Two Futures of Software Testing

Michael Bolton
DevelopSense

TestNet

May 2011

Who I Am

Michael Bolton

(not the singer, not the guy in Office Space)

DevelopSense, Toronto, Canada

mb@developsense.com

+1 (416) 992-8378

http://www.developsense.com

Acknowledgements

- James Bach
  - some of the material comes from the Rapid Software Testing Course, of which James is the senior author and I am co-author
- Cem Kaner
- Bret Pettichord
- Jerry Weinberg
- Jonathan Kohl
- TestNet

These are not *predictions*.

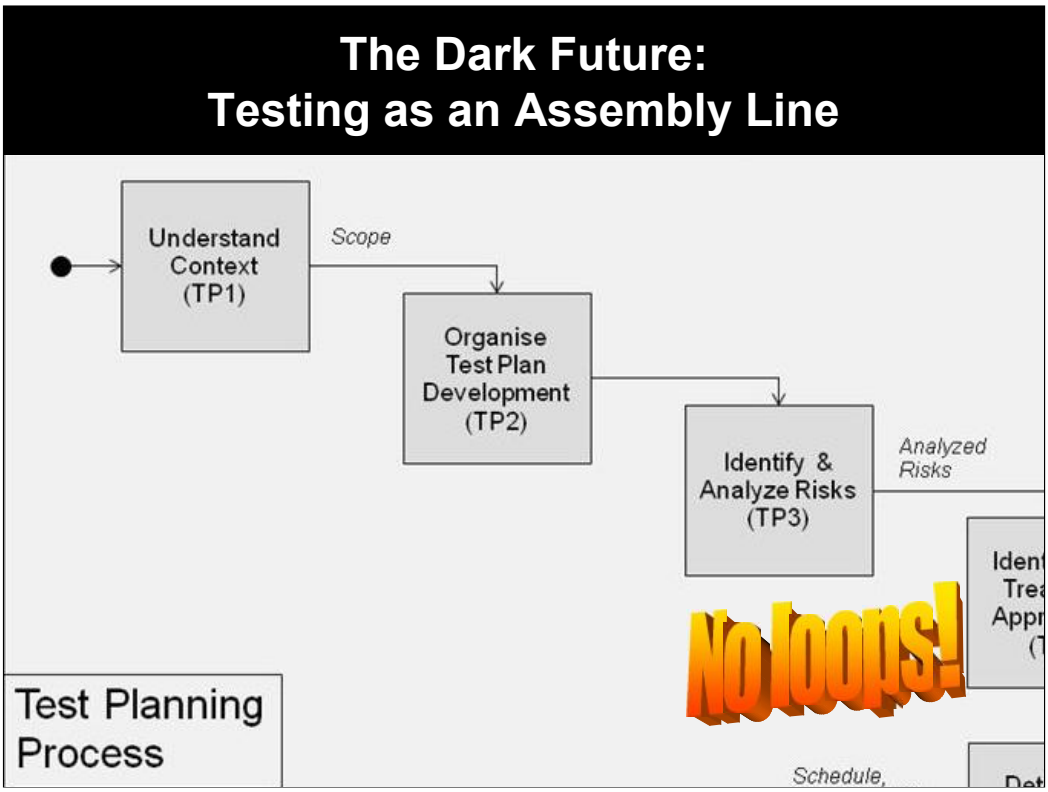
These are *proposals*.

These are not the only two futures.
They're offered for your consideration.
The choices are up to you.

The Dark Future: The Tester's Role is to Inhibit Change

- Nothing is more important than following our plans and our processes strictly
- If our clients want change, we'll accuse them of being "against quality".

In the Dark Future, it is the role of the tester—excuse me, the *Quality Assurance Analyst*—to inhibit change. Change brings a chance of invalidating things that we believe we know about the product and the project, and thus change involves risk. So even though the customer needs, the market conditions, the schedule, the budget, the product scope, the staff, and everything else about the project might change, we should stay the course and stick to the plan. It doesn't matter if we learn things through the course of developing the product; we should have known those things beforehand. It's not merely our job to inform; we must also *enforce*.



In the Dark Future, ISO Standard 29119 will tell us what to test and how to test it. *"Whatever type of testing you do, it will affect you."* It doesn't matter if the people who drafted the standard know *your* business; they're experts, and they know what's good for you better than you do. *"The standard uses a four layer process model starting with two organizational layers covering test policy and organizational test strategy. The next layer moves into project management, while the bottom layer defines the fundamental test process used for all levels of testing, such as unit testing, system testing, acceptance testing, and the test types (e.g. performance and usability testing). Parts 2 and 3, on process and documentation respectively, are particularly closely linked as all outputs from the test processes potentially correspond to documents defined in the document part. There is also a 'new work item' being suggested that would see a fifth part initiated on test process improvement – imagine a testing industry without the emergence of another new test improvement model every couple of years."* Doesn't that sound swell? Not only will they be telling you what to do, but also how to improve it–despite the acknowledged caveat, *"Probably the biggest complaint raised against IT standards is that they do not meet the needs of actual practitioners – most of us have come across such 'shelfware'."* Don't worry about the standard being unmanageable, either. The current draft of Part 2 of the standard is, as of this writing, a mere 100 pages.

Note also that there is a standard vocabulary associated with the standard. That standard vocabulary will be in English. Translating it into other languages will only increase complexity and ambiguity. Let's all just test in English. If other cultures don't like that… well, tough. There's not much to learn from them anyway.

The Dark Future: Promoting Orthodoxy

- All testers must be certified with easy-to-pass multiple choice exams
  - testing doesn't really require skilled labour anyway
- Everyone in "the testing industry" uses the same language and tests to a standard

In the Dark Future, testers will be evaluated based on their ability to memorize testing terms from a particular authority's body of knowledge. Context or interpretation have no place in the Dark Future. Exams should always be set up for the convenience of the certifiers, so multiple choice is definitely the way to go here. If there are concerns that this approach is insufficient to evaluate skills, no worries: testing isn't an especially skilled trade anyway. Some testers are able to write code to automated the work, which is a good idea because testing is mostly an uninteresting, repetitious, confirmatory task anyway.

We don't want testers to be hobnobbing with the developers (that is, the programmers—but programmers are the only developers in the Dark Future). Testers are too weak-willed to avoid the pernicious influence of programmers, so mingling might compromise the testers' objectivity. Testers might even be tempted not to report bugs.

Repeatability is very important in the Dark Future. We want to run the same tests over and over, without variation, because variation might lead to unpredictability. Discovering and investigating bugs could throw our whole schedule off. So i

The Dark Future: Testing ISN'T About Learning

- Testing is focused on confirmation, verification, and validation
- Testers check to make sure that prescribed tests pass.
- Exploration and investigation are luxuries at best, threats at worst

In the Dark Future, testing is a relentlessly routine, mechanical activity, even when it's done by humans. It's not about learning, it's about confirming things that we already know, answering questions for which we already have the answer, repeating the same mindless tests over and over again. There's no place in the Dark Future for exploration, investigation, or discovery, or learning, and that means that there's no place for skill, creativity, or imagination. Nor is there room for asking questions about the customers and how they might value our product. We just do what we're told, and we learn nothing.

The Dark Future:
Testing is reduced to non-sapient *checking*

"Sapient" means "requiring human wisdom"

A ***non-sapient*** activity can be performed

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

by a human who has been
instructed NOT to think
(and who is slow and erratic)

The Dark Future: Putting The Testers In Charge

Project management is not *mature* enough to make *proper* decisions.

In the Dark Future, testers are quality gatekeepers. We decide when to start testing, and we only do so when the product and the accompanying documentation adhere to our rigourous standards, and when we've received We decide whether the product is of sufficient quality to ship. Managers must obtain our signature and our permission to be sure that they're releasing a quality product. We can block releases if the product isn't good enough for us. We decide when to start testing, and we do so only. We're not obliged to follow these standards ourselves, of course; that's not our role. In the Dark Future, our role is to tell other people what they're doing wrong and how to do it right. In the dark future, t

The Dark Future: Not Putting The Testers In Charge

Testers don't have control over schedules, budgets, product scope, staffing, contracts, and so on… but we're still responsible for quality.

The Dark Future: Measurement

- We measure…
  - requirements scope by *counting requirements*
  - test coverage by *counting test cases*
  - product quality by *counting bugs*
  - the value of testers by *counting bug reports*
  - programmer output by *counting lines of code*
  - complexity by *counting code branches*
- The relevance of the number to the attribute doesn't matter.
- *So simple a child can do it!*

Requirements, productivity, complexity, test coverage, product quality, and tester value are influenced by dozens, hundreds of factors that we could observe. Yet most of these factors are not tangible or countable in a meaningful way, and simplistic attempts to count instances of them are practically guaranteed to mislead. In the Dark Future, we make these problems go away by *ignoring them*.

A bug is not a thing in the world. A bug is a *construct*; thought-stuff; a mental thing. It's a relationship between some person and some product, such that some other person might not view it as a bug. Even when two people or more agree that some behaviour seems to be a bug, they may disagree on the significance of the bug. Despite this, in the Dark Future, we'll just count 'em. More bugs means higher quality; fewer bugs means lower quality. That applies to testers too. We'll ignore all the other activities and dimensions of value that a tester might bring to a project, and count their bug reports to measure their effectiveness.

We'll simply apply the idea that there should be one test case traceable to each requirement. No, wait! Two! One *positive* test case and one *negative* test case.

Cem Kaner has said that a test case is a question that we want to ask about the product. As James Bach has said many times, a test case is a container for a question. In the Dark Future, we'll evaluate the quality of work in an office by counting the briefcases that come in the door every morning. We won't bother to look inside them. If more briefcases come in, it's obvious that the quality of the company's work will improve.

We'll certainly ignore problems associated with simple metrics by avoiding *Software Engineering Metrics: What Do They Measure and How Do We Know?* by Cem Kaner and Pat Bond (http://www.kaner.com/pdfs/metrics2004.pdf); the classic *How To Lie With Statistics*, by Darrell Huff; *Quality Software Management, Vol. 2: First Order Metrics* by Gerald M. Weinberg; and especially *Measuring and Managing Performance in Organizations*, by Robert D. Austin.

Einstein said that "not everything that counts can be counted; and not everything that can be counted counts." We'll ignore him too.

In the Dark Future,

esters have blame without responsibility, culpability without authority. Since they were the last people to have their hands on the code, it is assumed that any undetected problems are their fault. Testers are required to sign documents asserting that the product is acceptable for release, even though the release of the product is a business decision, rather than a technical one.

In the Dark Future, all product failures are seen as testing failures. There's no recognition that problems are problems for the whole development team. Read the daily newspaper, and you'll see over and over that problems are pinned on poorly tested software. Not poorly *programmed* software, nor poorly managed projects, not poorly conceived products, not poorly developed requirements. In the popular view, product problems are testing problems; no more, no less. In this view, if software development were a sport, the entire responsibility for the loss of a game would be laid on the goalie.

The Bright Future:
Testers Light The Way

This is our role.

We see things for what they are.
We make informed decisions about quality possible,
because we think critically about software
BUT
We let project owners make the business decisions.

The Bright Future:
Testers Embrace Change and Complexity

- The real world is messy and complex
- Change WILL happen
  - in market conditions…
  - contracts…
  - requirements…
  - specifications…
  - designs…
  - documents…
  - products…
  - systems…
- We help our clients understand the *implications* of change and complexity

The Bright Future: Adaptability over Repeatability

- Repeatability, for computers, is relatively easy, but testing is not mere repetition. It's an open search.
- Skilled testing therefore focuses on thinking and looking for *adaptability*, *value*, and *threats to value*

This kind of testing can NOT be scripted.

- A skilled tester doesn't ask, "Pass or fail?"
- A skilled tester asks

Is there a problem here?

The Movement to Standardize Testing

- A standard approach to testing works brilliantly …if you only want to find standard bugs.
- How has the "standardized tester" worked out
  - for the testing community at large?
  - for individual testers?
  - for organizations who have fallen for the marketing?
  - AND for a small group of certification salespeople?
- Ask yourself:
  - 150,000 testers times (at least) $100 per exam… where has that (at least) $15,000,000 gone?
  - Who is most aggressively promoting ISO 29119?

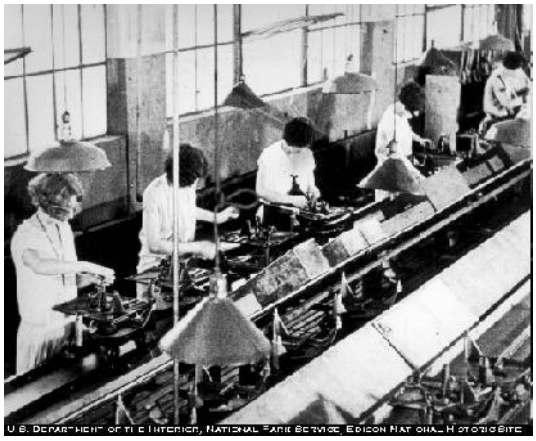
My Alternative to Certification: Being Prepared for Any Testing Mission

- I practice and teach *testing*
  - whereby I gain experience by succeeding and failing
- I practice critical thinking
  - whereby I try to avoid fooling myself and others
- I practice systems thinking
  - whereby I learn to see the big and small pictures
- I practice programming
  - whereby I obtain humility
- I practice describing my practices
  - orally
  - in writing (magazine articles, blogs, etc.)
  - in presentations (like this one)
- I participate in a community that works this way.

My Alternative to Certification
Bringing Ideas and Knowledge to the Job

- I read books and articles that are *not* about testing
  - science and physics
  - mathematics and statistics
  - cognitive psychology and critical thinking
  - computer programming and software design
  - food and cooking
  - general systems
  - medicine
  - economics
  - social sciences
  - history
  - comedy
- I relate these disciplines to testing, and describe the value of the relationships

Software Development Is Not Much Like Manufacturing



- In manufacturing, the goal is to make zillions of widgets *all the same*.
- Repetitive checking makes sense for manufacturing, but…
- In software, creating zillions of identical copies is not the big issue.

Software Development Is More Like Design

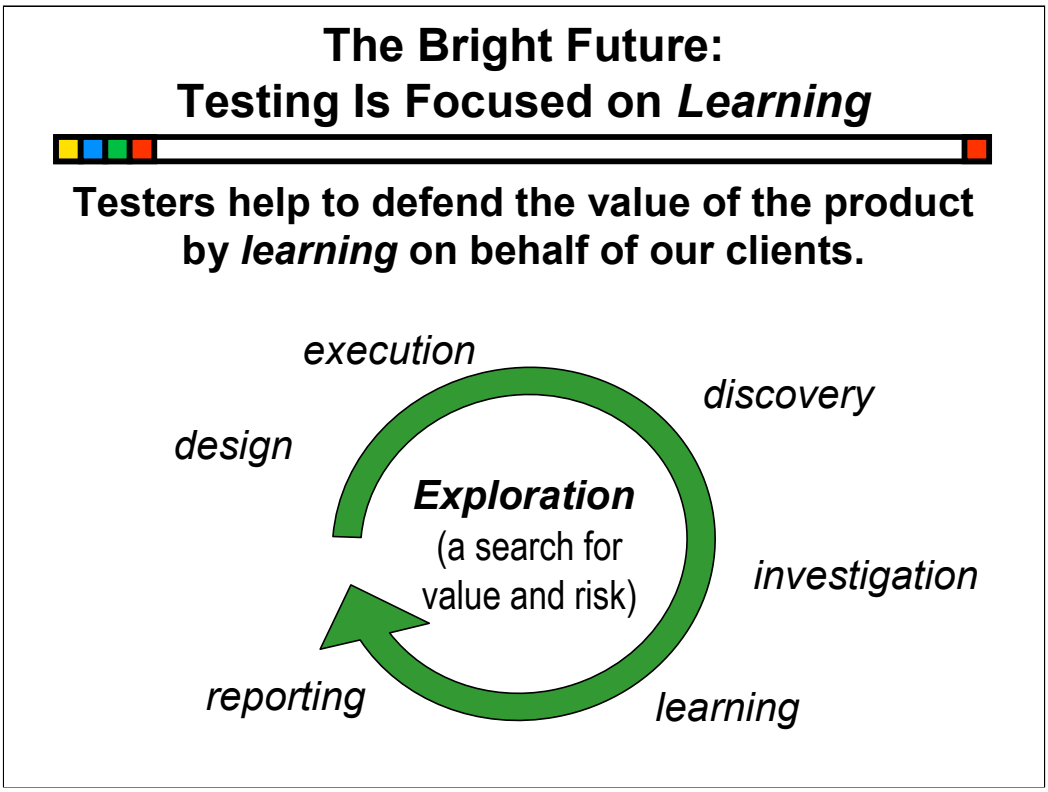
- Each new software product is novel to some degree, which means a new set of relationships and designs every time.
- New designs cannot be checked only; they must be *tested*.

If existing products sufficed, we wouldn't bother create a new one.

Testing of *Design* Is Like Work in a Crime Lab

- There is too much evidence to test anything like all of it
- There are many tools, procedures, sources of evidence.
- Tools and procedures don't *define* an investigation or its goals.
- Tools are often expensive
- Investigators work under conditions of uncertainty and extreme time pressure
- Our clients (not we) make the decisions on how to proceed based on the available evidence

These ideas come largely from Cem Kaner, *Software Testing as a Social Science*
http://www.kaner.com/pdfs/KanerSocialScienceSTEP.pdf



We're not in the business of confirming beliefs. We're in the business of demolishing unwarranted beliefs.

Testers Are Like Sensory Instruments for the Project Community

So What Are We Testers?

Skilled investigators

Software testing is the investigation of *systems* composed of people, computer programs, and related products and services.

The tester doesn't have to reach conclusions or make recommendations about how the product *should* work. **Her task is to expose credible concerns to the stakeholders**.

- Cem Kaner, *Approaches to Test Automation*, 2009 (my emphases)

What Is Testing?

- Excellent testing is not merely a branch of computer science
  - focus only on programs, and you leave out questions of *value* and other relationships that include people
- To me, excellent testing is more like *anthropology*—interdisciplinary, systems-focused, investigative, storytelling

Biology

Archaeology

Language

Culture

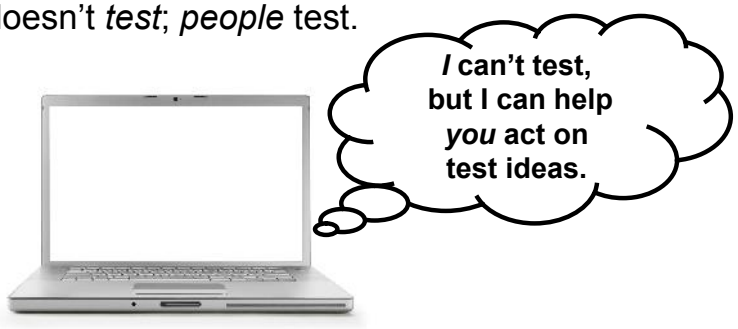
Anthropology, the study of humankind, is intensely investigative and interdisciplinary.

Anthropologists know that different cultures exist to bind people together, and to solve specific, local problems. As Wade Davis puts it so beautifully, what we might think of as primitive cultures are not failed attempts to be modern. "When asked the fundamental question, 'What does it mean to be human?', mankind responds in seven thousand different voices."

True process *maturity* would recognize the need for different approaches to deal with different testing missions.

The Bright Future: Automation Has *Many* Purposes

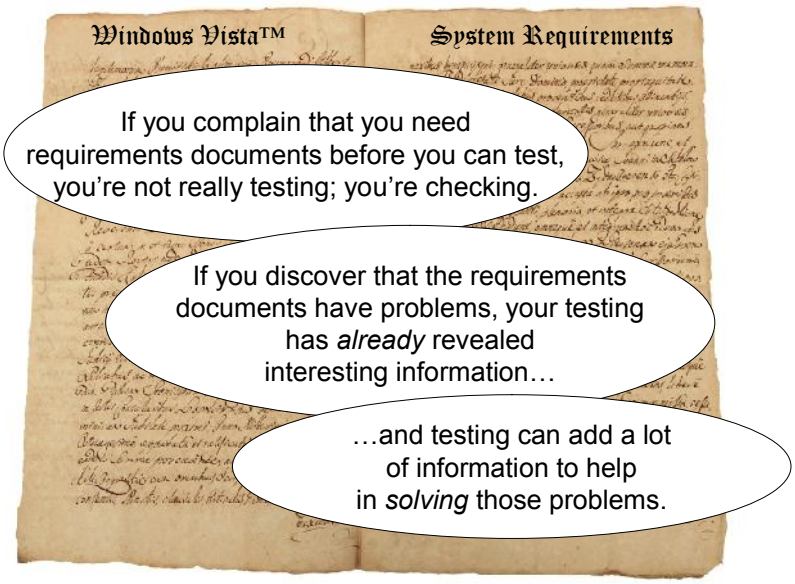
- The tester's mindset and skill set are at the centre of an exploratory process; programming is only one skill
- Automation extends our capacity to generate data, visualize, analyze, sort, search, observe, interpret…
- Automation doesn't *test*; *people* test.



See http://www.developsense.com/2009/08/testing-vs-checking.html

The Client, Not The Tester, Is In Charge

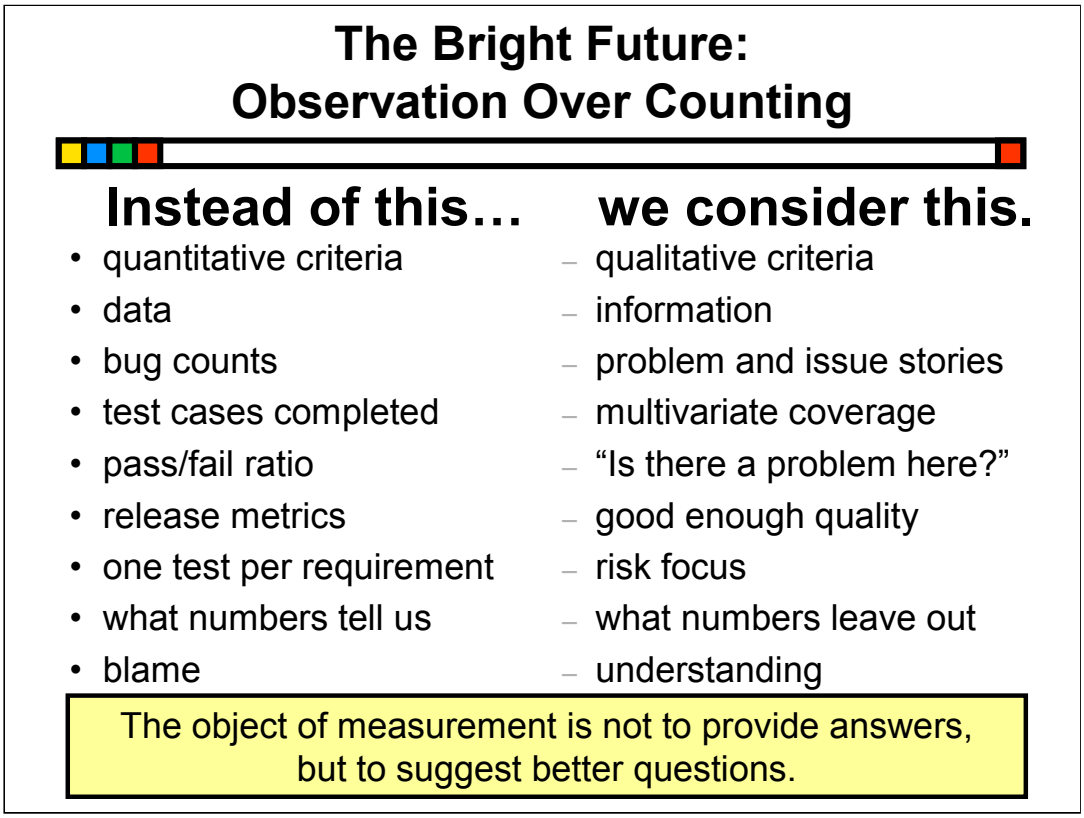
"Inadequate" Requirements Documents? No problem!



If you complain that you need requirements documents before you can test, you're not really testing; you're checking.

If you discover that the requirements documents have problems, your testing has *already* revealed interesting information…

…and testing can add a lot of information to help in *solving* those problems.



A requirement is not a line or a paragraph in a document; those things are representations—literally *re-presentations*—of the difference between what someone has and what someone desires. Counting a requirement by counting a line in a requirements document ignores everything about the meaning and the significance of the requirement, like counting tricycles and space shuttles as equivalent.

People say that bad metrics are "better than nothing". That's like saying that death by "friendly fire" is better than not shooting.

In domains where decisions are political and emotional, decisions will always be made on the basis of whose values matter, and how they feel about things. The object of measurement is not to provide answers, but to suggest better questions.

How Do We Measure?

- Third-order measurement
  - highly instrumented, used to discover natural laws
  - "What *will* happen? What *always* happens?"

- Second-order measurement
  - often instrumented, used to refine first-order observation
  - used to tune existing systems
  - "What's *really* going on here, exactly? What's happening?"

People often quote Lord Kelvin: "I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of *science*, whatever the matter may be."[1] But, few note the sentence that precedes the passage: "In *physical* science the first essential step in the direction of learning any subject is to find principles of numerical reckoning and practicable methods for measuring some quality connected with it." Note that he's not talking about quality in terms of value, but rather in terms of properties of things. (Oh, and by the way, Lord Kelvin also asserted that heavier-than-air flight was impossible, and that there was no practical purpose for radio.)

Kaner and Bond offer this definition of measurement: "Measurement is the empirical, objective assignment of numbers, according to a rule derived from a model or theory, to attributes of objects or events with the intent of describing them." I like this definition, to a point, but I'm still struggling with two problems. Kaner and Bond discuss and attack one of these problems quite thoroughly: the problem of construct validity. A *construct* is a class, category, or idealized (as opposed to concrete) object. Excellent measurement requires us to specify our constructs and the functions by which we apply numbers to them. (That's the difference between a measurement and a metric, by the way; a measurement is an observation linked to an evaluation. A metric is a function—a way of mapping a number to an observation. A construct is *valid* when the description and the classification map to reality in a reasonable way.

Problems ensue when constructs aren't valid. If the construct isn't valid, then neither the metric nor the measurement can be valid either. When that's a problem, some of your employees feel it—and the rest know it.

What Is Measurement?

Measurement is the art and science of making reliable observations.
—Jerry Weinberg

- Since the time of Aristotle (at least), we've known about two kinds of measurement that inform decisions
  - "Two pounds of meat"
  - "Too much", "too little", "just right".
- Measurement might be qualitative or quantitative, but assessment and evaluation are *always* qualitative:

What do we want?

We waste time and effort when we try to obtain six-decimal-place answers to whole-number questions.

See two articles on this: http://www.developsense.com/articles/2009-05-IssuesAboutMetricsAboutBugs.pdf, and http://www.developsense.com/articles/2009-07-ThreeKindsOfMeasurement.pdf. A related article, too: http://www.developsense.com/articles/2007-11-WhatCounts.pdf

How Else Do We Measure?

- First-order measurement
  - minimal fuss, direct observation, minimal instrumentation
  - used to inform a control action OR to prompt search for more refined information
  - "What's going on? What should we do? Where should we look?"

Weinberg suggests that, in software development, we're obsessed with trying to make third- and second-order measurements when first-order measurements might be all we need—and are cheaper in every way.

Why Prefer First-Order Measures?

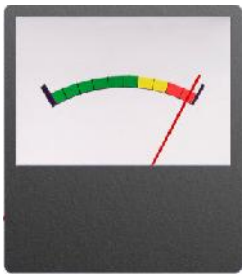
- When you're driving, are you mostly concerned about…
  - your velocity, acceleration, vehicle mass, drag co-efficient, frictional force? (third-order)
  - your engine temperature, RPMs, and current rate of gas consumption? (second-order)
  - looking out the window to avoid hitting something?

I've observed *many* projects that have crashed because managers were focused on the dashboard instead of the traffic and obstacles around them, and the road ahead.

What kind of driver do you trust?

Control vs. Inquiry Measurement

- A **control measurement** is a measurement that **drives decisions**.
  - *Any measurement you use to control a self-aware system will be used by that system to control YOU.*
- An **inquiry measurement** is any measurement that **helps you ask the right questions** at the right time.
  - Inquiry measurements are also vulnerable to gaming, but the stakes are far lower, so there's less incentive for manipulation.



This slide is taken from the work of my colleague, James Bach.
http://www.satisfice.com

A metric, by definition, is a simplification of reality, and as such, the same number can represent different realities.

Any system that involves humans is self-aware. *Any metric you use to control a self-aware system will be used by that system to control YOU.*

An inquiry metric might look like a control metric. The difference is how you use it and what you infer from it.

Observation can go directly to assessment and steering actions without quantified measurement. This is the first-order approach. Ask what other modes, beside numerical ones, you could use for observation and evaluation. Start by asking *what problem you want to solve* or *what situation you'd like to assess.* Prefer immediate observation to mediated observation. Make sure that you've considered a number of different possible interpretations, then choose a control action OR a search for more detail.

If you're worried that observations and assessments are subjective (they are), ask *several* people who matter

The Bright Future: Measurement for Inquiry, NOT Control

- Metrics like Pass/Fail Ratios and Defect Detection Percentage ignore almost every relevant factor
  - the fact that requirements and bugs are *relationships*
  - difficulty of the problems being solved
  - quality of the product design
  - quality of the code
  - release timing
  - who made the release decision, and why
  - timing of customer adoption
- …yet we use these bogus metrics to evaluate the quality of *testing*?

A line of code is a representation of an idea. A line of code can be as simple as placing a value in a CPU register or as complex as a multi-branch, multi-condition decision point. A developer's job is about learning, solving problems, and shaping and reshaping solutions. Sometimes that means removing lines of code rather than adding them. There's far more to a developer's job than counting the number of characters that she's typed. Lines of code are just scaled-up versions of the same silly measure.

What IS Exploratory Testing?

- I follow (and to some degree contributed to) Kaner's definition, which was refined over several peer conferences through 2007:

Exploratory software testing is…

- a style of software testing
- that emphasizes the personal freedom and responsibility
- of the individual tester
- to continually optimize the value of his or her work
- by treating test design, test execution, test result interpretation, and test-related learning
- as mutually supportive activities
- that run in parallel
- throughout the project.

Whoa. Maybe it would be a good idea to keep it brief *most* of the time…

See Kaner, "Exploratory Testing After 23 Years", www.kaner.com/pdfs/ETat23.pdf

Why Explore?

- You cannot use a script to
  - investigate a problem that you've found
  - decide that there's a problem with a script
  - escape the script problem you've identified
  - recognize terrible risks in the product
  - determine the best way to phrase a report
  - unravel a puzzling situation

Even "scripted" testers explore all the time!

•Some managers fear that E.T. depends on skill, but who benefits from *any* unskilled testing?

•Some managers fear that E.T. is unstructured, but it *is* structured

•Some managers fear that E.T. is unaccountable, but it can be *entirely* accountable

•Some managers fear that E.T. is unmanageable, but you can manage *anything* if you put your mind to it

Yes, Exploratory Testing Requires Skill

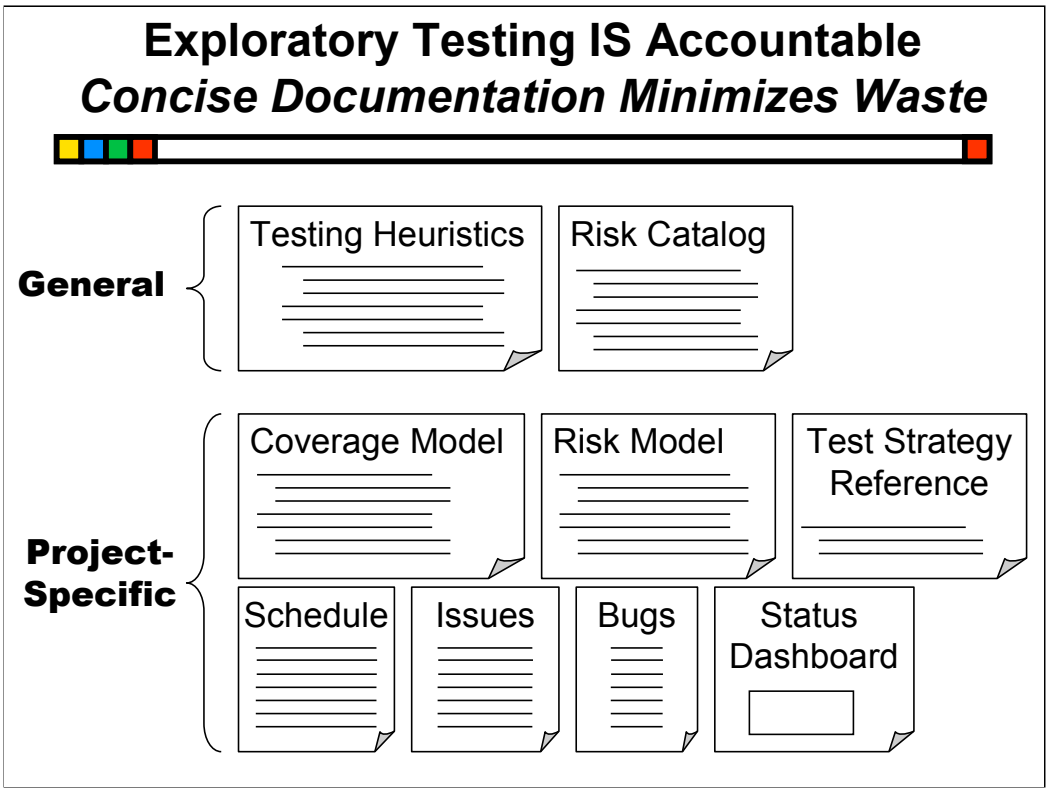
Exploratory Testing IS Structured

- We've studied the structures of ET, we've written about it, and we know how to teach it
- The structure of ET comes from *many* sources
  - Test design heuristics
  - Chartering
  - Time boxing
  - Perceived product risks
  - The nature of specific tests
  - The structure of the product being tested
  - The process of learning the product
  - Development activities
  - Constraints and resources afforded by the project
  - The skills, talents, and interests of the tester
  - The overall mission of testing

Not *procedurally* structured, but *cognitively* structured.

In other words, it's not "random", but systematic.

http://www.developsense.com/resources.html#exploratory

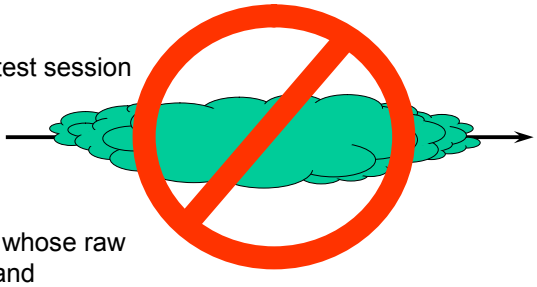


Detailed procedural documentation is expensive and largely unnecessary.

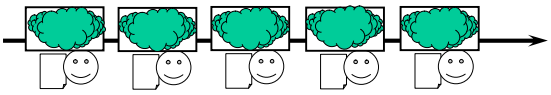
Tutorial documentation is also usually unnecessary, but if you do it, then keep it separate from the working documents.

Exploratory Testing IS Accountable Session-Based Test Management

- Charter
  - A clear, concise mission for a test session
- Time Box
  - 90-minutes (+/- 45)
- Reviewable Results
  - a session sheet—a test report whose raw data can be scanned, parsed and compiled by a tool
- Debriefing
  - a conversation between tester and manager or test lead



vs.

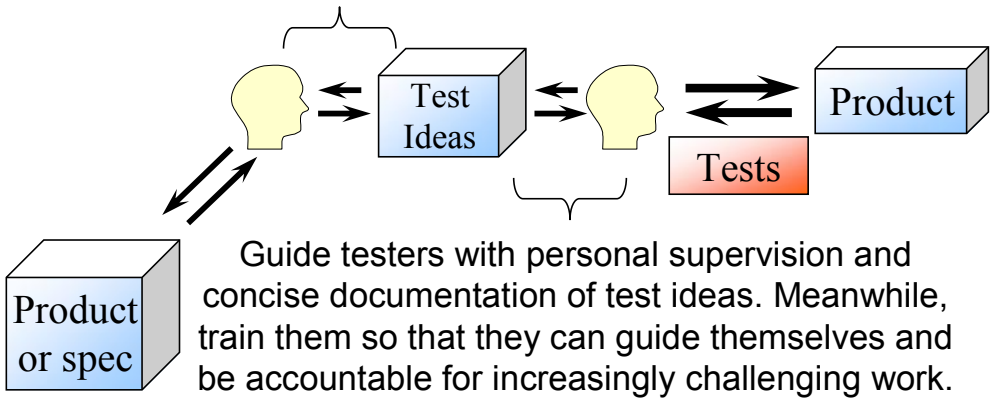


For more info, see http://www.satisfice.com/sbtm

42

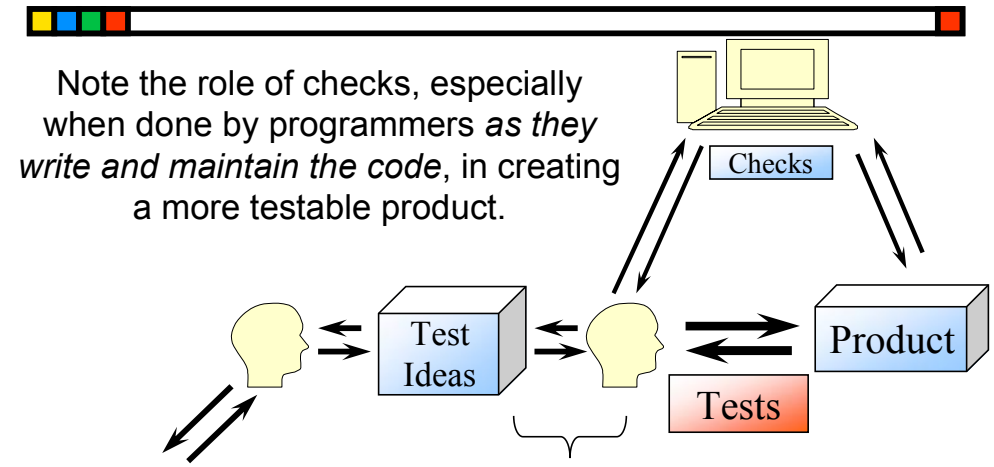
Exploratory Testing IS Manageable

Achieve excellent test design by exploring different test designs *while actually testing and interacting with the system*



Guide testers with personal supervision and concise documentation of test ideas. Meanwhile, train them so that they can guide themselves and be accountable for increasingly challenging work.

Exploratory Testing IS Manageable



Note the role of checks, especially when done by programmers *as they write and maintain the code*, in creating a more testable product.

Guide testers with personal supervision and concise documentation of test ideas. Meanwhile, train them so that they can guide themselves and be accountable for increasingly challenging work.

The future of testing is up to us.

These are not the only two futures.
They're offered for your consideration.
The choices are up to you.

Who I Am

Michael Bolton

(not the singer, not the guy in Office Space)

DevelopSense, Toronto, Canada

mb@developsense.com

+1 (416) 992-8378

http://www.developsense.com

Web Resources

- Michael Bolton
  http://www.developsense.com
- James Bach http://www.satisfice.com
- Cem Kaner http://www.kaner.com
- The Florida Institute of Technology
  - http://www.testingeducation.org
  - http://www.testingeducation.org/BBST/index.html
- StickyMinds http://www.StickyMinds.com
- Risks Digest http://catless.ncl.ac.uk/risks

Bibliography
How To Think About Testing

- *Perfect Software and Other Illusions About Testing*
  - Gerald M. Weinberg
- *Lessons Learned in Software Testing*
  - Cem Kaner, James Bach, and Bret Pettichord
- "Software Testing as a Social Science"
  - Cem Kaner; http://www.kaner.com/pdfs/KanerSocialScienceSTEP.pdf
- *Testing Computer Software*
  - Cem Kaner, Jack Falk, and Hung Quoc Nguyen
- *An Introduction to General Systems Thinking*
  - Gerald M. Weinberg
- *Exploring Requirements: Quality Before Design*
  - Gerald M. Weinberg

Bibliography
Recommended Test Technique Books

- *A Practitioner's Guide to Test Design*
  - Lee Copeland
- *How to Break Software*
  - James Whittaker
- *How to Break Software Security*
  - James Whittaker and Herbert Thompson
- *Lessons Learned in Software Testing*
  - Cem Kaner, James Bach, and Bret Pettichord
- *Testing Applications on the Web*
  - Hung Quoc Nguyuen
- *Hacking Web Applications Exposed*
  - Joel Scambray and Mike Shema

Bibliography
Jerry Weinberg

- *Quality Software Management Vol. 1: Systems Thinking*
- *Quality Software Management Vol. 2: First Order Measurement*
- *Secrets of Consulting: How to Give and Get Advice Successfully*
- **Anything** by Jerry Weinberg

Bibliography
Richard Feynman

- *The Pleasure of Finding Things Out*
  - see the Appendix to the Challenger Report.
- *Surely You're Joking, Dr. Feynman! Adventures of a Curious Character*
- *What Do You Care About What Other People Think?*

Bibliography
Other Areas

- *The Social Life of Information*
  - Paul Duguid and John Seely Brown
- *Please Understand Me*
  - David Kiersey
  - The Myers-Briggs Type Inventory, which provides insight into your own preferences and why *other people* seem to think so strangely
- *The Visual Display of Quantitative Information*
  - Edward Tufte
  - How to present information in persuasive, compelling, and beautiful ways
- *A Pattern Language*
  - Christopher Alexander et. al
  - A book about architecture
  - even more interesting as a book about thinking and creating similar but unique things—like computer programs and tests for them