Testing, Checking & Tools

FreeTest
Trondheim, Norway
March 2010

Michael Bolton
DevelopSense
http://www.developsense.com

Testing Isn't Just *Checking*

- Checking is a process of confirming and verifying existing beliefs
  - Checking can (and I argue, largely should) be done by automation
  - It is a *non-sapient* process

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

What *IS* Checking?

- A *check* has three attributes
  - It requires an *observation*
  - The observation is linked to a *decision rule*
  - The observation and the rule can be applied

without sapience

Oh no! What Does "*Sapient*" Mean?

- "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)

What Is *Sapience*?

- A ***sapient*** activity is one that requires a thinking human to perform
- We test not only for repeatability, but also for *adaptability*, *value*, and *threats to value*

This kind of testing CAN NOT be scripted

Checking IS Important…

- Despite what the Agilists might have you believe, checking is *not* new
  - D. McCracken (1957) refers to "program checkout"
  - Jerry Weinberg: checking was important in the early days because
    - computer time was expensive
    - programmers were cheap
    - the machinery was so unreliable
- Checking has been *rediscovered* by the Agilists
  - centrally important to test-driven development, refactoring, continuous integration & deployment
  - worthwhile checking must surrounded by good testing work
- CHECks are *CH*ange det*EC*tors

…But Checking Has Limitations

- Checks tend to be designed early…
- *…when we know less than we'll ever know* about the product and the project
- Checks focus on "pass vs. fail?"

But…

But…

- A good tester doesn't just ask

Pass or Fail?

- A good tester asks

Is there a problem here?

Machines can't…

recognize new risks, investigate, speculate, empathize, anticipate, predict, judge, suggest, project, recognize, refocus, appreciate, contextualize, elaborate, strategize, evaluate, become resigned, question, charter, assess, teach, learn, get frustrated, reframe, work around a problem, invent, make conscious decisions, model, refine, resource, troubleshoot, collaborate

Humans can…

recognize new risks, investigate, speculate, empathize, anticipate, predict, judge, suggest, project, recognize, refocus, appreciate, contextualize, elaborate, strategize, evaluate, become resigned, question, charter, assess, teach, learn, get frustrated, reframe, work around a problem, invent, make conscious decisions, model, refine, resource, troubleshoot, collaborate

FEEL THINK

Testing IS Exploring

- Our community sees testing as exploration, discovery, investigation, and learning
  - Testing can be *assisted* by machines, but can't be done by machines alone
  - Testing is a *sapient* process

I can't test, but I can help *you* act on test ideas.

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

What IS Exploratory Testing?

- Simultaneous test design, test execution, and learning.

—James Bach, 1995

But maybe it would be a good idea to underscore why that's important…

What IS Exploratory Testing?

- *Simultaneous test design, test execution, and learning, with an emphasis on learning.*

–Cem Kaner, 2005

But maybe it would be a good idea to be really explicit about what goes on…

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

Besides…

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

Even "scripted" testers explore all the time!

Checking Requires Testing

A check is skill-free, but it is dominated by testing skill.

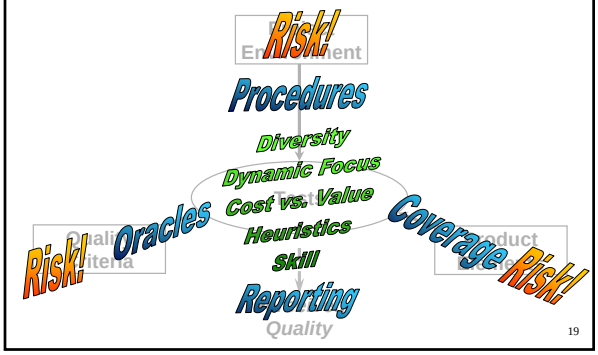
Before the Check

Recognize a risk	⇨ Testing skill
Translate to a test idea	⇨ Testing skill
Express a test idea as a bit	⇨ Testing skill
Turn the question into code	⇨ Programming skill
Determine the trigger	⇨ Testing skill
Encode the trigger	⇨ Programming skill

After The Check

Read the bit	⇨ Programming skill
Aggregate bits	⇨ Programming skill
Design a report	⇨ Testing, design skill
Encode the report	⇨ Programming skill
Observe the report	⇨ Testing skill
Determine meaning	⇨ Testing skill
Determine significance	⇨ Testing skill
Respond	⇨ Testing, programming, and management skill

A Heuristic Test Strategy Model



19

The Four-Part Risk Story

- Some *person*
- might suffer *harm or loss*
- because of a *vulnerability* in the product
- triggered by some *threat*.

Excellent checking reduces risks of unexpected implementation behaviour and of change.

Excellent risk-based testing is less about calculating and more about learning.

Oracles

An *oracle* is a heuristic principle or mechanism by which someone might recognize a problem.

(usually works, might fail)

(but not decide conclusively)

📖 Bug (n): Something that bugs someone who matters

Consistency ("this agrees with that")
an important theme in oracles

History
Image
Comparable Products
Claims
User Expectations
Purpose
Product
Standards

Consistency heuristics rely on the quality of your models of the product and its context.

Test Coverage Isn't Just Code Coverage

Test coverage is the amount of the system space that has been tested.

There are as many kinds of coverage as there are ways to model the product.

Product Elements

- Structure
- Function
- Data
- Platform
- Operations
- Time

Quality Criteria

Capability, Reliability, Usability, Security, Scalability, Performance, Installability, Compatibility, Supportability, Testability, Maintainability, Portability, Localizability

Test Techniques

- Function testing: *test what it does*
- Domain testing: *test what it does things to*
- Stress testing: *overwhelm or starve the product*
- Flow testing: *do one thing after another*
- Scenario testing: *test to a story*
- Claims testing: *test what people say*
- User testing: *involve the users*
- Risk testing: *anticipate a problem*
- Automatic testing: *run a zillion tests*

Cost as a Simplifying Factor
Try quick tests *as well as* careful tests

In my travels, I've seen extraordinary emphasis on long cycles of planning without feedback. This makes testing ineffective and slow.

A *quick test* is a cheap test that has some value, gives fast feedback, but requires little preparation, knowledge, or time to perform.

Bursts of quick tests represent a great way to *discover* risks upon which careful testing can be better focused.

Non-Rapid Test Automation

1. Obtain a sophisticated GUI test execution tool (extra points if you overpay for it).
2. Define a lot of low-value manual tests.
3. Hire an automation team to automate each one.
4. Build a comprehensive test library and framework.
5. Keep fixing it.

This can work if your product is *very easy to test* and *it doesn't change much*. Does that describe your product?

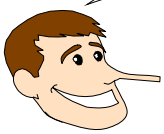
"Step right up!"

"Automated tests execute a sequence of actions without human intervention."

Actual text from a Microsoft whitepaper on automated testing

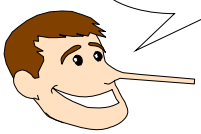
"Step right up!"

"This approach helps eliminate human error, and provides faster results."



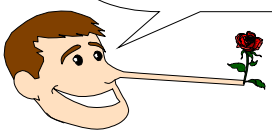
"Step right up!"

"Since most products require tests to be run many times, automated testing generally leads to significant labor cost savings over time."



"Step right up!"

"Typically a company will pass the break-even point for labor costs after just two or three runs of an automated test."



This argument boils down to:

Reckless Assumptions

1

Testing is a "sequence of actions"

Actually, testing is better viewed as an interactive cognitive process. Tools don't test; they may play a role in the test.

Reckless Assumptions

2

Testing means repeating the same actions over and over.

Testing has repetitive elements, but it isn't as repetitive as it seems, and variety is critical to the process. Use tools to enhance variety, rather than repetition.

Reckless Assumptions

3

We can automate testing actions.

Actually, in most cases we can only automate a subset of testing actions, for the rest, we need human testers. Testing dominates checking.

Reckless Assumptions

4

An automated test is faster, because it needs no human intervention.

Faster simulation of keystrokes & clicks is not the same as faster testing.

Human intervention is reduced during execution, but increased during diagnosis.

Reckless Assumptions

5

Automation reduces human error.

It reduces one kind of human error, but it multiplies other kinds of error. Automation allows us to do many kinds of ***bad*** *testing faster than ever.*

Reckless Assumptions

6

We can quantify the costs and benefits of manual vs. automated testing.

Manual testing and automated testing involve very different costs and benefits, many of which are hidden.

Reckless Assumptions

7

Automation will lead to "significant labor cost savings."

It can reduce some kinds of labor, while incurring other kinds. Avoid narrow analyses.

Reckless Assumptions

8

Automation will not harm the test project.

Test automation can distract testers, obscure the true test strategy, and provide a false sense of security.

Towards a better model of test automation…

- Test automation is…

Any use of tools to support testing.

Tool-Supported Exploration

- **Test generation (data and script generators).** Tools might create specialized data such as randomized email messages, or populate databases, or generate combinations of parameters that we'd like to cover with our tests.
- **System configuration.** Tools might preserve or reproduce system parameters, set systems to a particular state, or create or restore "ghosted" disk drives.
- **Simulators.** Tools might simulate sub-systems or environmental conditions that are not available (or not yet available) for testing, or are too expensive to provide live on demand.
- **Test execution (harnesses and test scripts).** Tools might operate the software itself, either simulating a user working through the GUI, or bypassing the GUI and using an alternative testable interface.

Tool-Supported Exploration

- **Probes.** Tools might make visible what would otherwise be invisible to humans. They might statically analyze a product, parse a log file, or monitor system parameters.
- **Oracles.** An oracle is any mechanism by which we detect failure or success. Tools might automatically detect certain kinds of error conditions in a product.
- **Activity recording & coverage analysis.** Tools might watch testing as it happens and retrospectively report what was and was not tested. They might record actions for later replay in other tests.
- **Visualization:** Tools can help us to display data sets, highlight key elements, map relationships, illustrate timing…
- **Test management.** Tools might record test results; organize test ideas or metrics.

Test tools are all over the place.

- **On your desktop** (never forget spreadsheets and text editors)
- **Web-based web testing resources** (HTML checkers, accessibility analyzers, Rubular, BrowserShots.org)
- **Scripting languages** (Perl, Ruby, Python, TCL) and associated libraries
- **Shareware repositories** (www.download.com)
- **O/S monitoring tools** (www.sysinternals.com)
- **Open source testware** (www.opensourcetesting.org, www.sourceforge.com)
- **Spyware for monitoring exploratory tests** (www.spectorsoft.com)
- **Any Microsoft development tool** (they always include useful utilities)
- **Microsoft compatibility toolkit** Windows Resource Kit and other free tools (www.microsoft.com)
- **The cubicle next door…** (someone else in your company has a tool for you)

Key Points:

- Testing is intellectual, not just clerical.
- Test automation is software development.
- Automation skill and automation projects aren't cheap.
- Tools can accelerate, extend, and enhance a good test process, but can slow down, limit, and degrade a poor one.
- Test automation is a promising idea that often falls far short of its promise.

These are warnings.
You can be very successful with test tools if you cope with these problems well.

Acknowledgements

- Much of this material is from *Rapid Software Testing* and *Rapid Software Testing for Managers*, by James Bach and Michael Bolton

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
  - http://www.developsense.com/blog/category/testing-vs-checking/
- 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
- *An Introduction to General Systems Thinking*
  - Gerald M. Weinberg
- *Exploring Requirements: Quality Before Design*
  - Gerald M. Weinberg

Bibliography

- *Testing Computer Software*
  - Cem Kaner, Jack Falk, and Hung Quoc Nguyen
- *A Practitioner's Guide to Test Design*
  - Lee Copeland
- *How to Break Software*
  - James Whittaker
- *Hacking Web Applications Exposed*
  - Joel Scambray and Mike Shema
- The Visual Display of Quantitative Information
  - Edward Tufte
  - How to present information in persuasive, compelling, and beautiful ways