Architectures of Test Automation: Alternatives to GUI Regression Testing

Cem Kaner, BA, JD, PhD, ASQ-CQE

Professor of Software Engineering
Department of Computer Science
Florida Institute of Technology

October, 2000

Notice and Acknowledgement

Many of the ideas in this presentation were jointly developed with Doug Hoffman, in a course that we taught together on test automation, and in the Los Altos Workshops on Software Testing (LAWST) and the Austin Workshop on Test Automation (AWTA).

- 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, Elizabeth 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.
- AWTA also reviewed and discussed several strategies of test automation. Participants in the first meeting were Chris Agruss, Robyn Brilliant, Harvey Deutsch, Allen Johnson, Cem Kaner, Brian Lawrence, Barton Layne, Chang Lui, Jamie Mitchell, Noel Nyman, Barindralal Pal, Bret Pettichord, Christiano Plini, Cynthia Sadler, and Beth Schmitz.

I'm indebted to Hans Buwalda, Elizabeth Hendrickson, Alan Jorgensen, Noel Nyman , Harry Robinson, James Tierney, and James Whittaker for additional explanations of test architecture and/or stochastic testing.

The Regression Testing Paradigm

- **This is the most commonly discussed automation approach:**
  - **create a test case**
  - **run it and inspect the output**
  - **if the program fails, report a bug and try again later**
  - **if the program passes the test, save the resulting outputs**
  - **in future tests, run the program and compare the output to the saved results. Report an exception whenever the current output and the saved output don't match.**

THERE ARE MANY ALTERNATIVES THAT CAN BE MORE APPROPRIATE UNDER OTHER CIRCUMSTANCES.

If your only tool is a hammer, everything looks like a nail.

Is this Really Automation?

•**Analyze product**	**--**	**human**
•**Design test**	**--**	**human**
•**Run test 1st time**	**--**	**human**
•**Evaluate results**	**--**	**human**
•**Report 1st bug**	**--**	**human**
•**Save code**	**--**	**human**
•**Save result**	**--**	**human**
•**Document test**	**--**	**human**
•**Re-run the test**	**--**	**MACHINE**
•**Evaluate result**	**--**	**machine *plus human if there's any mismatch***
•**Maintain result**	**--**	**human**

•==========

Woo-hoo! We really get the machine to do a *whole lot* of our work! (Maybe, but not this way.)

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**

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

A Different Class of Tools: Improve 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 Automation in Context

- **Dominant paradigm for automated testing.**
- **Prone to failure**
  - **Expensive to create test cases.**
  - **Unless you design the tests well, they are quickly obsoleted by GUI changes.**
  - **Key tests are often delayed while you automate.**
  - **Your most technically skilled staff are tied up in automation.**
  - **Capture/replay approaches that appear to let you automate without programming typically fail because they generate undocumented, unmaintainable, fragile code.**

There Are Successful Uses . . .

- Smoke testing
- Configuration testing
- Variations on a theme
- Stress testing
- Load testing
- Life testing
- Performance benchmarking
- Other tests that extend your reach

Essential to Design for Maintainability

- **Regression testing typically has low power because:**
  - **Rerunning tests that the program has already passed *usually* doesn't yield many bugs**
- **Therefore, our main payback is often in the next release, not this one.**

Maintainability using a Data-Driven Architecture

- **In test automation, there are three interesting programs:**
  - **The software under test (SUT)**
  - **The automation tool that executes the automated test code**
  - **The test code (test scripts) that define the individual tests**
- **From the point of view of the automation software,**
  - **The SUT's variables are data**
  - **The SUT's commands are data**
  - **The SUT's UI is data**
  - **The SUT's state is data**
- **Therefore it is entirely fair game to treat these implementation details of the SUT as values assigned to variables of the automation software.**
- **Additionally, we can think of the externally determined (e.g. determined by you) test inputs and expected test results as data.**
- **Additionally, if the automation tool's syntax is subject to change, we might rationally treat the command set as variable data as well.**

Data-Driven Architecture

- **In general, we can benefit from separating the treatment of one type of data from another with an eye to:**
  - **optimizing the maintainability of each**
  - **optimizing the understandability (to the test case creator or maintainer) of the link between the data and whatever inspired those choices of values of the data**
  - **minimizing churn that comes from changes in the UI, the underlying features, the test tool, or the overlying requirements**

Data-Driven Architecture: Calendar Example

- Imagine testing a calendar-making program.

  The look of the calendar, the dates, etc., can all be thought of as being tied to physical examples in the world, rather than being tied to the program. If your collection of cool calendars wouldn't change with changes in the UI of the software under test, then the test data that define the calendar are of a different class from the test data that define the program's features.
  - Define the calendars in a table. This table should not be invalidated across calendar program versions. Columns name features settings, each test case is on its own row.
  - An interpreter associates the values in each column with a set of commands (a test script) that execute the value of the cell in a given column/row.
  - The interpreter itself might use "wrapped" functions, i.e. make indirect calls to the automation tool's built-in features.

Data-Driven Architecture: Calendar Example

- This is a good design from the point of view of optimizing for maintainability because it separates out four types of things that can vary independently:
  - *The descriptions of the calendars themselves come from real-world and can stay stable across program versions.*
  - *The mapping of calendar element to UI feature will change frequently because the UI will change frequently. The mappings (one per UI element) are written as short, separate functions that can be maintained easily.*
  - *The short scripts that map calendar elements to the program functions probably call sub-scripts (think of them as library functions) that wrap common program functions. Therefore a fundamental change in the program might lead to a modest change in the software under test.*
  - *The short scripts that map calendar elements to the program functions probably also call sub-scripts (think of them as library functions) that wrap functions of the automation tool. If the tool syntax changes, maintenance involves changing the wrappers' definitions rather than the scripts.*

Data Driven Architecture

- **Note with this example:**
  - **we didn't run tests twice**
  - **we automated execution, not evaluation**
  - **we saved SOME time**
  - **we focused the tester on design and results, not execution.**
- **Other table-driven cases:**
  - **automated comparison can be done via a pointer in the table to the file**
  - **the underlying approach runs an interpreter against table entries.**
    - **Hans Buwalda and others use this to create a structure that is natural for non-tester subject matter experts to manipulate.**

GUI Regression is Just a Special Case

- **Source of test cases**
  - ***Old***
  - Intentionally new
  - Random new
  - Ongoing monitoring
- **Size of test pool**
  - ***Small***
  - Large
  - Exhaustive
- **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
- **Serial dependence among tests**
  - ***Independent***
  - Sequence is relevant

A Different Special Case: Exhaustive Testing

MASPAR functions: square root tests

- 32-bit arithmetic, built-in square root
  - 2^32 tests (4,294,967,296)
  - 6 minutes to run the tests
  - 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?

- 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**
  - **A stochastic process involves a series of random events over time**
    - **Stock market is an example**
    - **Program typically passes the individual tests when run in isolation: The goal is to see whether it can pass a large series of the individual tests.**
- **Independent Random Testing**
  - **Our interest is in each test individually, the test before and the test after don't matter.**

Independent Random Tests: Function Equivalence Testing

Arithmetic in a Spreadsheet

- Suppose we had a pool of functions that worked well in previous version of the spreadsheet.
  - For individual functions, generate random number and take function (e.g. log) in current version and previous version. Compare the results by machine.
    - *Spot check results (perhaps 10 cases)*
  - Build a model to combine functions into expressions
    - Generate and compare expressions
    - Spot check results

- 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

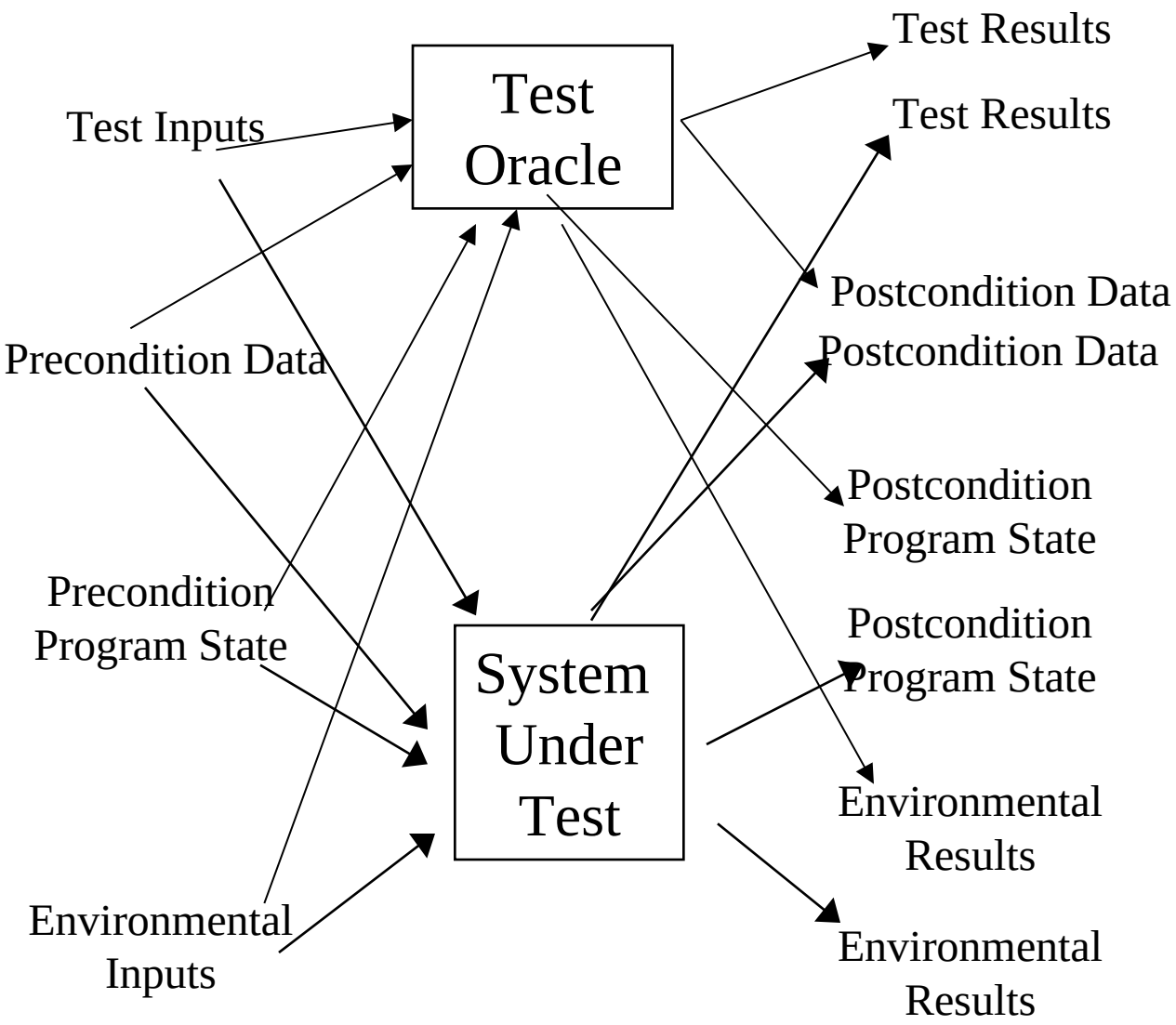
- **Incomplete information from oracle**
  - **May be more than one oracle for SUT**
  - **Inputs may effect more than one oracle**
- **Accuracy of information from oracle**
  - **Close correspondence makes common mode faults likely**
  - **Independence is necessary:**
    - **algorithms**
    - **sub-programs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



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.)

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

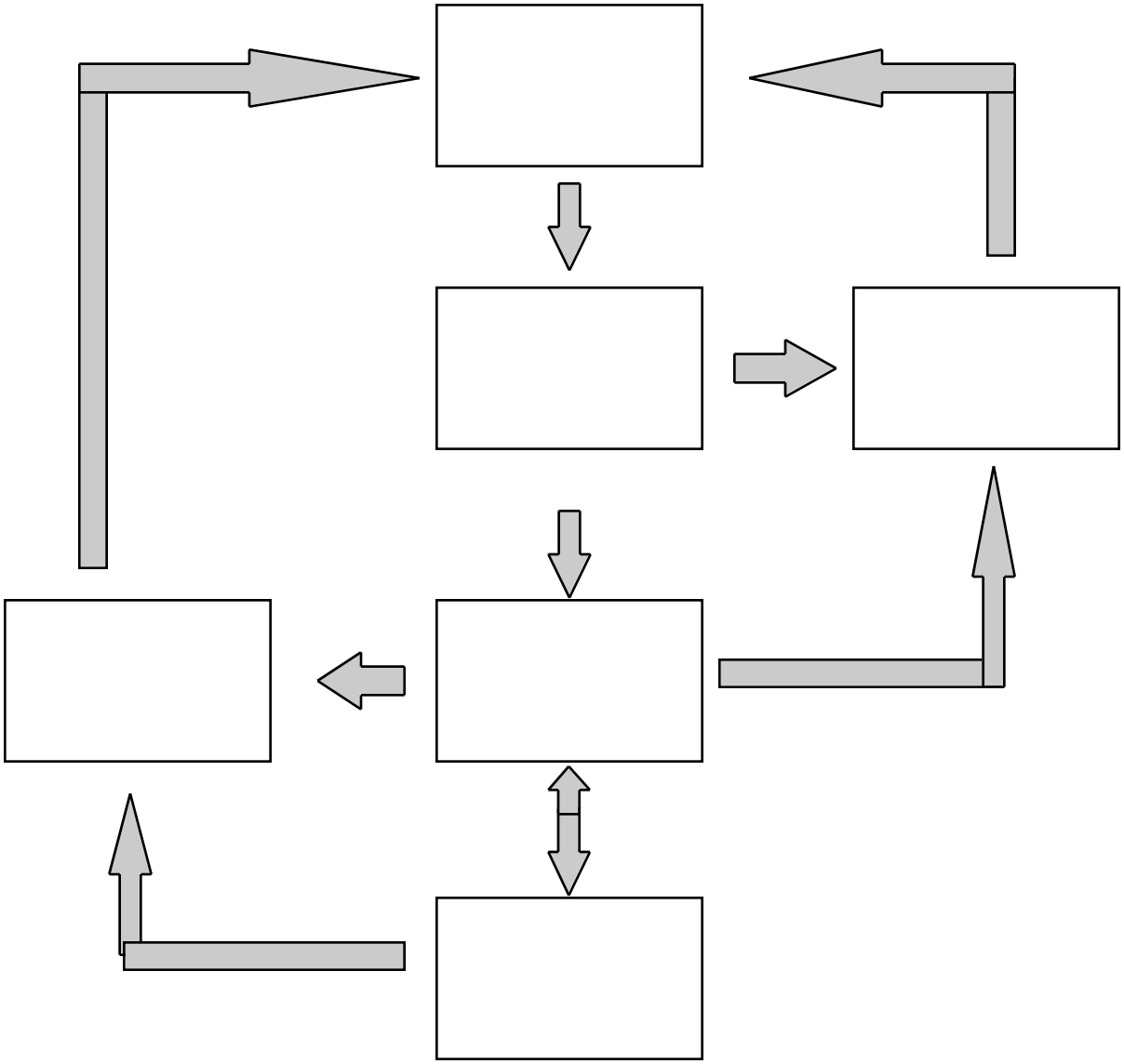
Stochastic Test Using Diagnostics

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.

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

The Need for Stochastic Testing: An Example



– ***Refer to Kaner, The Impossibility of Complete Testing.***

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

---

- Source of test cases
  - Random new
- Size of test pool
  - Large
- 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.

- 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

Another Approach to Evaluating Strategies for Automation

What characteristics of the

- **goal of testing**
- **level of testing (e.g. API, unit, system)**
- **software under test**
- **environment**
- **generator**
- **reference function**
- **evaluation function**
- **users**
- **risks**

would support, counter-indicate, or drive you toward a strategy?

- **consistency evaluation**
- **small sample, pre-specified values**
- **exhaustive sample**
- **random (aka statistical)**
- **heuristic analysis of a large set**
- **embedded, self-verifying data**
- **model-based testing**

For more on this, see the paper in the proceedings.