

# ***Black Box Software Testing***

*Spring 2005*

PART 8 -- TEST DESIGN

by

Cem Kaner, J.D., Ph.D.

Professor of Software Engineering

Florida Institute of Technology

and

James Bach

Principal, Satisfice Inc.

**Copyright (c) Cem Kaner & James Bach, 2000-2004**

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

These notes are partially based on research that was supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# *Test design: Let's take stock*

We've studied four techniques so far

- Function testing
- Domain testing
- A first look at risk-based testing
- Scenario testing

Many more techniques, perhaps 150, have been published.

# *Test design: Today's objectives*

In this lecture, we ask:

- What is a test technique?
- How does knowing test techniques help us create tests?
- How should we decide to use one technique instead of another?
- Can we simplify the diverse space of techniques into some underlying principles?

# *What's a test technique?*

## *Ten dominating techniques*

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

These are  
10 common  
Examples.

There are *many*  
Others.

# *Ten dominating techniques*

- **Function testing**
- **Specification-based testing**
- **Domain testing**
- **Risk-based testing**
- **Scenario testing**
- **Regression testing**
- **Stress testing**
- **User testing**
- **State-model based testing**
- **High volume automated testing**

**Test each feature or function on its own.**

**Scan through the product, covering every feature or function with at least enough testing to determine what it does and whether it is working.**

# *Ten dominating techniques*

- **Function testing**
- **Specification-based testing**
- **Domain testing**
- **Risk-based testing**
- **Scenario testing**
- **Regression testing**
- **Stress testing**
- **User testing**
- **State-model based testing**
- **High volume automated testing**

**Check every claim made in the reference document (such as, a contract specification).**

**Test to the extent that you have proved the claim true or false.**

# *Ten dominating techniques*

- **Function testing**
- **Specification-based testing**
- **Domain testing**
- **Risk-based testing**
- **Scenario testing**
- **Regression testing**
- **Stress testing**
- **User testing**
- **State-model based testing**
- **High volume automated testing**

**Focus on variables, such as inputs, outputs, configuration, or internal (e.g. file-handling) variables.**

**For every variable or combination of variables, consider the space of possible values. Simplify it by partitioning into subsets. Pick a few representatives of each subset.**

# *Ten dominating techniques*

- Function testing
- Specification-based testing
- Domain testing
- **Risk-based testing**
- Scenario testing
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

**A program is a collection of opportunities for things to go wrong.**

**For each way that you can imagine the program failing, design tests to determine whether the program actually will fail in that way.**

# *Ten dominating techniques*

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- **Scenario testing**
- Regression testing
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

**Tests are complex stories that capture how the program will be used in real-life situations.**

**These are combination tests, whose combinations are credible reflections of real use.**

# *Ten dominating techniques*

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- **Regression testing**
- Stress testing
- User testing
- State-model based testing
- High volume automated testing

**Repeat the same test after some change to the program.**

**You can use any test as a regression test, but if you do a lot of regression testing, you will (or should) learn to design cases for efficient reuse.**

# *Ten dominating techniques*

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- **Stress testing**
- User testing
- State-model based testing
- High volume automated testing

Many definitions of stress testing.

When I say stress testing, I mean tests intended to overwhelm the product, to subject it to so much input, so little memory, such odd combinations that I expect it to fail and am exploring its behavior as (and after) it fails.

# *Ten dominating techniques*

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- **User testing**
- State-model based testing
- High volume automated testing

Give the program to “a user,” see what he does with it and how it responds.

User tests can be tightly structured or very loosely defined. The essence is room for action and response by “users”.

# *Ten dominating techniques*

- Function testing
- Specification-based testing
- Domain testing
- Risk-based testing
- Scenario testing
- Regression testing
- Stress testing
- User testing
- **State-model based testing**
- High volume automated testing

**Model the program as a state machine that runs from state to state in response to events (such as new inputs).**

**In each state, does it respond correctly to each event?**

# *Ten dominating techniques*

- **Function testing**
- **Specification-based testing**
- **Domain testing**
- **Risk-based testing**
- **Scenario testing**
- **Regression testing**
- **Stress testing**
- **User testing**
- **State-model based testing**
- **High volume automated testing**

**Program the computer to design, implement, execute and interpret a large series of tests.**

**You set the wheel in motion, supply the oracle(s) and evaluate the pattern of results.**

# *Test design: Today's objectives*

In this lecture, we ask:

- What is a test technique?
- **How does knowing test techniques help us create tests?**
- How should we decide to use one technique instead of another?
- Can we simplify the diverse space of techniques into some underlying principles?

# *Designing test cases*

- Focus on procedure?
  - “A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.” (IEEE)
- Focus on the test idea?
  - “A test idea is a brief statement of something that should be tested. For example, if you're testing a square root function, one idea for a test would be ‘test a number less than zero’. The idea is to check if the code handles an error case.” (Marick)

## *Test cases*

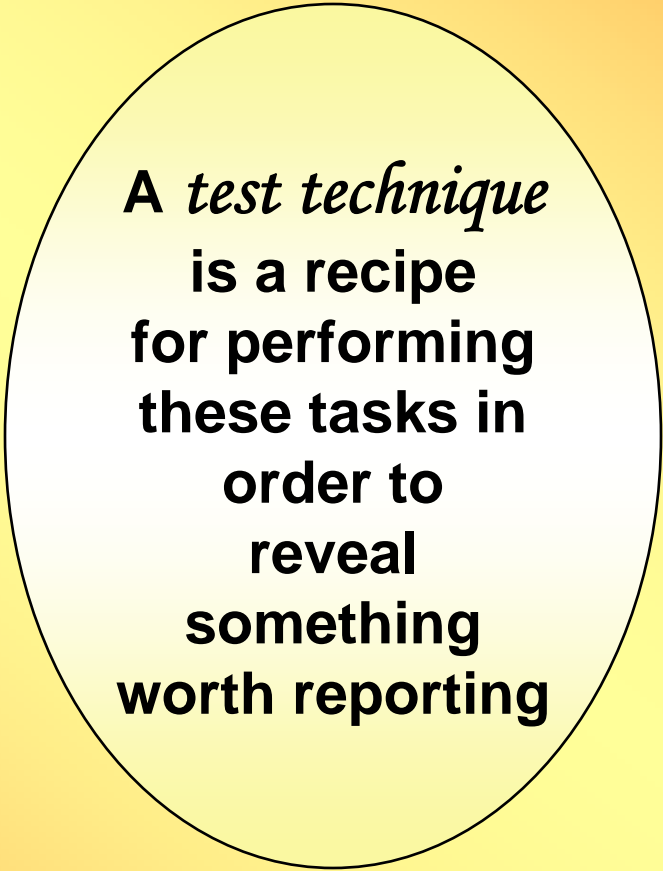
- In my view,

A test case is a question  
you ask the program.

- The point of running the test is to gain information, for example whether the program will pass or fail the test.
  - The test must be **CAPABLE** of revealing valuable information
  - The **SCOPE** of a test changes over time, because the information value of tests changes as the program matures
  - The **METRICS** that count test cases are essentially meaningless because test cases merge or are abandoned as their information value diminishes.

# *How do we use test techniques to create tests?*

- Analyze the situation.
- Model the test space.
- Select what to cover.
- Determine test oracles.
- Configure the test system.
- Operate the test system.
- Observe the test system.
- Evaluate the test results.



*A test technique*  
**is a recipe  
for performing  
these tasks in  
order to  
reveal  
something  
worth reporting**

# *Test design: Today's objectives*

In this lecture, we ask:

- What is a test technique?
- How does knowing test techniques help us create tests?
- **How should we decide to use one technique instead of another?**
- Can we simplify the diverse space of techniques into some underlying principles?

# *How to choose a test technique?*

- This is a multi-dimensional challenge:
  - *Your information objectives*
  - Attributes of the potential tests
  - The development context:
    - Product elements
    - Quality criteria
    - Risks
    - Project factors (constraints and opportunities)

# *Information Objectives: Which Group is Better?*

## *Testing Group 1*

	Found pre-release
Function A	100
Function B	0
Function C	0
Function D	0
Function E	0
Total	100

## *Testing Group 2*

Function A	50
Function B	6
Function C	6
Function D	6
Function E	6
Total	74

**From Marick,  
*Classic Testing  
Mistakes***

Two groups test the same program.

- The functions are equally important
- The bugs are equally significant

*This is artificial, but it sets up a simple context for a discussion of tradeoffs.*

## *Which group is better?*

	Found pre-release	Found later	Total
Function A	100	0	100
Function B	0	12	12
Function C	0	12	12
Function D	0	12	12
Function E	0	12	12
Total	100	48	148
Function A	50	50	100
Function B	6	6	12
Function C	6	6	12
Function D	6	6	12
Function E	6	6	12
Total	74	74	148

# *Information objectives*

- Find important bugs, to get them fixed
- Help managers make ship / no-ship decisions
- Check interoperability with other products
- Block premature product releases
- Minimize technical support costs
- Assess conformance to specification
- Conform to regulations
- Minimize safety-related lawsuit risk
- Find safe scenarios for use of the product
- Assess quality

# *How to choose a test technique?*

- This is a multi-dimensional challenge:
  - Your information objectives
  - *Attributes of the potential tests*
  - The development context:
    - Product elements
    - Quality criteria
    - Risks
    - Project factors (constraints and opportunities)

# *Test attributes*

To different degrees, good tests have these attributes:

- **Power.** When a problem exists, the test will reveal it.
- **Valid.** When the test reveals a problem, it is a genuine problem.
- **Value.** It reveals things your clients want to know about the product or project.
- **Credible.** Your client will believe that people will do the things that are done in this test.
- **Representative** of events most likely to be encountered by the user. (xref. Musa's *Software Reliability Engineering*).
- **Non-redundant.** This test represents a larger group that address the same risk.
- **Motivating.** Your client will want to fix the problem exposed by this test.
- **Performable.** It can be performed as designed.
- **Maintainable.** Easy to revise in the face of product changes.
- **Repeatable.** It is easy and inexpensive to reuse the test.
- **Pop.** (*short for Karl Popper*) It reveal things about our basic or critical assumptions.
- **Coverage.** It exercises the product in a way that isn't already taken care of by other tests.
- **Easy to evaluate.**
- **Supports troubleshooting.** Provides useful information for the debugging programmer.
- **Appropriately complex.** As the program gets more stable, you can hit it with more complex tests and more closely simulate use by experienced users.
- **Accountable.** You can explain, justify, and prove you ran it.
- **Cost.** This includes time and effort, as well as direct costs.
- **Opportunity Cost.** Developing and performing this test prevents you from doing other work

# *Contrasting the techniques*

- Domain testing
  - Focused on non-redundancy, validity, power, and variables-coverage. Tests are typically highly repeatable, simple, and *should be* easy to maintain.
  - *Not* focused on representativeness, credibility, or motivational effect.
- Scenario testing
  - Focused on validity, complexity, credibility, and motivational effect.
  - *Not* focused on power, maintainability, or coverage.

# *How to choose a test technique?*

- This is a multi-dimensional challenge:
  - Your information objectives
  - Attributes of the potential tests
  - ***The development context:***
    - ***Product elements***
    - ***Quality criteria***
    - ***Risks***
    - ***Project factors (constraints and opportunities)***

# *Test design: Today's objectives*

In this lecture, we ask:

- What is a test technique?
- How does knowing test techniques help us create tests?
- How should we decide to use one technique instead of another?
- **Can we simplify the diverse space of techniques into some underlying principles?**

# *Test design*



*A simplifying model for  
classifying and  
generating test  
techniques*

# *Test design*

Testing combines techniques that focus on:

- **Testers**: *who* does the testing.
- **Coverage**: *what* gets tested.
- **Potential problems**: *why* you're testing (what risk you're testing for).
- **Activities**: *how* you test.
- **Evaluation**: *how to tell whether the test passed or failed.*
- **Artefact**: *What you will report*

*All testing involves  
all five dimensions.*

# *Test design*

- What is the difference between
  - User testing?
  - Usability testing?
  - User interface testing?

## *Closing thoughts*

- Good test design involves developing tests that
  - Can help you satisfy your overall information objectives for this project (or this part of it)
  - Address the things that you want to test in ways that can reveal the information that you want to find out about them
  - Are realistically achievable within your constraints
  - Include the support materials (documentation, code, etc.) that you will need for the level of reuse that you consider appropriate
  - Are optimized for the qualities (e.g. power, credibility) most important for your present purposes.
- No one technique will fill all of your needs. Use many techniques, designing each test in a way that makes a given design problem seem easy and straightforward.

# *Test Design: Some Readings*

- Kaner, Bach & Pettichord, “Testing Techniques” in *Lessons Learned in Software Testing*.
- Kaner, C. (2003) “What is a good test case?” <http://www.testingeducation.org/a/testcase.pdf>
- Whittaker, “What is testing? And why is it so hard?”  
<http://www.computer.org/software/so2000/pdf/s1070.pdf>
- Whittaker & Atkin, *Software Engineering is not Enough*,  
<http://www.sisecure.com/pdf/jwsasofteng.pdf>

