

# *Black Box Software Testing*

## *(Professional Seminar)*

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

Professor of Computer Sciences  
Florida Institute of Technology

### **Section:29**

### **Automation Architectures**

Summer, 2002

Contact Information:

kaner@kaner.com

[www.kaner.com](http://www.kaner.com) (testing website)

[www.badsoftware.com](http://www.badsoftware.com) (legal website)

I grant permission to make digital or hard copies of this work for personal or classroom use, with or without fee, provided that (a) copies are not made or distributed for profit or commercial advantage, (b) copies bear this notice and full citation on the first page, and if you distribute the work in portions, the notice and citation must appear on the first page of each portion, (c) each page bear the notice "Copyright (c) Cem Kaner" or if you changed the page, "Adapted from Notes Provided by Cem Kaner". Abstracting with credit is permitted. The proper citation for this work is Cem Kaner, *A Course in Black Box Software Testing (Professional Version)*, Summer 2002, [www.testing-education.org](http://www.testing-education.org). To copy otherwise, to republish or post on servers, or to distribute to lists requires prior specific permission and a fee. Request permission to republish from [kaner@kaner.com](mailto:kaner@kaner.com).

# *Black Box Software Testing*

## **Alternatives to GUI-Based Automated Regression Testing**

Several of these slides were developed by Doug Hoffman or in co-authorship with Doug Hoffman for a course that we co-taught on software test automation.

Many of the ideas in this presentation were presented and refined in Los Altos Workshops on Software Testing.

LAWST 5 focused on oracles. Participants were Chris Agruss, James Bach, Jack Falk, David Gelperin, Elisabeth Hendrickson, Doug Hoffman, Bob Johnson, Cem Kaner, Brian Lawrence, Noel Nyman, Jeff Payne, Johanna Rothman, Melora Svoboda, Loretta Suzuki, and Ned Young.

LAWST 1-3 focused on several aspects of automated testing. Participants were Chris Agruss, Tom Arnold, Richard Bender, James Bach, Jim Brooks, Karla Fisher, Chip Groder, Elisabeth Hendrickson, Doug Hoffman, Keith W. Hooper, III, Bob Johnson, Cem Kaner, Brian Lawrence, Tom Lindemuth, Brian Marick, Thanga Meenakshi, Noel Nyman, Jeffery E. Payne, Bret Pettichord, Drew Pritsker, Johanna Rothman, Jane Stepak, Melora Svoboda, Jeremy White, and Rodney Wilson.

I'm indebted to James Whittaker, James Tierney, Harry Robinson, and Noel Nyman for additional explanations of stochastic testing.

# *What is Automation Design?*

- **Determine the goals of the automation**
- **Determine the capabilities needed to achieve those goals**
- **Select automation components**
- **Set relationships between components**
- **Identify locations of components and events**
- **Sequence test events**
- **Evaluate and report results of test events.**

# *Issues Faced in A Typical Automated Test*

- What is being tested?
- How is the test set up?
- Where are the inputs coming from?
- What is being checked?
- Where are the expected results?
- How do you know pass or fail?

# *Automated Software Test Functions*

- Automated test case/data generation
- Test case design from requirements or code
- Selection of test cases
- Able to run two or more specified test cases
- Able to run a subset of all the automated test cases
- No intervention needed after launching tests
- Automatically sets-up and/or records relevant test environment
- Runs test cases
- Captures relevant results
- Compares actual with expected results
- Reports analysis of pass/fail

# *Characteristics of “fully automated” tests*

- **A set of tests is defined and will be run together.**
- **No intervention needed after launching tests.**
- **Automatically sets-up and/or records relevant test environment.**
- **Obtains input from existing data files, random generation, or another defined source.**
- **Runs test exercise.**
- **Captures relevant results.**
- **Evaluates actual against expected results.**
- **Reports analysis of pass/fail.**

*Not all automation is full automation. Partial automation can be very useful.*

# *Capabilities of Automation Tools*

**Automated test tools combine a variety of capabilities. For example, GUI regression tools provide:**

- capture/replay for easy manual creation of tests
- execution of test scripts
- recording of test events
- compare the test results with expected results
- report test results

**Some GUI tools provide additional capabilities, but no tool does everything well.**

# *Capabilities of Automation Tools*

## **Here are examples of automated test tool capabilities:**

- Analyze source code for bugs
- Design test cases
- Create test cases (from requirements or code)
- Generate test data
- Ease manual creation of test cases
- Ease creation/management of traceability matrix
- Manage testware environment
- Select tests to be run
- Execute test scripts
- Record test events
- Measure software responses to tests (Discovery Functions)
- Determine expected results of tests (Reference Functions)
- Evaluate test results (Evaluation Functions)
- Report and analyze results

# *Tools for Improving Testability by Providing Diagnostic Support*

- **Hardware integrity tests.** Example: power supply deterioration can look like irreproducible, buggy behavior.
- **Database integrity.** Ongoing tests for database corruption, making corruption quickly visible to the tester.
- **Code integrity.** Quick check (such as checksum) to see whether part of the code was overwritten in memory.
- **Memory integrity.** Check for wild pointers, other corruption.
- **Resource usage reports:** Check for memory leaks, stack leaks, etc.
- **Event logs.** See reports of suspicious behavior. Probably requires collaboration with programmers.
- **Wrappers.** Layer of indirection surrounding a called function or object. The automator can detect and modify incoming and outgoing messages, forcing or detecting states and data values of interest.

# *GUI Regression is Just a Special Case*

## Source of test cases

- **Old**

## Size of test pool

- **Small**

## Serial dependence among tests

- **Independent**

## Evaluation strategy

- **Comparison to saved result**

# *GUI Regression is Just a Special Case*

## **Source of test cases**

- *Old*
- **Intentionally new**
- **Random new**

## **Size of test pool**

- *Small*
- **Large**
- **Exhaustive**

## **Serial dependence among tests**

- *Independent*
- **Sequence is relevant**

# *GUI Regression is Just a Special Case*

## **Evaluation strategy**

- *Comparison to saved result*
- **Comparison to an oracle**
- **Comparison to a computational or logical model**
- **Comparison to a heuristic prediction. (NOTE: All oracles are heuristic.)**
- **Crash**
- **Diagnostic**
- **State model**

# *A Different Special Case: Exhaustive Testing*

## **MASPAR functions: square root tests**

- **32-bit arithmetic, built-in square root**
  - » **2<sup>32</sup> tests (4,294,967,296)**
  - » **6 minutes to run the tests**
  - » **Much longer to run the oracle**
  - » **Discovered 2 errors that were not associated with any boundary (a bit was mis-set, and in two cases, this affected the final result).**
- **64-bit arithmetic?**

# *A Different Special Case: Exhaustive Testing*

## **MASPAR functions: square root tests**

- **Source of test cases**
  - » Intentional new
- **Size of test pool**
  - » Exhaustive
- **Evaluation strategy**
  - » Comparison to an oracle
- **Serial dependence among tests**
  - » Independent

# *Random Testing: Independent and Stochastic Approaches*

## **Random Testing**

- Random (or statistical or stochastic) testing involves generating test cases using a random number generator. Because they are random, the individual test cases are not optimized against any particular risk. The power of the method comes from running large samples of test cases.

## **Stochastic Testing**

- Stochastic process involves a series of random events over time
  - » Stock market is an example
  - » Program typically passes the individual tests: The goal is to see whether it can pass a large series of the individual tests.

## **Independent Testing**

- Our interest is in each test individually, the test before and the test after don't matter.

# *Independent Random Tests: Function Equivalence Testing*

## Hypothetical case: Arithmetic in Excel

- **Suppose we had a pool of functions that worked well in previous version.**
  - » **For individual functions, generate random number and take function (e.g. log) in Excel 97 and Excel 2000.**
    - *Spot check results (e.g. 10 cases across the series)*
  - » **Build a model to combine functions into expressions**
    - *Generate and compare expressions*
    - *Spot check results*

# *Independent Random Tests: Function Equivalence Testing*

## **Hypothetical case: Arithmetic in Excel**

- **Source of test cases**
  - » Random new
- **Size of test pool**
  - » Large
- **Evaluation strategy**
  - » Comparison to an oracle
- **Serial dependence among tests**
  - » Independent

# *Comparison Functions*

## **Parallel function (Oracle)**

- Previous version
- Competitor
- Standard function
- Custom model

## **Computational or logical model**

- Inverse function
  - » mathematical inverse
  - » operational inverse (e.g. split a merged table)
- Useful mathematical rules (e.g.  $\sin^2(x) + \cos^2(x) = 1$ )

# *Oracles: Challenges*

- **Incomplete information from oracle**
  - May be more than one oracle for SUT
  - Inputs may effect more than one oracle
- **Accuracy of information from oracle**
  - Close correspondence makes common mode faults likely
  - Independence is necessary:
    - » algorithms
    - » sub-programs and libraries
    - » system platform
    - » operating environment

# *Oracles: Challenges*

---

- **Close correspondence reduces maintainability**
- **Must maintain currency of oracle through changes in the SUT**
- **Oracle may become as complex as SUT**
- **More complex oracles make more errors**
- **Speed of predictions**
- **Usability of results**

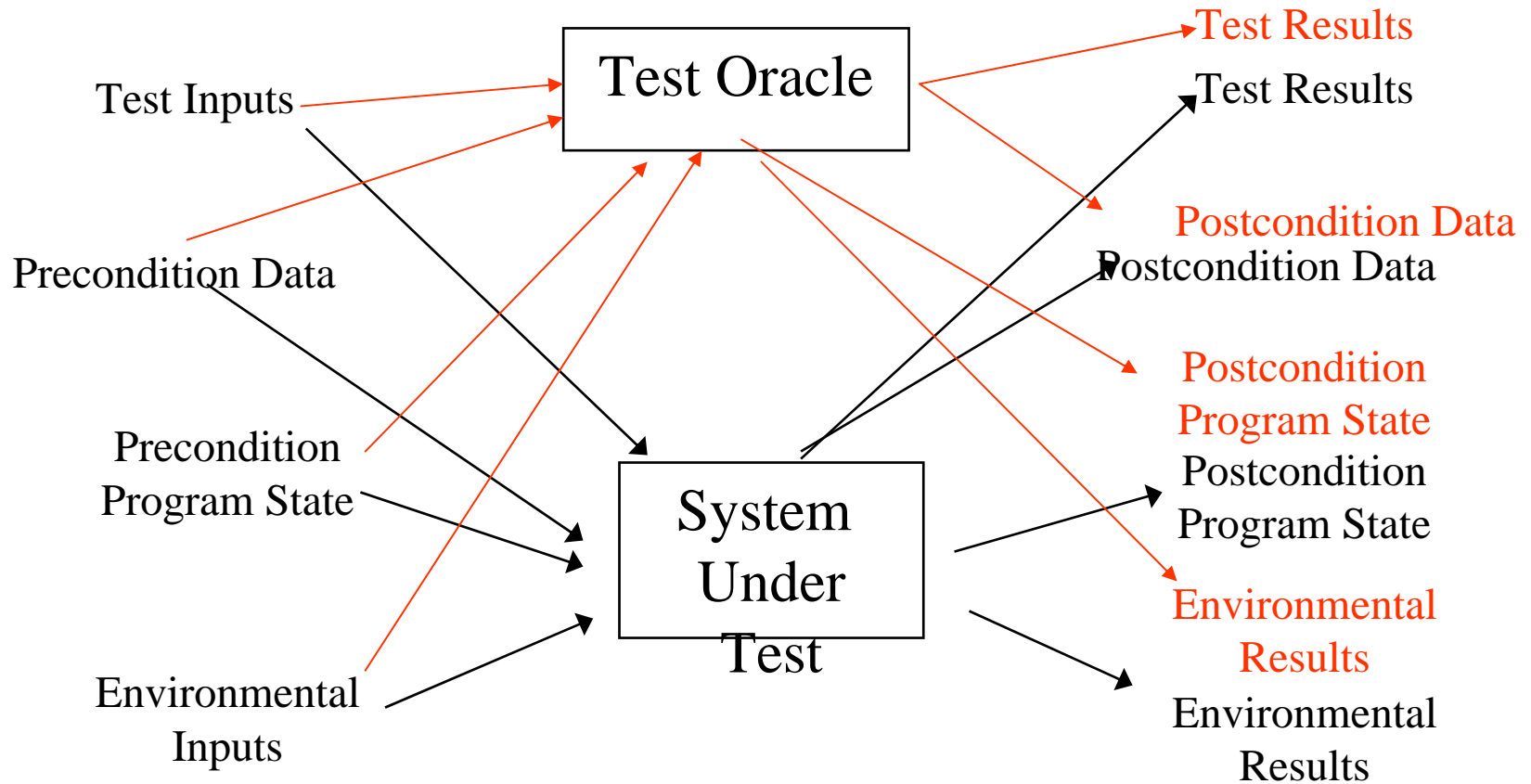
# *Heuristic Oracles*

- Heuristics are rules of thumb that support but do not mandate a given conclusion. We have partial information that will support a probabilistic evaluation. This won't tell you that the program works correctly but it can tell you that the program is broken. This can be a cheap way to spot errors early in testing.

## **Example:**

- History of transactions → Almost all transactions came from New York last year.
- Today, 90% of transactions are from Wyoming. Why? Probably (but not necessarily) the system is running amok.

# The “Complete” Oracle



Reprinted with permission of Doug Hoffman

# *Stochastic Test: Dumb Monkeys*

## **Dumb Monkey**

- **Random sequence of events**
- **Continue through crash (Executive Monkey)**
- **Continue until crash *or* a diagnostic event occurs. The diagnostic is based on knowledge of the system, not on internals of the code. (Example: button push doesn't push—this is system-level, not application level.)**

# *Stochastic Test: Dumb Monkeys*

## **Dumb Monkey**

- **Source of test cases**
  - » Random new
- **Size of test pool**
  - » Large
- **Evaluation strategy**
  - » Crash or Diagnostics
- **Serial dependence among tests**
  - » Sequence is relevant

# *Stochastic Test Using Diagnostics*

## **Telephone Sequential Dependency**

- **Symptoms were random, seemingly irreproducible crashes at a beta site**
- **All of the individual functions worked**
- **We had tested all lines and branches.**
- **Testing was done using a simulator, that created long chains of random events. The diagnostics in this case were assert fails that printed out on log files.**

# *Stochastic Test Using Diagnostics*

## **Telephone Sequential Dependency**

- **Source of test cases**
  - » Random new
- **Size of test pool**
  - » Large
- **Evaluation strategy**
  - » Diagnostics
- **Serial dependence among tests**
  - » Sequence is relevant

# *Stochastic Test: Model Based*

## **Testing Based on a State Model**

- **For any state, you can list the actions the user can take, and the results of each action (what new state, and what can indicate that we transitioned to the correct new state).**
- **Randomly run the tests and check expected against actual transitions.**
- See **[www.geocities.com/model based testing/online papers.htm](http://www.geocities.com/model_based_testing/online_papers.htm)**

# *Stochastic Test: Model Based*

## **Testing Based on a State Model**

- **Source of test cases**
  - » Random new
- **Size of test pool**
  - » Large, medium or small (different substrategies)
- **Evaluation strategy**
  - » State model or crash
- **Serial dependence among tests**
  - » Sequence is relevant

# *Stochastic Test: Save Tests Based*

## **Testing with Sequence of Passed Tests**

- **Collect a large set of regression tests, edit them so that they don't reset system state.**
- **Randomly run the tests in a long series and check expected against actual results.**
- **Will sometimes see failures even though all of the tests are passed individually.**

# *Stochastic Test: Save Tests Based*

## **Testing with Sequence of Passed Tests**

- **Source of test cases**
  - » Old
- **Size of test pool**
  - » Large
- **Evaluation strategy**
  - » Saved results or Crash or Diagnostics
- **Serial dependence among tests**
  - » Sequence is relevant

