What Is Software Testing? And Why Is It So Hard?

James A. Whittaker, Florida Institute of Technology

IEEE Software 17(1), pp. 70-79, Jan-Feb 2000

Avital Braner

Basic Seminar of Software Engineering

Hebrew University 2009

Software testing

- The process of executing a software system to determine whether it matches its specification and executes in its intended environment.
- When the user reports a bug:
  - The user executed untested code
  - The order in which statements were executed in actual use differed from that during testing
  - The user applied a combination of untested input values
  - The user's operating environment was never tested

The Testing Process

- To get a clearer view of some of software testing's inherent difficulties, we can approach testing in four phases:

Modeling the Software's Environment

Selecting Test Scenarios

Running and Evaluating Test Scenarios

Measuring testing progress

Phase 1: Modeling the Software's Environment

- Testers must identify and simulate the interfaces that a software system uses and enumerate the inputs that can cross each interface.
- Four common interfaces are as follows:
  1. Human interfaces
  2. Software interfaces
  3. File system interfaces
  4. Communication interfaces

Phase 1: Modeling the Software's Environment

- Testers must understand the user interaction that falls outside the control of the software under test:
  - Deletes a file that another user has open
  - A device gets rebooted in the middle of a stream of communication
  - Two software systems compete for duplicate services from an API

Phase 1: Modeling the Software's Environment

- Testers face two difficulties:
  1. select values for any variable input
  2. decide how to sequence inputs

- *The boundary value partitioning technique* - selecting single values for variables at or around boundaries

Phase 1: Modeling the Software's Environment

- In deciding how to sequence inputs, testers define a model.
- The most common model:
  a graph or state diagram.
- Other popular models:
  regular expressions and grammars,
  tools from language theory.
- Less-used models:
  stochastic processes and genetic algorithms.

Phase 1: Modeling the Software's Environment

- In deciding how to sequence inputs, testers

Text Editor Example:

Filemenu.Open filename* (ClickOpen | ClickCancel)

- Other popular models:
  regular expressions and grammars, tools from language theory.
- Less-used models:
  stochastic processes and genetic algorithms.

Phase 2: Selecting Test Scenarios

- There are infinite number of test scenarios, but only a subset can be applied.
- Testers strive for coverage :
  - covering code statements
  - covering inputs
- But if code and input coverage were sufficient, released products would have very few bugs
  - execution paths
  - input sequences

Phase 2: Selecting Test Scenarios

- the *best* possible **test data *adequacy* criteria**
  - the set that will find the most bugs
  - typical use scenarios

Text Editor Example:

Typical use: editing and formatting.
However, to find bugs, a more likely place to look is in the harder-to-code features, like figure drawing and table editing.

Phase 2: Selecting Test Scenarios

- **Execution path test criteria:**
  - Control Flow
    - Set of tests that cause each source statement to be executed at least once.
    - Set of tests that cause each branching structure to be evaluated with each of its possible values.
  - Dataflow
    - Set of tests that cause each data structure to be initialized and then subsequently used.
  - Fault Seeding
    - Set of tests that expose each of the seeded faults.

Phase 2: Selecting Test Scenarios

- **Input domain test criteria**
  - Simple coverage
    - Select a set of tests that contain each physical input.
    - Select a set of tests that cause each interface control to be stimulated.
  - The discrimination criterion
    - Select a set of tests that have the same statistical properties as the entire input domain.
    - Select a set of paths that are likely to be executed by a typical user.

Phase 2: Selecting Test Scenarios

- Most researchers would agree that it is prudent to use multiple criteria when making important release decisions.
- Testers should be aware which criteria are built into their methodology and understand the inherent limitations.

Phase 3: Running and Evaluating Test Scenarios

- Running:
- Testers try to automate the test scenarios as much as possible.
- Testers often include data-gathering code in the simulated environment as test hooks or asserts.

Phase 3: Running and Evaluating Test Scenarios

- Scenario Evaluation:
- The comparison of the software's actual output, resulting from test scenario execution, to its expected output as documented by a specification.
- Easily stated but difficult to do.
  Usually performed by a human *oracle.*

Text Editor Example:

If the output is supposed to be "highlight a misspelled word," how can we determine that each instance of misspelling has been detected?

Phase 3: Running and Evaluating Test Scenarios

- **Two approaches to evaluating your test:**
  1. Formalism
  2. Embedded test code.
     - test code that exposes certain internal data objects or states
     - self-testing programs

Phase 4: Measuring Testing Progress

- Counting measures yield very little insight about the progress of testing.
- Therefore, many testers answering questions designed to ascertain structural and functional testing completeness.

Phase 4: Measuring Testing Progress

- Have I tested for common programming errors?
- Have I exercised all of the source code?
- Have I forced all the internal data to be initialized and used?
- Have I found all seeded errors?
- Have I thought through the ways in which the software can fail and selected tests that show it doesn't?
- Have I applied all the inputs?
- Have I run all the scenarios that I expect a user to execute?

Phase 4: Measuring Testing Progress

- Determining when to stop testing is more complex.
- If testers can achieve a measure of the number of bugs left in the software and of the probability that any of these bugs will be discovered, they know to stop testing.

Phase 4: Measuring Testing Progress

- From a structural standpoint: *Testability*
  - Jeffrey Voas has proposed testability.
  - Testability is a compelling concept but in its infancy. No data on its predictive ability has yet been published.

Phase 4: Measuring Testing Progress

- From a functional standpoint: *Reliability models*
  - Mathematical models
  - Most of them require a description of how users are expected to apply inputs.
  - To compute the probability of failure, These models make some assumptions about the underlying probability distribution that governs failure occurrences.
  - Researchers and practitioners alike have expressed skepticism, but successful case studies have shown these models to be credible.

Summary

- The first and most important thing to be done is to recognize the complex nature of testing and take it seriously.
- The author advice: Hire the smartest people you can find, help them get the tools and training they need to learn their craft, and listen to them when they tell you about the quality of your software. Ignoring them might be the most expensive mistake you ever make.

A Sample Software Testing Problem

- Two input sources:
  - the human user who supplies inputs from the set
  - the operating system “user” that supplies memory, the current system time, and date as an application service.

A Sample Software Testing Problem

- Consider the valid and invalid inputs from each of these sources:
  - other Altsequences or keystrokes
  - available memory is insufficient
  - the system clock is malfunctions
- How users interact in ways that might cause the software to fail?
  - some other program changes the time and date

A Sample Software Testing Problem

- How many different times are there in a day?
  12 hours X 60 minutes X 60 seconds
  X 2 am/pm = 86,400 different input values
- Invalid values like 29 o'clock must also be tested
- Which inputs will be applied consecutively?
  - several consecutive Tab keys
  - a change to the Time field only
  - a change to the Date field only
  - changes to both

A Sample Software Testing Problem

```
Input = GetInput()
While (Input ≠ Alt-F4) do
  Case (Input = Time)
    If ValidHour(Time.Hour) and ValidMin(Time.Minute) and
      ValidSec(Time.Second) and ValidAP(Time.AmPm)
    Then
      UpdateSystemTime(Time)
    Else
      DisplayError("Invalid Time.")
    Endif
  Case (Input = Date)
    If ValidDay(Date.Day) and ValidMnth(Date.Month) and
       ValidYear(Date.Year)
    Then
      UpdateSystemDate(Date)
    Else
      DisplayError("Invalid Date.")
    Endif
  Case (Input = Tab)
    If TabLocation = 1
    Then
      MoveCursor(2)
      TabLocation = 2
    Else
      MoveCursor(1)
      TabLocation = 1
    Endif
  Endcase
  Input = GetInput()
Enddo
```

28 test cases

Possible cases	While	Case 1	If 1	Case 2	If 2	Case 3	If 3
1	F	–	–	–	–	–	–
2	T	T	T	–	–	–	–
3	T	T	F	–	–	–	–
4	T	F	–	T	T	–	–
5	T	F	–	T	F	–	–
6	T	F	–	F	–	T	T
7	T	F	–	F	–	T	F
8	T	F	–	F	–	F	–