Black Box Testing* – CS 314 Spring 2016

James M. Bieman
Sudipto Ghosh
Geri Georg

Terminology:

Test input
- A set of values given as input to a program
- Includes environment variables
- Example: {2, 3, 6, 5, 4}

Test case
- Test input with expected output
- Example: ({2, 3, 6, 5, 4}, "yes") – here the inputs are in curly braces, and the expected output is "yes"

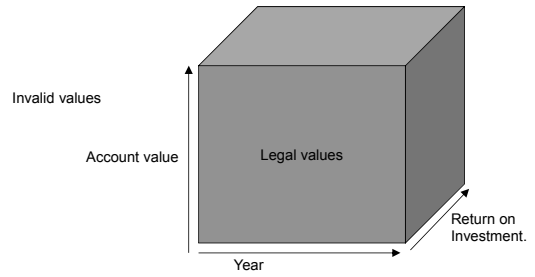
Test set
- A set of test cases

* The term "black box testing" is considered obsolete by some researchers. The idea of BB testing is that you don't know any specifics of how a program is written, just what outputs it should give for a certain set of inputs. Researchers sometimes use the term "top-down" testing. In this case the program is thought of as a graph where nodes are methods and edges are calls. The graph starts with the main method of the program. Testing starts with the main method too, and then proceeds to the methods called down through the tree. BB testing is also a kind of "dynamic testing", where the program is actually run in order to view results.

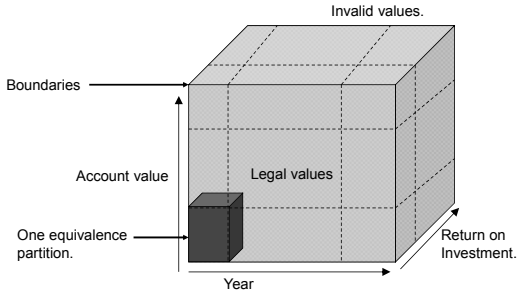
Black-Box Class Testing

- Black-Box testing: test a "component" taking an external view.
  - Use the specification to derive test cases.
  - No access to source code.
- Black-Box class testing.
  - Generate tests by analyzing the class interface.
  - Don't look at method bodies.
- Look at the class in isolation, and in conjunction with other associated classes.
- Test each class method, and test sequences of messages that class objects should respond to.
- May need stubs and/or test drivers.
- Group class objects into categories.
- Test each method for each category.
- A *test plan* documents all of the tests to be performed.

An Infinite Number of Possible Inputs, But a Finite Number of Tests



Partition Inputs and Test Boundaries



Determining Equivalence Partitions

- Find an ordering of the class objects.
- Example: Finding Java String objects to test.
  - Canonical ordering of string objects: "", "a", "b", ..., "aa", "ab", ..., "zz...zz"
- Use the ordering to select test objects.

Use the Ordering to Select Test Cases

- Find objects at extremes and next to extremes:
  - Minimum size: "", "a"
  - Long strings: "zz...zz", "zz...zy"
- Middle length Strings.
- Different types of Strings:
  - numbers
  - control characters: "^D^C"
  - symbols: "&$@+-><"
- Invalid strings: a null String variable. For C++, you can set a String variable to an integer.

Ex: Testing Java Class *Stack*

Stack method interface:
- boolean empty()
- Object peek()
- Object pop()
- void push(Object element)
- int search(Object element)
  Boo Hiss!! This is non-stack type of operation.
- Object Stack(): The default constructor.

Classify the Operations

- Constructors/Destructors:
  - Stack()
- State changing operations:
  - pop()
  - push(Object e)
- Non-state changing operations:
  - empty()
  - peek()

Test Each Type of Operation

- Constructors: test each constructor with all orderings of parameter boundary values.
- Destructors: test with each constructor.
- State changing operations: try to change the object state from every "state" to every other "state".
- Non-state changing operations: test on stacks in each "state".

"State"

- Really a group of related states.
- Example stack states:
  - Empty stacks,
  - Mid-size stacks,
  - Just under the maximum size stack,
  - Large or full stacks.
  - Empty stacks,
  - Mid-size stacks,
  - Just under the maximum size stacks.

Testing *Multiplicity*

- Create several stacks and test, alternating between them.
- This will determine whether each stack object has an independent state (independent instance variables).

The Test Oracle

- How do you know if an item is successfully pushed onto a stack?
- Examine the behavior of the resulting stack after the push operation is performed.