

---

*Test Automation*  
*Beyond Regression Testing*

**Doug Hoffman, BA, MBA, MSEE, ASQ-CSQE**  
**Software Quality Methods, LLC. (SQM)**

**[www.SoftwareQualityMethods.com](http://www.SoftwareQualityMethods.com)**

**[doug.hoffman@acm.org](mailto:doug.hoffman@acm.org)**

STPCon Spring 2008

---

# *Why Automate Tests*

---

*The creative (and hardest) part of testing is designing good tests*

- **All the rest are technical implementation details that can be automated (or not)**
- **Test automation should reduce the burden of the technical implementation and maintenance details so the humans can concentrate on creating excellent tests**

# *Advantages of Automated Tests*

---

- Discovery of defects manual testing cannot expose
- Tests are repeatable
- Monitoring of information not visible to a person
- Stimulation of the software under test (SUT) through APIs
- Massive numbers of actions not possible manually
- Pseudo random event and value generation to generate variation in the activities of the SUT
- Automated verification of very large numbers of outcomes and potential error conditions not directly associated with the functions being exercised

# *Automated Regression Pros and Cons*

---

## Advantages

- Dominant automation paradigm
- Conceptually simple
- Straightforward
- Same approach for all tests
- Fast implementation
- Variations are easy
- Repeatable tests

## Disadvantages

- Breaks easily (GUI based)
- Tests are expensive
- Pays off late
- Prone to failure because:
  - difficult financing,
  - architectural, and
  - maintenance issues
- Low power even when successful (finds few defects)

# *Why Regression Tests Are Weak*

---

- Does the same thing over and over
- Finds most defects during test creation
- Only verifies things programmed into the test
- Automation reduces test variability
- Software doesn't break or wear out
- Any other test is equally likely to stumble over unexpected side effects

We **MINIMIZE** the **MINIMAL** chance of finding errors by automating regression testing.

# *Demo Creation Process*

---

## **A demo creation technique for new products:**

- conceive and create the demo exercise
- run it to see that it demonstrates the desired attributes
- if the program fails, report a defect
- in case of a defect, either wait for a fix or find a different way to do the exercise that doesn't fail
- remember the specific steps and values that work
- in future demos, do not deviate from the steps or try new values to minimize the chance of failure

*This is the same as a regression test!*

*It's designed to MINIMIZE the chance of encountering a defect!*

# *Making More Powerful Exercises*

---

**Increase the number of combinations**

**More frequency, intensity, duration**

**Increase the variety in exercises**

**Use the computer to extend your reach**

- Setting conditions
- Monitoring activities
- Controlling the system and SUT

**Self-verifying tests and diagnostics**

# *You Can't Do This Manually!*

---

## **Extending our reach**

- **API based testing**
- **Component testing**
- **Internal monitoring and control**
- **Massive input generation**
- **Real time oracles**
- **Large scale result checking**
- **Provide hooks and scaffolding**

## **Multiplying resources**

- **Platform testing**
- **Configuration testing**
- **Model based tests**
- **Data driven tests**

# *The Test Oracle*

---

The oracle is the mechanism by which we tell “pass” from “fail”

- ***Reference Function:*** You ask it what the “correct” answer is. *(This is how I usually use the term.)*
- ***Reference and Evaluation Function:*** You ask it whether the SUT passed the test. *(Some oracles work this way)*

Using an oracle, you can compare the program’s result to a reference value (predicted or expected value) and decide whether the program passed the test.

- ***Deterministic oracle*** (mismatch means program fails) *(This is the commonly considered case.)*
- ***Probabilistic oracle*** (mismatch means investigation is warranted.) *(These often lead to more powerful automation.)*

# Oracle Strategies for Verification

	No Oracle	True Oracle	Consistency	Self Referential	Heuristic Strategy
Definition	-Doesn't check correctness of results, (only that some results were produced)	-Independent generation of all expected results	-Verifies current run results with a previous run (Regression Test)	-Embeds answer within data in the messages	-Verifies some values, as well as consistency of remaining values
Advantages	-Can run any amount of data (limited only by the time the SUT takes)	-No encountered errors go undetected	-Fastest method using an oracle -Verification is straightforward -Can generate and verify large amounts of data	-Allows extensive post-test analysis -Verification is based on message contents -Can generate and verify large amounts of complex data	-Faster and easier than True Oracle -Much less expensive to create and use
Disadvantages	-Only spectacular failures are noticed.	-Expensive to implement -Complex and often time-consuming when run	-Original run may include undetected errors	-Must define answers and generate messages to contain them	-Can miss systematic errors (as in <i>sine</i> wave example)

# *Automated Test Architectures*

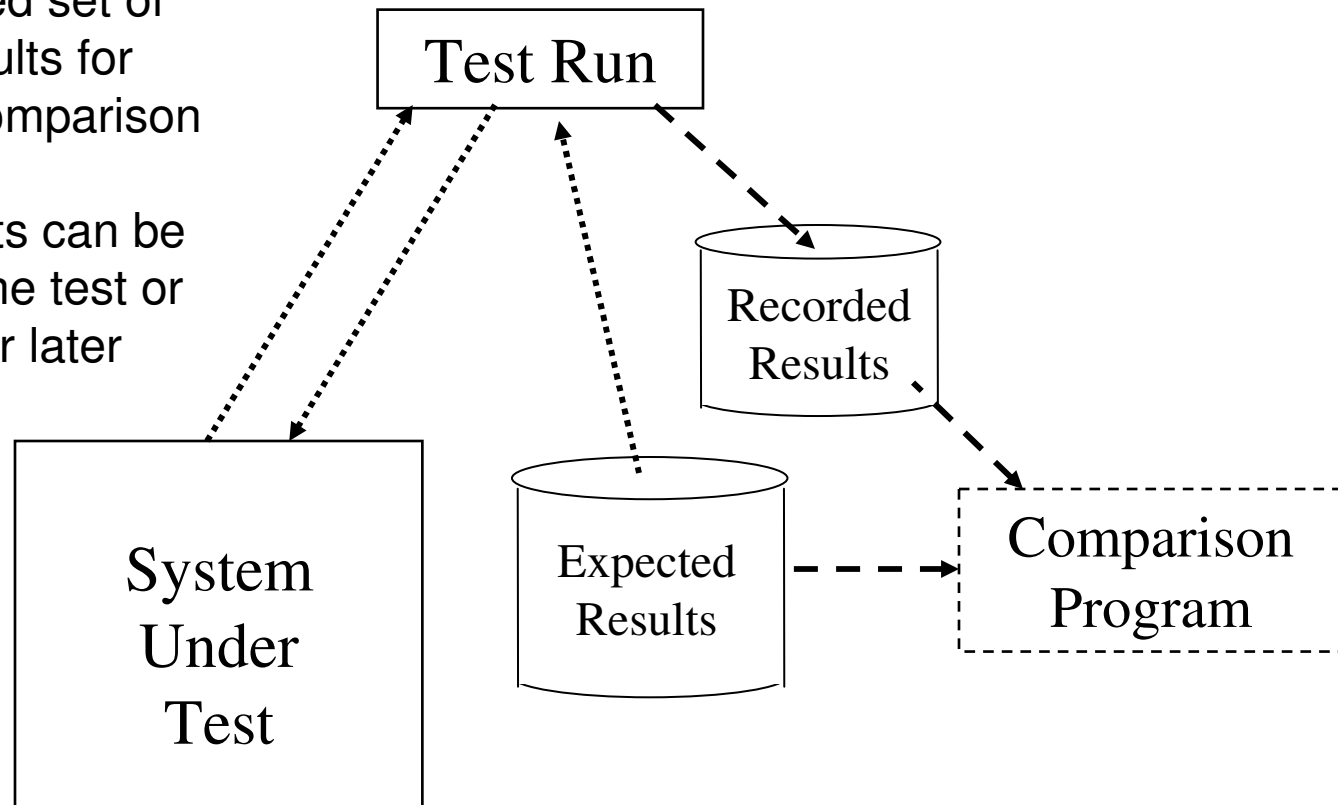
---

- **Equivalence Testing**
- **Scripted Testing**
- **Framework Based**
- **API Based**
- **Load/Stress/Performance Testing**
- **Data Driven Testing**
- **Stochastic or Random Testing**
- **Model Based**

# *A/B Comparisons*

An A/B comparison has a recorded set of expected results for automated comparison

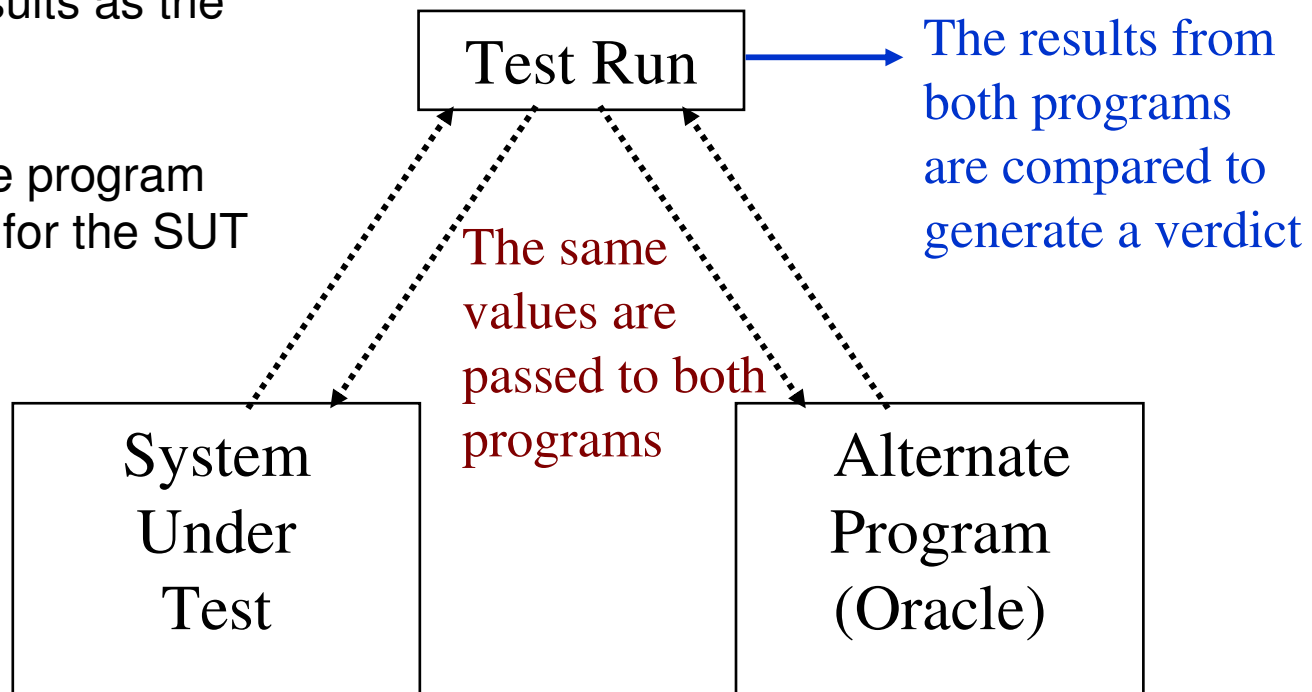
Current results can be checked by the test or put in a file for later comparison



# Function Equivalence Testing

A Function Equivalence Test uses an alternate program to generate expected results as the test runs

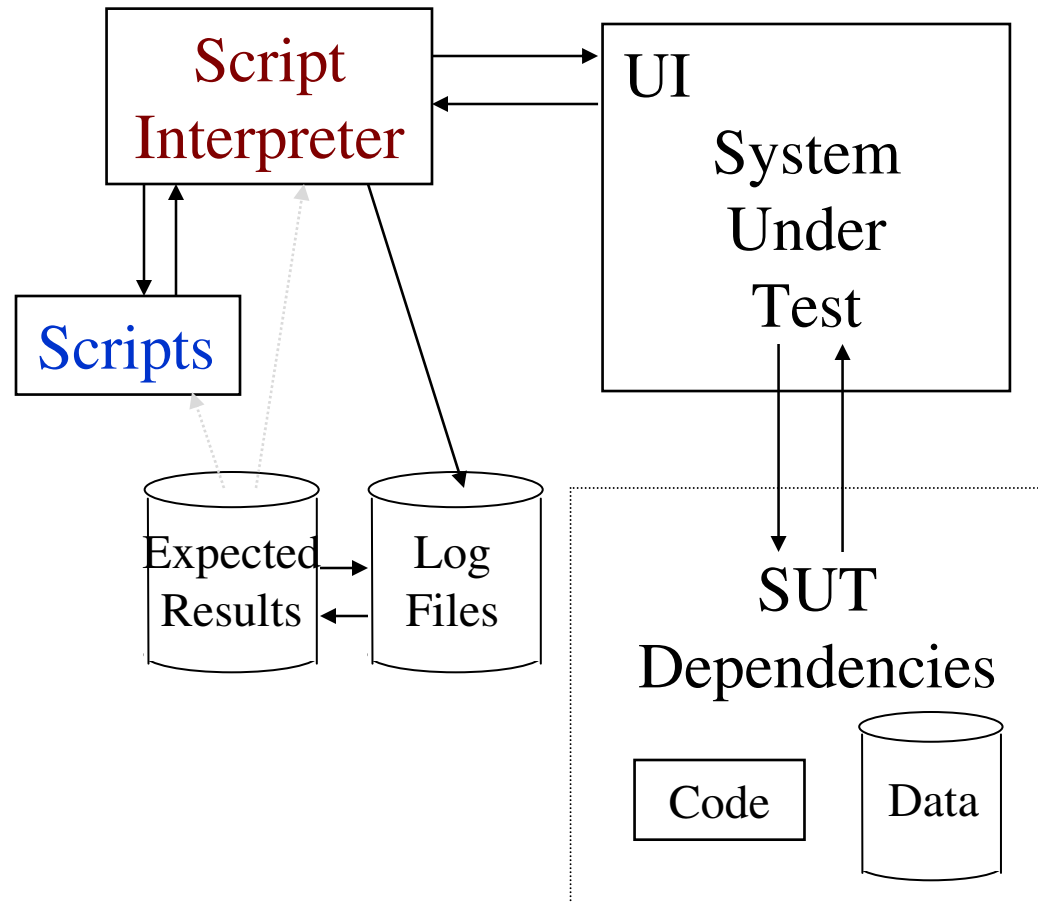
The alternate program is the oracle for the SUT



# Script Based Testing<sub>1</sub>

Tester starts a *Script Interpreter* program, which uses *Scripts* (programs) to exercise the SUT

Results are saved in *Log Files* for comparison with *Expected Results* (or compared as the results are generated).



# *Framework-Based Architecture*

---

**Frameworks are code libraries that separate routine calls from designed tests.**

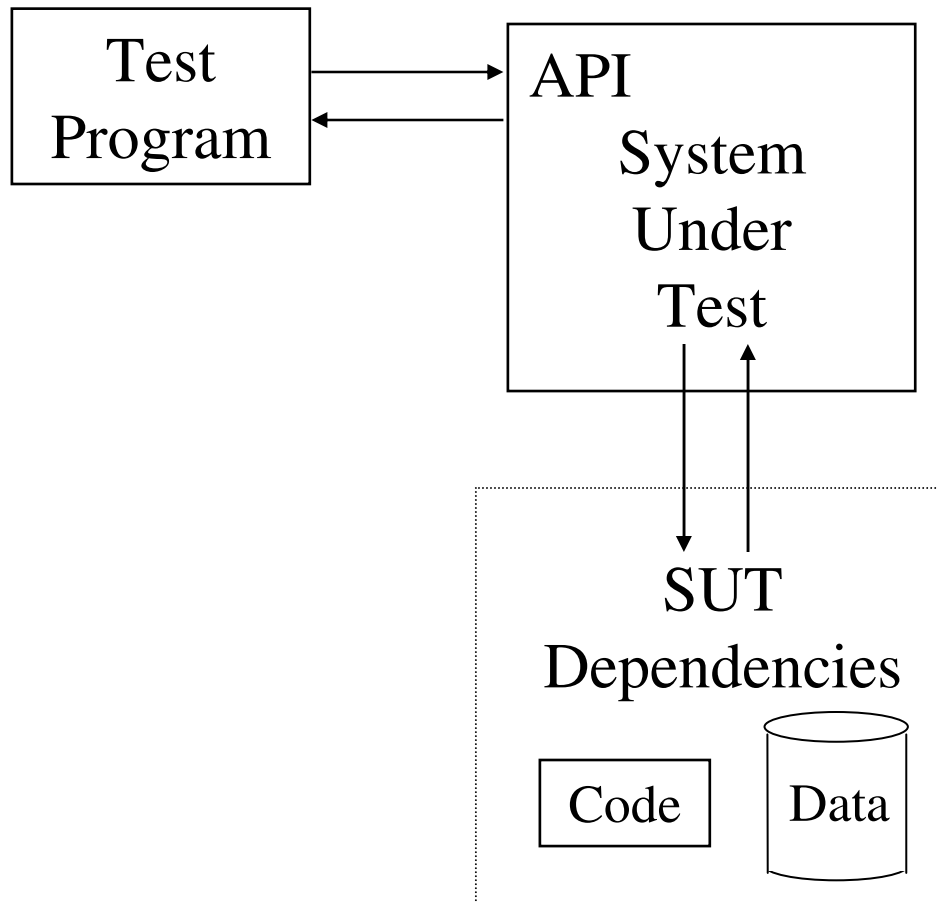
- modularity
- reuse of components
- hide design evolution of UI or tool commands
- partial salvation from the custom control problem
- independence of application (the test case) from user interface details (execute using keyboard? Mouse? API?)
- important utilities, such as error recovery

**For more on frameworks, see Linda Hayes' book on automated testing, Tom Arnold's book on Visual Test, and Mark Fewster & Dorothy Graham's excellent new book "Software Test Automation."**

# Code Based API Testing

For code based API testing, each test case is a program

The program provides the input and analyzes the result



# *Load and Stress Testing* (1 of 2)

---

- **Load: background for other tests**
  - Performance analysis
  - Collision testing
- **Stress: push it past the limits**
  - Classically: up to and just past the limits
  - More recently: overwhelm the product

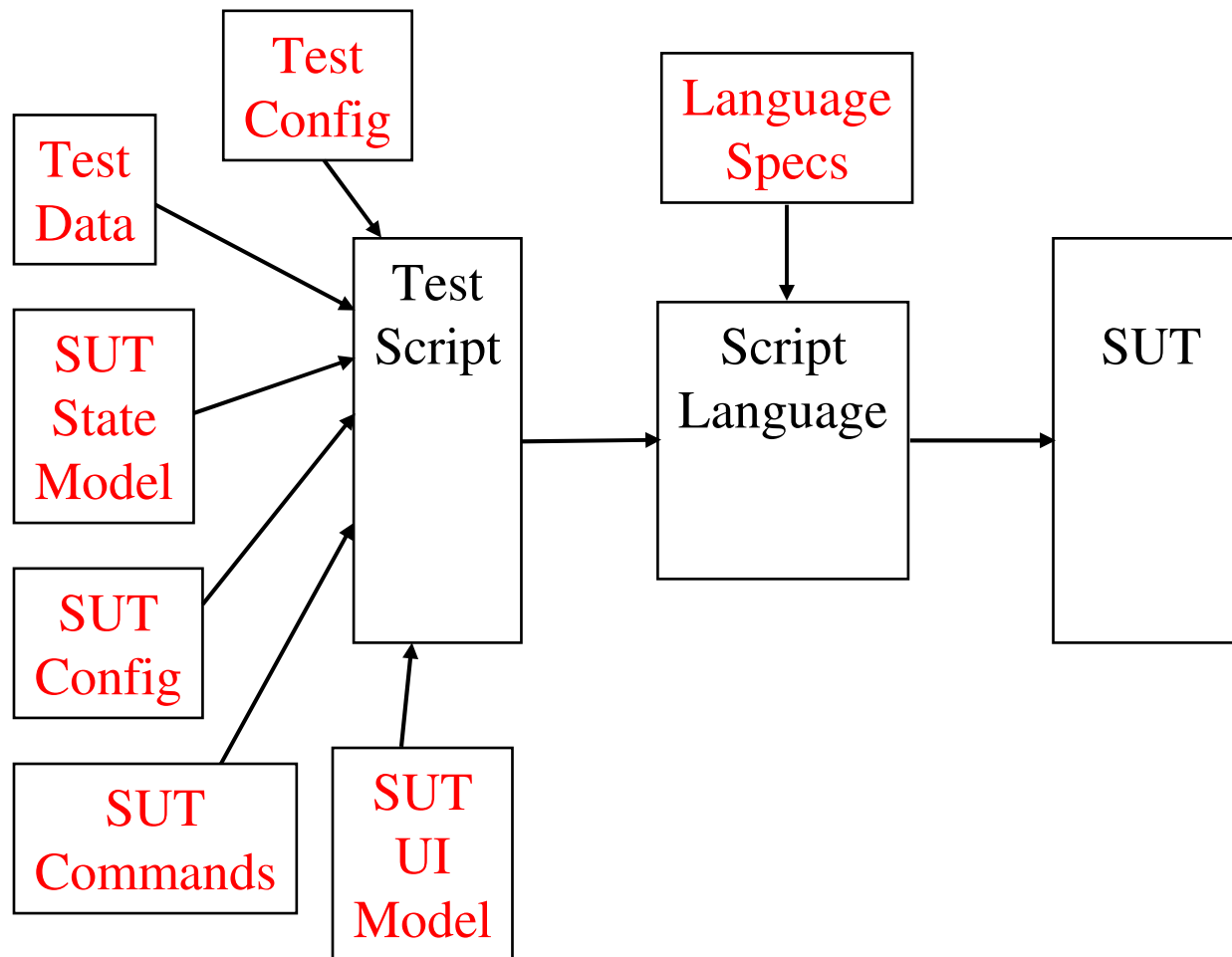
# *Load and Stress Testing* (2 of 2)

---

- **Methods for load and stress; increase:**
  - Frequency (how often)
  - Intensity (how much each time)
  - Duration (how long)
- **Reuse automated regression tests**
- **Create load or stress tests**

# Data Driven Architectures

---



# *Random, Independent, and Stochastic Approaches*

---

## **Random Testing**

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

## **Independent Tests**

- A set of tests where the previous and subsequent tests don't matter

## **Stochastic Testing**

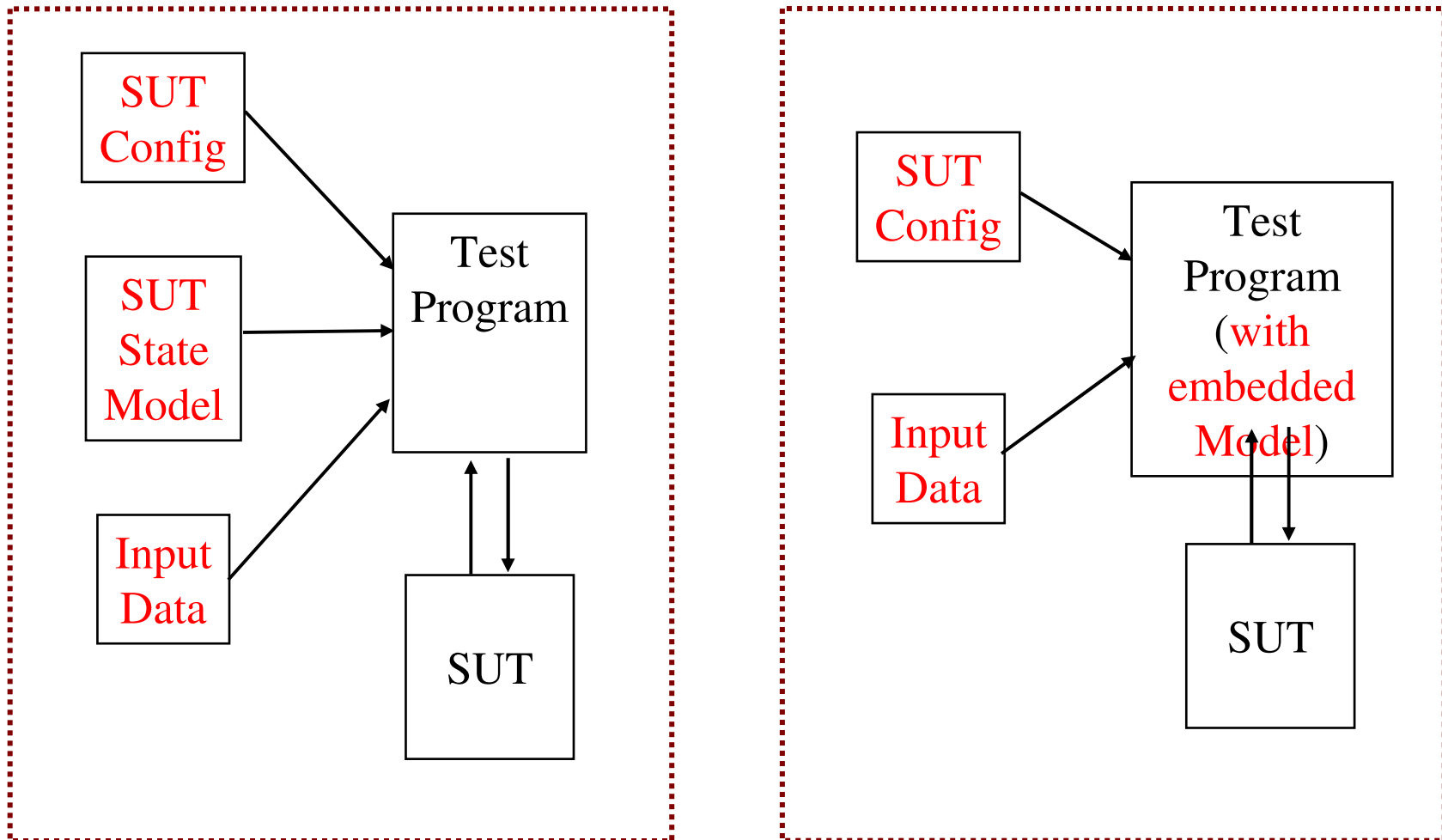
- Stochastic process involves a series of “random” events over time
  - » Program typically passes the individual tests: The goal is to see whether it can pass a large series of the individual tests
  - » Random walk in a state machine is an example

# *Model Based Automation*

---

- **Test embodies rules for activities**
  - Stochastic process event generation
  - Well formed random values
- **Possible monitors**
  - Code assertions
  - Event logs
  - State transition maps
  - Other oracles

# Model Based Architectures



# *Conclusions*

---

**Automated tests can do more than simple regression testing**

**Focus automation on doing things that cannot be done manually**

**Many possible architectural approaches for more powerful automated tests**

**A good oracle is critical to make automated tests worthwhile**

