The Often Overlooked Test Oracle

AST Webinar
May 3, 2019

Doug Hoffman
Software Quality Methods, LLC.
Doug.Hoffman@acm.org
http://www.SoftwareQualityMethods.com

A Question About SUT Misbehavior

What can happen when there is a bug in the SUT?

Anything!

Anywhere!

What is Testing?

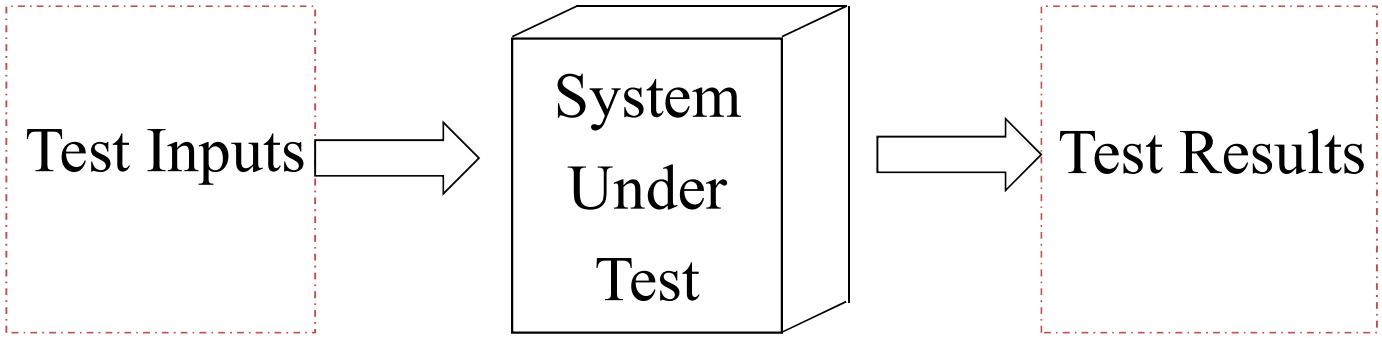
A technical investigation of the product under test conducted to provide stakeholders with quality-related information.

Cem Kaner

What Comprises a Test?

1. An exercise (stimulation of the SUT)
   - Setup before the test
   - Inputs (usually a series of steps and/or values)
2. Gathering of relevant data
   - May be test case specific
   - May be independent of the test case
   - Can occur before, during and/or after the exercise
3. Analysis of the data to determine a verdict from a test or test set

Typical Test Execution Model



Influence **Outcome**

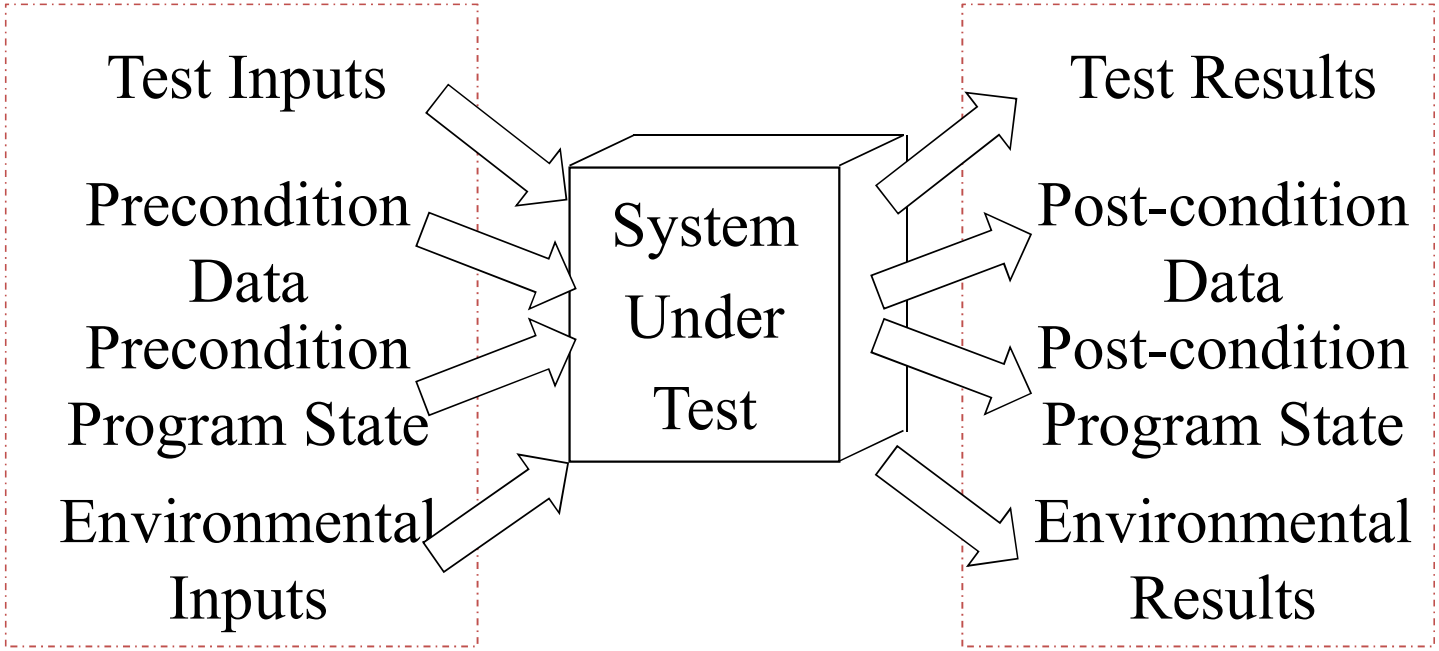
Implications of the Simple Model

- We control all the inputs
- We can verify all the results

But, we aren't dealing with all the factors

- Data (in memory and external data sets)
- Program state
- System environment

Expanded Test Execution Model



Implications of the Expanded Model

- We don't control all inputs
- We don't verify everything
- Multiple domains are involved
- The test exercise is usually the easy part
- We don't (can't) verify everything
- We don't (can't) know all the factors

I feel oh, so much better,
now that I've given up hope!

Defining A Test Oracle

The mechanism or principle by which we determine whether or not the software under test is behaving reasonably.

- *Unreasonable SUT behavior requires investigation by a person.*
- *When the oracle says the SUT behavior is unremarkable, we just move on.*

Oracle at Delphi

Notes On Test Oracles

- At least one type is used in every test
- Any particular instance of an oracle may blend characteristics of multiple types of oracles
- Multiple oracles may be used for one test
- Adding an oracle makes a test stronger
- Oracles may be independent of, or integrated into a test
- The oracles are key to good automated tests

Types of Comparisons

- Comparing the program's behavior to a reference of expected behavior may be inexact
  - *Deterministic oracle* (mismatch means behavior is abnormal)
  - *Probabilistic oracle* (indicates when behavior is warrants further investigation)
- Only sometimes is comparison done within the test
- Often checking is a separate operation

Types of Test Oracles

- No Oracle (Crash)
- Independent implementation
- Consistency
  - Saved master
  - Function equivalence
- Self-Verifying data
- Model based
- Constraint based
- Probabilistic
- Computational
- Statistical
- Property based
- Diagnostic
  - Hand-crafted
  - Human

'No Oracle' Strategy

- Method:
  - Generate [usually random] inputs
  - Run the test
  - Ignore the outcomes
- Easy to do
- Tests can run fast
- Only spectacular events are noticed
- May give a false sense of accomplishment

'Independent Implementation' Strategy

- Independent implementation
- Complete coverage over domains
  - Input ranges
  - Result ranges
- Generates "Correct" (trusted) results
- Usually expensive
- Can be more complex than the SUT

'Consistency' Saved Master Strategy

- Checks for differences (changes)
- Primarily used for regression checking
  - The most frequently used automation strategy
  - May be validated (difference means a bug)
  - Or, unvalidated (difference means investigate)
- Most common methods:
  - The test generates a log of activity
  - Compare this run's log with a previous run
  - Sometimes called a "Golden Master" approach

'Consistency' Functional Equivalence Strategy

- Checking for differences using high volume input and an alternative program
  - Validated (trusted results)
  - Unvalidated (unknown value of the results)
- Most common methods:
  - Use a competing product or another version of the SUT (e.g., another platform)
  - Feed test data into SUT and similar product
  - Compare results

'Self-Verifying Data' (SVD) Strategy

- Method builds expected outcomes into the data
  - Self-Descriptive data (**RED**)
  - Cyclic algorithms (Simple pattern to the data (e.g., start, increment, count))
  - Shared keys (with algorithms)
    - Random data generated from a seed value (a key)
    - Embed the key within the data
    - Use the algorithm and key to regenerate the data

'Model Based' Strategy

- Method:
  - Identify and describe a [machine readable] model of some aspect of the SUT (e.g., state machine states and events)
  - Design tests using the model as input
  - Implement the tests (reading in and applying the model)
  - Use the same model to check results within the test
  - Update the model as needed when code changes

'Constraint-Based' Strategy

- Look for simple valid (or invalid) individual or combinations of data values or characteristics
  - Limits on values (e.g., max 256 character names)
  - Values that constrain one another (e.g., spreadsheet cells)
  - Size, form, type, illegal values, etc. (e.g., page width)
  - Invariant rules (e.g., date of birth before hire date)
- Checking
  - Create checking mechanism to confirm conformance with the constraints (inside or outside the code)
  - Synchronous or asynchronous checking
  - May be test case specific or independent checking

'Probabilistic' Strategy

- Looks for relationships that usually, but not always hold
  - An approximation, a heuristic (rule of thumb), or partial information that supports but does not necessarily mandate a given conclusion
- "Error" means it probably is a problem
- Examples:
  - Employee is usually older than their dependents
  - A test should run between 1/3 and 3 times as fast as it took the last time it ran

'Computational' Strategy

- Principle idea:
  - Perform the reverse (inverse) function
  - Identify whether the results are possible
- The oracle:
  - Applies the reversal function on the results
  - Computes the possible value(s) for the inputs
  - Checks that the actual inputs are in the set of possible values
  - May be subject to common-mode problems
  - May miss obvious errors
- Example:
  - Input *X* to the square root function; square the result

'Statistical' Strategy

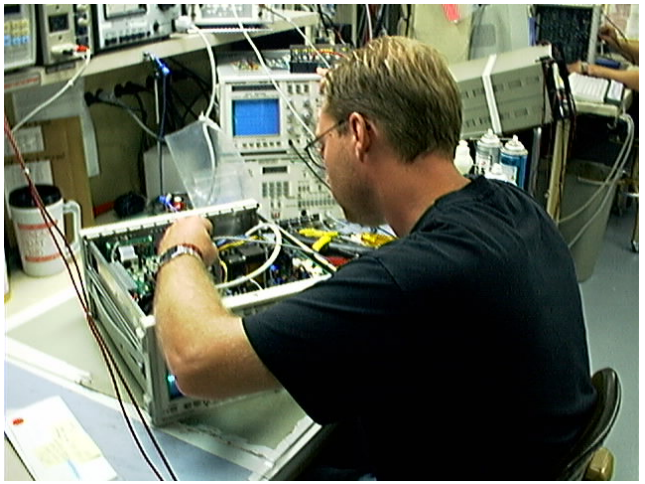
- Principle idea:
  - Requires a known mathematical relationship between inputs and results
  - Uses high-volume random tests
  - Checks results based on population statistical characteristics
- The method:
  - Computes the statistical characteristics of the inputs
  - Computes the statistical characteristics from the results
  - Compares the statistics in light of the expected transformation through the SUT

'Property-Based' Strategy

- Use a secondary characteristic of a test, variable, or variables
  - Compare coincidently correlated relationships between variables
  - Not necessarily complete
  - Not necessarily causal
- Examples:
  - Sales Order Number should be in time-sequence order
  - Checking the number of pages printed by a test
  - USA ZIP code is either 5 or 9 digits

'Diagnostic' Strategy

- Track execution of the code
  - Instrument the code (using assertions or logging)
  - Run tests and check assertions or trace execution
- Method
  - Code assertions (e.g., must be logged in with a valid UID)
  - Trace files generated by SUT (e.g., print a log of data values)



'Hand-Crafted Oracle' Strategy

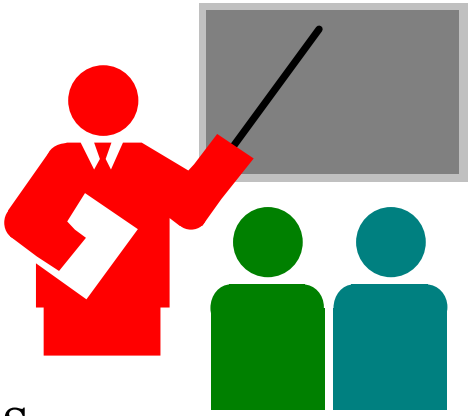
- Inputs and results are specified together
- Expected result is carefully crafted (or selected) with the input values
- The oracle is frequently built into the test
- This approach is often taken for manual regression tests in complex SUTs

'Human Oracle' Strategy

- Set a person in front of the SUT to observe
- Human uses their judgment to decide the verdict
- Works for manual or automated exercises
- Works for scripted or unscripted tests

Note that a human oracle is applied whenever any other oracle identifies a potential bug

Recapping



- All tests use oracles
- There are many types of oracles
- It is impossible to check everything
- Oracles do not need to be deterministic to be useful
- Oracles can be independent of tests