Software Testing

Background

- Main objectives of a project: High Quality & High Productivity (Q&P)
- Quality has many dimensions
  - reliability, maintainability, interoperability etc.
- Reliability is perhaps the most important
- Reliability: The chances of software failing
- More defects => more chances of failure => lesser reliability
- Hence quality goal: **Have as few defects as possible in the delivered software!**

Faults & Failure

- **Failure:** A software failure occurs if the behavior of the s/w is different from expected/specified.

- **Fault:** cause of software failure
- Fault = bug = defect
- Failure implies presence of defects

- A defect has the potential to cause failure.
- Definition of a defect is environment, project specific

Role of Testing

- Identify defects remaining after the review processes!
- Reviews are human processes - can not catch all defects
- There will be requirement defects, design defects and coding defects in code
- **Testing:**
  - Detects defects
  - plays a critical role in ensuring quality.

Detecting defects in Testing

- During testing, a program is executed with a set of test cases
- Failure during testing => defects are present
- No failure => confidence grows, **but can not say "defects are absent"**
- Defects detected through failures
- To detect defects, must cause failures during testing

Test Oracle

- To check if a failure has occurred when executed with a test case, we need to know the correct behavior
- That is we need a test oracle, which is often a human
- Human oracle makes each test case expensive as someone has to check the correctness of its output

Common Test Oracles

- specifications and documentation,
- other products (for instance, an oracle for a software program might be a second program that uses a different algorithm to evaluate the same mathematical expression as the product under test)
- an *heuristic oracle* that provides approximate results or exact results for a set of a few test inputs,
- a *statistical oracle* that uses statistical characteristics,
- a *consistency oracle* that compares the results of one test execution to another for similarity,
- a *model-based oracle* that uses the same model to generate and verify system behavior,
- or a human being's judgment (i.e. does the program "seem" to the user to do the correct thing?).

Role of Test cases

- Ideally would like the following for test cases
  - No failure implies “no defects” or “high quality”
  - If defects present, then some test case causes a failure
- Psychology of testing is important
  - should be to ‘reveal’ defects(not to show that it works!)
  - test cases must be “destructive”
- Role of test cases is clearly very critical
- Only if test cases are “good”, the confidence increases after testing

Test case design

- During test planning, have to design a set of test cases that will detect defects present
- Some criteria needed to guide test case selection
- Two approaches to design test cases
  - functional or black box
  - structural or white box
- Both are complimentary; we discuss a few approaches/criteria for both

Black Box testing

- Software tested to be treated as a block box
- Specification for the black box is given
- The expected behavior of the system is used to design test cases
- Test cases are determined solely from specification.
- Internal structure of code **not** used for test case design

Black Box Testing…

- Most thorough functional testing - exhaustive testing
  - Software is designed to work for an input space
  - Test the software with all elements in the input space
- Infeasible - too high a cost
- Need better method for selecting test cases
- Different approaches have been proposed

Equivalence Class partitioning

- Divide the input space into equivalent classes
- If the software works for a test case from a class the it is likely to work for all
- Can reduce the set of test cases if such equivalent classes can be identified
- Approximate it by identifying classes for which different behavior is specified

Equivalence Class Examples

In a computer store, the computer item can have a quantity between -500 to +500. What are the equivalence classes?

Answer: Valid class: -500 <= QTY <= +500
Invalid class: QTY > +500
Invalid class: QTY < -500

Equivalence Class Examples

Account code can be 500 to 1000 or 0 to 499 or 2000 (the field type is integer). What are the equivalence classes?

Answer:

Valid class: 0 <= account <= 499

Valid class: 500 <= account <= 1000

Valid class: 2000 <= account <= 2000

Invalid class: account < 0

Invalid class: 1000 < account < 2000

Invalid class: account > 2000

Equivalence class partitioning…

- Rationale: specification requires same behavior for elements in a class
- Software likely to be constructed such that it either fails for all or for none.
- E.g. if a function was not designed for negative numbers then it will fail for all the negative numbers
- For robustness, should form equivalent classes for invalid inputs also

Equivalent class partitioning..

- Every condition specified as input is an equivalent class
- Define invalid equivalent classes also
- E.g. range 0< value<Max specified
  - one range is the valid class
  - input < 0 is an invalid class
  - input > max is an invalid class
- Whenever that entire range may not be treated uniformly - split into classes

Equivalence class…

- Once eq classes selected for each of the inputs, test cases have to be selected
  - Select each test case covering as many valid equivalence classes as possible
  - Or, have a test case that covers at most one valid class for each input
  - Plus a separate test case for each invalid class

Example

- Consider a program that takes 2 inputs – a string **s** and an integer **n**
- Program determines **n** most frequent characters
- Tester believes that programmer may deal with diff types of chars separately

- Describe a valid and invalid equivalence classes

Example..

Input	Valid Eq Class	Invalid Eq class
S	1: Contains numbers 2: Lower case letters 3: upper case letters 4: special chars 5: str len between 0-N(max)	1: non-ascii char 2: str len > N
N	6: Int in valid range	3: Int out of range

Example…

- Test cases (i.e. s , n) with first method
  - **s** : str of len < N with lower case, upper case, numbers, and special chars, and n=5
  - Plus test cases for each of the invalid eq classes
  - Total test cases: 1 valid+3 invalid= 4 total
- With the second approach
  - A separate str for each type of char (i.e. a str of numbers, one of lower case, …) + invalid cases
  - Total test cases will be 6 + 3 = 9

Boundary value analysis

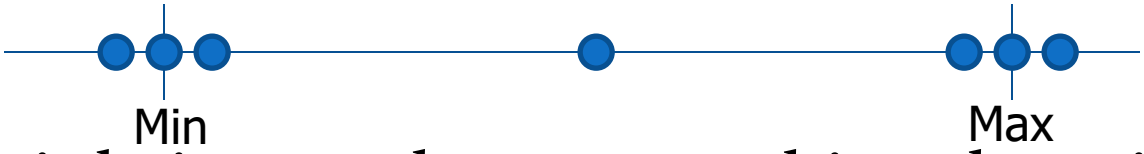
- Programs often fail on special values
- These values often lie on boundary of equivalence classes
- Test cases that have boundary values have *high yield*
- These are also called *extreme cases*
- A BV test case is a set of input data that lies on the edge of a eq class of input/output

Boundary value analysis (cont)...

- For each equivalence class
  - choose values on the edges of the class
  - choose values just outside the edges
- E.g. if 0 <= x <= 1.0
  - 0.0 , 1.0 are edges inside
  - -0.1,1.1 are just outside
- E.g. a bounded list - have a null list , a maximum value list
- Consider outputs also and have test cases generate outputs on the boundary

Boundary Value Analysis

- In BVA we determine the value of vars that should be used
- If input is a defined range, then there are 6 boundary values plus 1 normal value (tot: 7)



- If multiple inputs, how to combine them into test cases; two strategies possible
  - Try all possible combination of BV of diff variables, with n vars this will have 7^n test cases!
  - Select BV for one var; have other vars at normal values + 1 of all normal values

BVA.. (test cases for two vars – x and y)