SOFTWARE TESTING

Testing is the process of executing a program with the intent of finding bugs

– GLEN MYERS

“Testing can show the presence of bugs but never their absence.”

– DIJKSTRA

TRUTHS ABOUT SOFTWARE TESTING

- Your program contains defects. Find them.
- Finding defects early means that they are cheaper to fix.
- Difficult for **you** to test **your** program - because you make implicit assumptions
- As the number of detected defects in a piece of software increases, the probability of the existence of more undetected defects also increases
- Testing and debugging are easier if you make your software easy to test and to debug
- It is only when we think about testing we come up with complete specifications

TEST CASES

- A test case is a set of input values, expected output and preconditions for executing a test that aims at finding errors
- eg, for sum(x,y), a test case can be “sum(3,2) should give 5”
- for divideXbyY(x,y), a test case can be “divideXbyY(3,0) should give error”
- See other test cases here.

- The test case fails if the expected result differs from the actual result

SQLITE

- The implementation has 78,000 lines of code. How many lines in the test suite?

91 MILLIONS

100 PERCENT BRANCH COVERAGE AND 100% MCDC COVERAGE

GOOD TEST CASES

- Make sure test case data are PRECISELY defined. a complete idiot must be able to understand what to do.
- Before you can run test cases you need to prepare the system for it (clean usernames, data, etc... insert test data....)
- Try to write dependent cases in sequence, as they are supposed to be run, so a previous execution generates the data for the next one
- try to capture corner cases
- write test cases for each requirement (for each user story)

GOOD TEST PRACTICES

- A good test case is one that has a high probability of detecting an undiscovered defect, not one that shows that the program works correctly
- Favor “reproducible” testing
- Write test cases for valid as well as invalid input conditions.

IT'S NOT ALWAYS THE SOFTWARE...

- most software testing assumes that test cases are correct, specifications are correct, and we know what we want
- it often happens, however, that, a test may fail because
  - the test case is wrong
  - there is a bug in the OS or in the libraries, or even in the hardware

ARIANE 5

MARS POLAR LANDER

MARS CLIMATE ORBITER

THERAC 25

TYPES OF TESTING

BLACK AND WHITE

- **black-box testing** is a method of software testing that examines the functionality of an application (e.g. what the software does) without peering into its internal structures or workings. No need to have access to/knowledge of the source code.
- **white-box testing** tests internal code structures or workings of an application, as opposed to its functionality. The tester chooses inputs that cause the execution of specific paths through the code.

SOFTWARE UNDER TEST

- **Unit** testing: test a function, in isolation. Very often done by implementor. Google calls them small tests. When we do unit tests we have no hypothesis on how the function will be used.
- **Integration** testing: we take many functions (already tested) and test them in combination. We test essentially that the people writing the different modules made compatible assumptions on what other modules provide.
- **System** or validation testing: tests if the overall system meets its goals.

PURPOSE AND CONDITIONS OF TEST

- Functionality
- Performance/stress testing: test the system at the boundaries of expected usage conditions, or beyond.
- Security
- Usability
- Reliability (also via stress testing)
- **Acceptance**: A test conducted to determine if the requirements of a contract are met.
- **Regression**: running test cases on a new implementation of a component
- …

A/B TESTING

- Not “software testing” in the traditional sense
- A/B testing is a simple way to test changes to your page against the current design and determine which ones produce positive results. It is a method to validate that any new design or change to an element on your webpage is improving your conversion rate before you make that change to your site code. [Optimizely]
- and yes, there is a Udacity course on this as well

DESIGNING TEST CASES

TEST CASES

- The possible input space is often infinite. Impossible to test everything
  - Think about testing google search…
- We can’t test forever. testing is a tradeoff
- The key is designing a “good” set of test cases. Let’s see examples of that.

HOW USEFUL IS A TEST CASE?

enqueue (7)

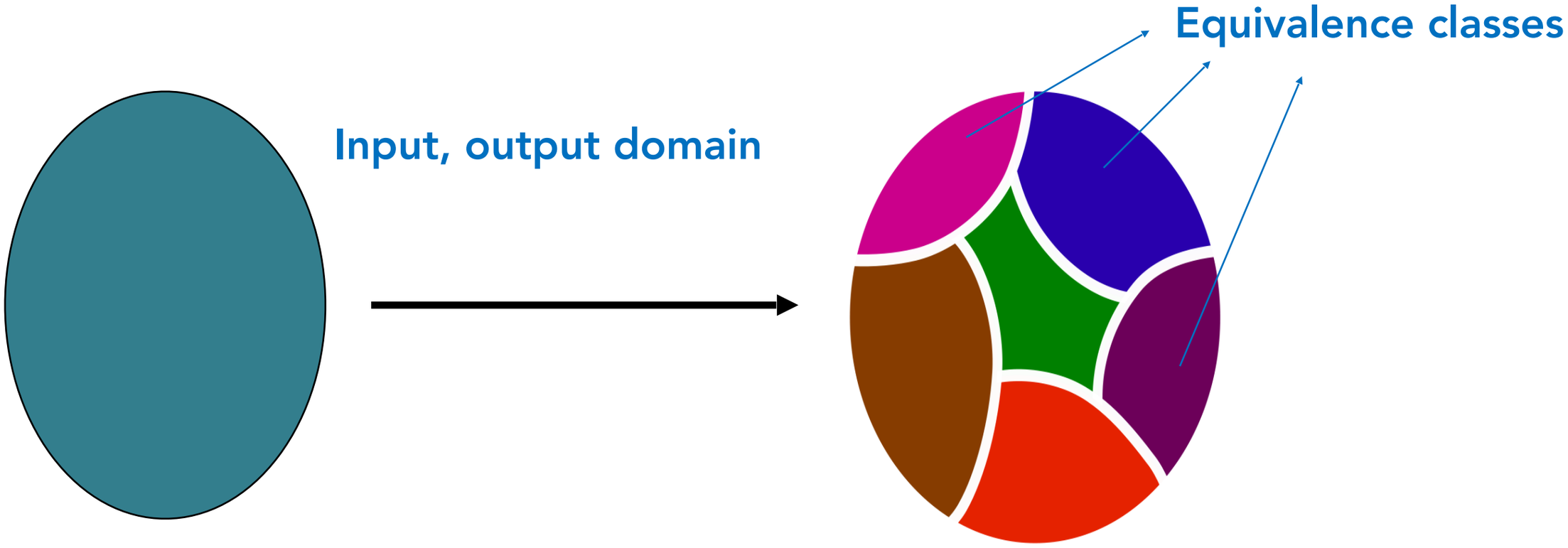
x=dequeue()

if x==7: success else fail

if we passed this test case, what else have we learned about our code, or have we learned anything? And your options are:

1. Our code passes this **one** test case,
2. Our code passes **any** test case where we replaced 7 with a different integer,
3. Our code passes **many** test cases where we replaced 7 with a different integer.

EQUIVALENCE PARTITIONING



Black-box : Equivalence Partitioning

- Divide all possible inputs into classes (partitions) such that
  - There is a finite number of input equivalence classes
  - You may reasonably assume that
    - the program behaves analogously for inputs in the same class
    - a test with a representative value from a class is sufficient
    - if representative detects fault then other class members will detect the same fault

Example : Equivalence Partitioning

- Test a function for calculation of absolute value of an integer
- Equivalence classes :

Condition	*Valid eq. classes*	*Invalid eq.* Classes
nr of inputs	1	0, > 1
Input type	integer	non-integer
particular *abs*	< 0, >= 0	

- Test cases (input only)

  x = -10, *x* = 100

  x = “XYZ”, x = - *x* = 10 20

CISS

Example : Equivalence Partitioning

- Test a program that computes the sum of the first *value* integers as long as this sum is less than *maxint.* Otherwise an error should be reported. If *value* is negative, then it takes the absolute value
- Formally:

  Given integer inputs *maxint* and *value* compute *result* :

$$result = \sum_{K=0}^{|value|} k \quad \text{if this <= } maxint, \quad \text{error otherwise}$$

CISS

Example : Equivalence Partitioning

- Equivalence classes :

Condition	*Valid eq. classes*	*Invalid eq. classes*
Nr of inputs	2	< 2, > 2
Type of input	int int	int no-int, no-int int
Abs(*value*)	*value* < 0, *value* ≥ 0	
maxint	$\sum k \leq$ *maxint*, $\sum k >$ *maxint*	

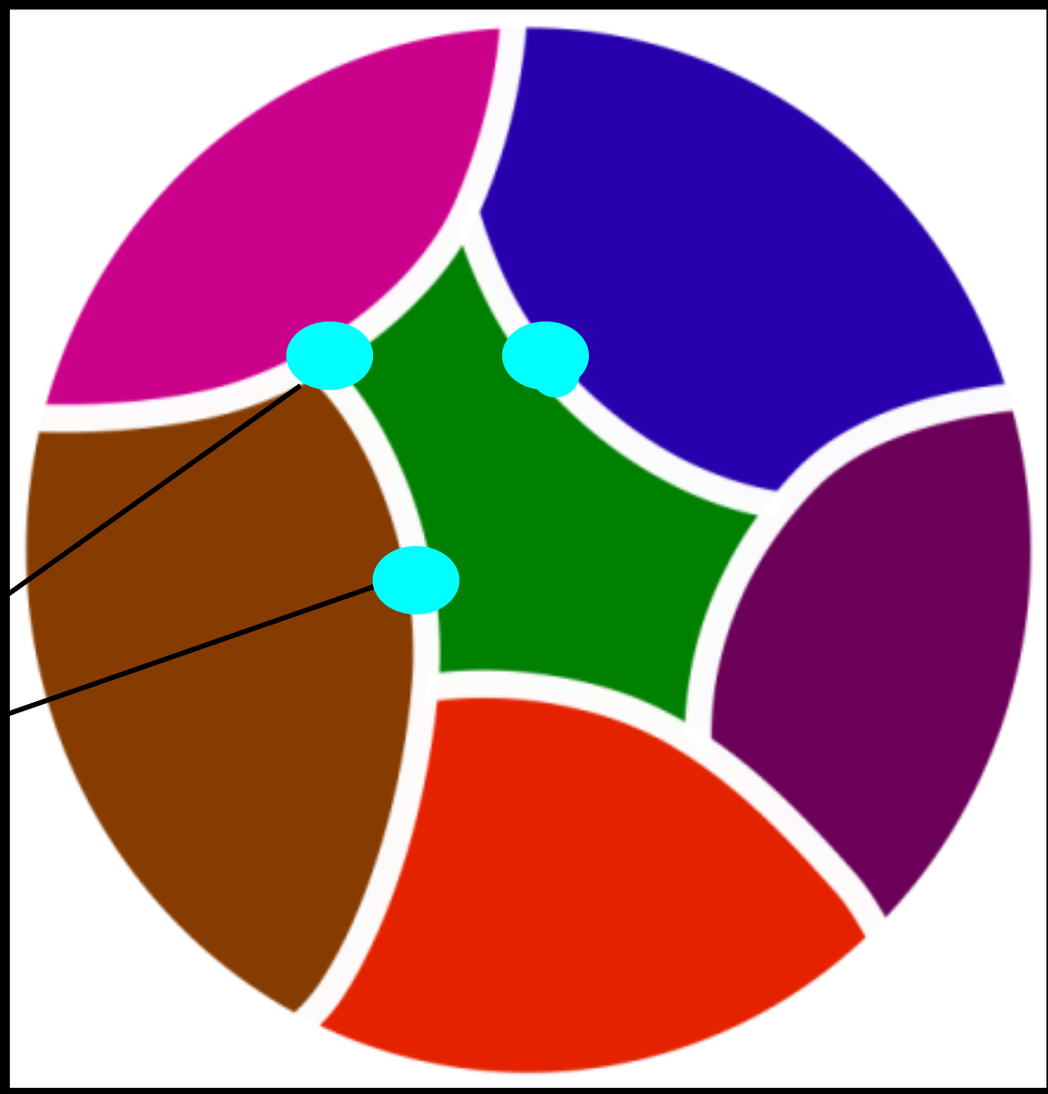
- Test Cases :

	maxint	*value*	*result*
Valid	100	10	55
	100	-10	55
	10	10	error
Invalid	10	-	error
	10 20	30	error
	"XYZ"	10	error

CISS

BOUNDARIES

Boundary values



Black-box : Boundary Value Analysis

- Based on experience / heuristics :
  - Testing *boundary conditions* of eq. classes is more effective
    i.e. values directly on, above, and beneath edges of eq. classes
  - Choose input boundary values as tests in input eq. classes
    instead of, or additional to arbitrary values
  - Choose also inputs that invoke *output boundary values*
    (values on the boundary of output classes)
  - Example strategy as extension of equivalence partitioning:
    - choose one (*n*) arbitrary value in each eq. class
    - choose values exactly on lower and upper boundaries of eq. class
    - choose values immediately below and above each boundary
      (if applicable)

CISS

Example : Boundary Value Analysis

- Test a function for calculation of absolute value of an integer
- Valid equivalence classes :

Condition	*Valid eq. classes*	*Invalid eq.* Classes
particular *abs*	< 0, >= 0	

- Test cases :

class x < 0, arbitrary value:	*x* = -10
class x >= 0, arbitrary value	x = 100
classes x < 0, x >= 0, on boundary :	x = 0
classes x < 0, x >= 0, below and above:	x = -1, x = 1

CISS

A SMALL EXAMPLE (MYERS)

- *The program reads three integer values 1..9 from an input dialog. The three values represent the lengths of the sides of a triangle. The program displays a message that states whether the triangle is scalene, isosceles, or equilateral.*

- try defining the test cases for this function

VALID INPUTS

- For input valuation we want to ensure that only sets of three positive non-zero integers are input. Thus we want to test that an error message is produced for any of the following:

- **Any** element of an input set is negative
- Any element of an input set is zero
- Any element of an input set is greater than 9
- Any element of an input set is an alphabetic letter
- Any element of an input set is a symbol
- There are less than three elements in the input set
- There are more than three elements in the input set

INVALID TRIANGLES

- Next we want to test against the risk of sets of inputs of three numbers 1-9 that cannot form triangles. If any of these is input a suitable error message should result
- The sum of any two numbers equals or is less than the third.

CORRECTNESS FOR VALID TRIANGLES

- Next we want to ensure that given a proper set of inputs for a triangle the correct type is determined by the program:

- If and only if all three digits of a valid input are the same the program displays that it has recognized an "equilateral" triangle
- If and only if any two of the digits of a valid input are the same the program displays that it has recognized an "isosceles" triangle
- If and only if each digit of a valid input is different the program displays that it has recognized a "scalene" triangle.

TEST CASES

- check out http://www.testingeducation.org/conference/wtst3_collard5.pdf

Tests	Inputs	Expected Results
Any element of an input set is negative	(-1,1,1), (1,-1,1), (1,1,-1)	Error Message
Any element of an input set is zero	(0,1,1), (1,0,1), (1,1,0), (1,0,0), (0,1,0), (0,0,1), (0,0,0)	Error Message
Any element of an input set is greater than 10	(11,1,1), (1,11,1), (1,1,11)	Error Message
Any element of an input set is an alphabetic letter	(1,1,'L'), (1,'L',1), ('L',1,1)	Error Message
Any element of an input set is a character symbol	(1,1,'$'), ...'$', ('$',1,1)	Error Message
There are less than three elements of the input set	(1,2)	Error Message
There are more than three elements of the input set	(2,4,3,4)	Error Message
Any element of an input set is a combination of one of the invalid inputs	(1A, 3,4), (1a,'W',3), (2,4,-)	Error Message
The sum of two numbers equals the third in any of three permutations (a+b=c, b+c=a), (a+c=b)	(2,4,6), (4,3,2), (8,4,4)	Error Message
The sum of two sides is less than the third in any of three permutations of a	(3,3,6), (2,6,1), (7,3,3)	Error Message
If all three digits of a valid input are the same the program displays that it has recognized an "equilateral" triangle	(3,3,3)	Equilateral ✓
If any two of the digits of a valid input are the same the program displays that it has recognized an "isosceles" triangle	(3,3,4),(7,6,7),(5,5,6)	Isosceles ✓
If each digit of a valid input is different, the program displays that it has recognized a "scalene" triangle	(3,4,5), (5,6,4), (6,4,3)	Scalene ✓