *(I will also report things you might consider unimportant, just in case they turn out to be important after all, but I will spend less time on those.)*

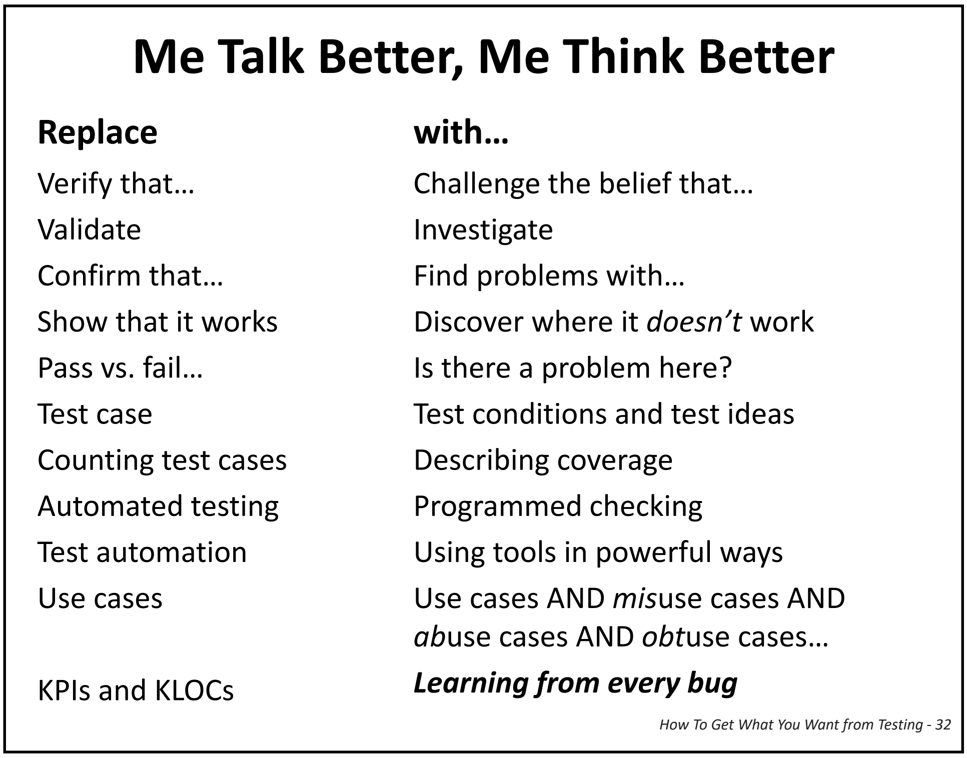
03-TheRoleOfTheTester.pdf - 21



13 Commitments for Testers to Make to Clients

8. I will strive to test in the interests of everyone whose opinions matter, including you, so that you can make better decisions about the product.
9. I will deliver clear, concise, thoughtful, and respectful problem reports. (I may make suggestions about design, but I will never presume to be the designer.)
10. I will let you know how I’m testing, and invite your comments. And I will confer with you about little things you can do to make the product much easier to test.
11. I invite your special requests, such as if you need me to spot check something for you, help you document something, or run a special kind of test.
12. I will not carelessly waste your time. Or if I do, I will learn from that mistake.
13. I will not mislead you, either knowingly or by accident. If you ask for information that might mislead you, I will resist that request and offer something more helpful.

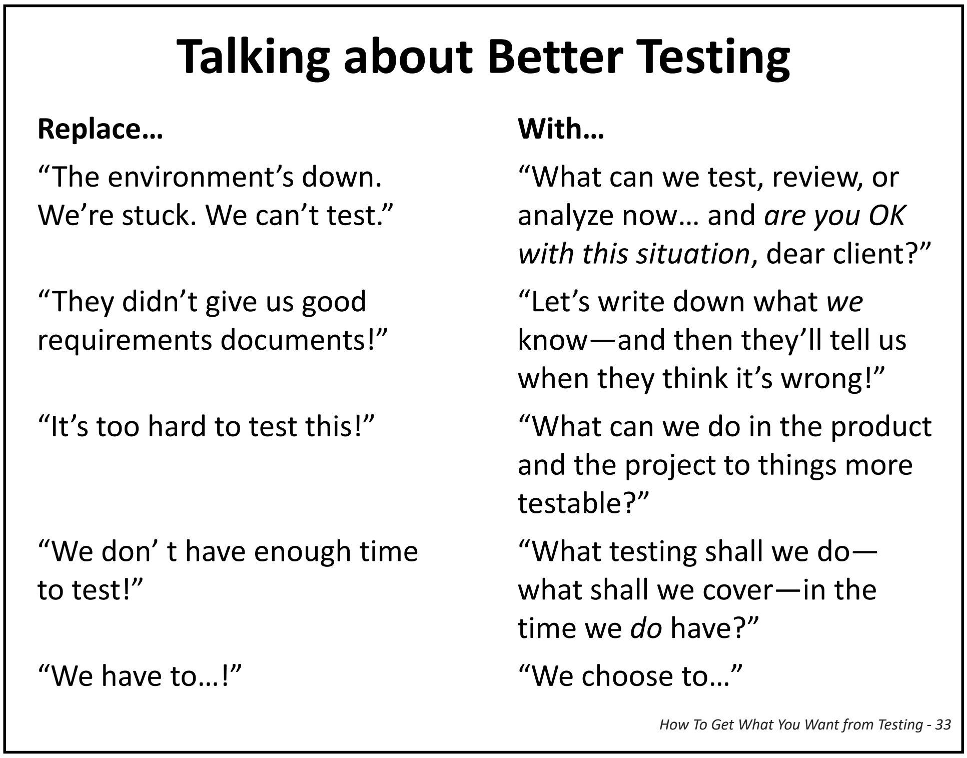
03-TheRoleOfTheTester.pdf - 22



Me Talk Better, Me Think Better

Replace	**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?
Test case	Test conditions and test ideas
Counting test cases	Describing coverage
Automated testing	Programmed checking
Test automation	Using tools in powerful ways
Use cases	Use cases AND *mis*use cases AND *ab*use cases AND *obt*use cases…
KPIs and KLOCs	***Learning from every bug***

How To Get What You Want from Testing - 32



Talking about Better Testing

Replace…	**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…”

How To Get What You Want from Testing - 33