

## Test Framing

Michael Bolton  
DevelopSense  
<http://www.developsense.com>  
TestNet  
May, 2011

## I'm Michael Bolton



Not the singer.



Not the guy  
in Office Space.



No relation.

## Michael Bolton



<http://www.developsense.com>  
michael@developsense.com

## Important Questions

### Why run that test?

- Variations:
  - Why are you planning to run that test?
  - Why are you running that test *right now*?
  - Why did you run that test?
  - Why perform that step?
  - Why make that observation?

## Important Questions

### Why NOT run THAT test?

- Variations:
  - Why aren't you planning to run that test?
  - Why aren't you running that test *right now*?
  - Why didn't you run that test?
  - Why not perform this step?
  - Why not make that observation?

## Important Questions

### Why didn't you find that bug?

- Variations:
  - Why didn't you find that bug earlier?
  - Why did you apparently ignore that requirement?
  - Why did you miss that symptom?
  - Why did you misinterpret that symptom?

## Important Questions

### Why do you think that's a bug?

- Variations:
  - Why do you say that this isn't working properly?
  - What requirement is being left unfulfilled here?
  - Why do you think that's a requirement?
  - For whom might this be a problem?
  - Do you think a user would ever do that?

## Even more generally...

### Why are you doing this?

- Variations:
  - Why are you not doing that?
  - How does this test relate to a requirement?
  - How does this test relate to a risk?
  - How does this test relate to your mission?

## To test is to compose, edit, narrate, and justify *three parallel stories*.

1. You must tell a story about the product...
  - ...about how it failed, and how it *might* fail...
  - ...in ways that matter to your various clients.
2. But also tell a story about how you tested it...
  - ...how you configured, operated and observed it...
  - ...about what you haven't tested yet...
  - ...or won't test at all...
3. And also tell a story that explains how good your testing was...
  - ...why your testing has been good enough...
  - ...why what you haven't done (so far) doesn't matter...
  - ...what the risks and costs of testing are...
  - ...how testable (or not) the product is...
  - ...what you need and what you recommend.

9

## What is test framing?

Test framing is *the chain of logical connections that structure and inform a test*, from the mission to the test result

Given this...



If these premises and facts are true...  
...and if we observe *this specific behaviour*...  
...and we apply *this relevant oracle*...  
...then we can make *these conclusions* about the...



## Vocabulary

- system
  - a set of things in meaningful relationship
- structure
  - that which forms the unchanging parts and relationships of a system; a consistent pattern for the system; "that which remains"
- narration
  - telling a story that fits in time
- framing
  - via logic and narrative, placing the test in logical relationship with the structures that inform it

## Vocabulary

- logic
  - a formal means of convincing or proving facts via valid arguments
  - ...using a set of propositions linked by connectives or operators
- proposition
  - a simple statement of fact or inference
- connectives
  - formal logic: if, and, or, and not, else, if and only if, therefore
  - less formally: because, unless, otherwise...

Framing provides traceability, but testers often limit traceability as being between tests and requirements *documents*—explicit requirements.

Can you demonstrate traceability between tests and *implicit* requirements?

## Much More Traceability

1. Product traces to specifications.
2. Specifications trace to standards.
3. Test sessions trace to product versions.
4. Test sessions trace to specifications.
5. Test sessions trace to logs which trace to product, playbook and specifications.
6. Test sessions trace to charters and charters to playbook.
7. Playbook traces to standards.
8. Playbook traces to specifications.
9. Playbook traces to risks which trace to specifications...
10. Tests trace to risk...
11. Tests trace to implicit requirements...
12. Tests trace to other tests...

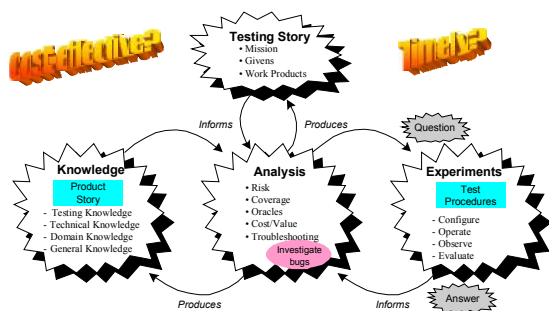
## How Do We Know What "Is"?

*We see the signs!*

"If I see X, then probably Y, because probably A, B, C, D, etc."

- THIS CAN FAIL:
  - Getting into a car—oops, not my car.
  - Bad driving—why?
  - Bad work—why?
  - Inexplicable behaviour—why?
  - I can never find the sugar—why?

## One Structure of Testing



15

## Project Environment Aspects of Our Context

*"CITESTD--Mother Approved!"*

- Customers
  - Anyone who is a client of the test project.
- Information
  - Information about the product or project that is needed for testing.
- Developer relations
  - How you get along with the programmers.
- Team
  - Anyone who will perform or support testing.
- Equipment & tools
  - Hardware, software, or documents required to administer testing.
- Schedule
  - The sequence, duration, and synchronization of project events.
- Test Items
  - The product to be tested.
- Deliverables
  - The observable products of the test project.

16

## Quality Criteria Identifying Value and Threats To It

*CRUSSPIC STMP*

- |                  |                   |
|------------------|-------------------|
| • Capability     | • Compatibility   |
| • Reliability    | • Supportability  |
| • Usability      | • Testability     |
| • Security       | • Maintainability |
| • Scalability    | • Portability     |
| • Performance    | • Localizability  |
| • Installability |                   |

Many test approaches focus on Capability (functionality) and underemphasize the other criteria 17

## Product Elements Ways to Model and Cover The Product

*"SFDPOPT-San Francisco DePOT"*

- Structure
  - What are the pieces and how do they fit together?
- Function
  - What does the product do?
- Data
  - What does the product do things to?
- Platform
  - What does the product depend upon?
- Operations
  - How do people actually use the program?
- Time
  - How does the product interact with time?

18

### Test Techniques General Ways to Test

**FDSFSCURA**

- Function testing
  - *Test what it does*
- Domain testing
  - *Divide and conquer the data*
- Stress testing
  - *Overwhelm or starve the product*
- Flow testing
  - *Do one thing after another after another*
- Scenario testing
  - *Test to a compelling story*

19

### Test Techniques General Ways to Test

**FDSFSCURA**

- Claims testing
  - *Test everything that people say it should do*
- User testing
  - *Involve the users (or systematically simulate them)*
- Risk testing
  - *Imagine a problem, and then look for it*
- Automatic testing
  - *Perform zillions of tests, aided by machines*

20

### Thirty-Six Test Strategy Heuristics

**"cidtestdsfdpotcrusspicstmplfdsfscura"**

Customers	Structures	Capability	Function testing
Information	Functions	Reliability	Domain testing
Developer relations	Data	Usability	Stress testing
Team	Platforms	Security	Flow testing
Equipment & tools	Operations	Scalability	Scenario testing
Schedule	Time	Performance	Claims testing
Test Items	<b>Product Elements</b>	Instability	User testing
Deliverables		Compatibility	Risk testing
		Supportability	Automatic testing
<b>Project Environment</b>		Testability	
		Maintainability	<b>Test Techniques</b>
		Portability	
		Localizability	
		<b>Quality Criteria</b>	

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

- **History:** The present version of the system *is consistent* with past versions of it.
- **Image:** The system *is consistent* with an image that the organization wants to project.
- **Comparable Products:** The system *is consistent* with comparable systems.
- **Claims:** The system *is consistent* with what important people say it's supposed to be.
- **Users' Expectations:** The system *is consistent* with what users want.
- **Product:** Each element of the system is *consistent* with comparable elements in the same system.
- **Purpose:** The system *is consistent* with its purposes, both explicit and implicit.
- **Statutes:** The system *is consistent* with applicable laws.
- **Familiarity:** The system *is not consistent* with the pattern of any familiar problem.

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

22

### What if you have an unframed test?

**Try framing it!**

But if you can't do it perfectly, or right away, that might be okay. Why?

To test a *very simple* product meticulously, part of a complex product meticulously, or to maximize test *integrity*...

**FOCUS!**

1. Start the test from a *known* (clean) state.
2. Prefer *simple, deterministic* actions.
3. Trace test steps to a *specified model*.
4. Follow *established and consistent* lab procedures.
5. Make *specific* predictions, observations and records.
6. Make it *easy to reproduce* (automation helps).

## How can you justify an unframed test?

All of the  
hidden frames!

Focus in testing is valuable, but we must also practice de-focusing to expose new problems and to trigger new ideas about risk.

To find *unexpected problems*,  
*elusive problems* that may occur in the field,  
or more problems *quickly* in a complex product...

De-focus!

1. Start from *different states* (not necessarily clean).
2. Prefer *complex, challenging* actions.
3. Generate tests from a *variety* of models.
4. *Question* your lab procedures and tools.
5. Try to *see everything* with open expectations.
6. Make the test *hard to pass*, instead of easy to reproduce.

## Galumphing

*A Defocusing Heuristic to Exploit Variability*

- doing things in a deliberately over-elaborate way
- adding lots of unnecessary but inert actions that are inexpensive and shouldn't (in theory) affect the test outcome
  - bring up a dialog and dismiss it
  - modify an option and rescind it
  - perform an action and reverse it
  - re-selecting default options
  - inserting an expression where a single value would do
  - over-filling an input field, then fixing it

## Exploiting Variation To Find More Bugs

- **Micro-behaviors**
  - Unreliable and distractible humans make each test a little bit new each time through.
- **Randomness**
  - Can protect you from unconscious bias (but be careful; humans almost always act non-randomly)
- **Data Substitution**
  - The same actions may have dramatically different results when tried on a different database, or with different input.
- **Timing/Concurrency Variations**
  - The same actions may have different results depending on the time frame in which they occur and other concurrent events.
- **Platform Substitution**
  - Supposedly equivalent platforms may not be.

## Exploiting Variation To Find More Bugs

- **Scenario Variation**
  - The same functions may operate differently when employed in a different flow or context.
- **State Pollution**
  - Hidden variables of all kinds frequently exert influence in a complex system. By varying the order, magnitude, and types of actions, we may accelerate state pollution, and discover otherwise rare bugs.
- **Sensitivities and Expectations**
  - Different testers may be sensitive to different factors, or make different observations. The same tester may see different things at different times or when intentionally shifting focus to different things.