Testing: are we building the product right?

Advanced Programming

Problem

- The class Dates.java compiles, but the output is not that we expected.
- Q: How do we ensure that the class Dates works right?
- A: We test its functionalities and its logic
- There are different ways to do it

Definition of Testing

- **A process to ensure that a product meets its specification (how it works)**

Definition of Testing

- In mathematics, we do not prove a theorem by testing
  We use a formal proof
  - It would be impossible to prove Pythagoras' theorem by testing all possible values
- Why don't we do the same in software?

Testing and formal methods

- This was what has been attempted with formal methods
  - We cannot prove that in general software meets its requirements
- **Halting problem**: given a program and an input, it is not decidable whether the program will eventually halt when run with that input, or will run forever

Good processes

- Exhaustive testing is not feasible:
  - if a failure is detected then the software is a failure software,
  - but this does not imply that if no failure has been detected the software is correct

Good processes

- Measuring and evaluating a software process may lead to the only notion of "good" process we can define:
  - a "good" process is correlated with the probability that no failures occur in a given interval of time

  -> Software reliability

Who tests the software better??

developer

Understands the system
but, will test "gently"
and, is driven by "delivery"

independent tester

Must learn about the system, but, will attempt to break it and, is driven by quality

Software testing: two perspectives

- Verification
  - "How" – the process of building
  - **"Are we building the product right?"**
- Validation
  - Ensuring that the product meets the customer's requirements
  - "What" – the product itself
  - **"Are we building the right product?"**

Software testing: two perspectives

- Verification:
  - A music player plays (it does play) the music when I press "Play"
- Validation:
  - A music player plays a music (it does not show a video) when I press "Play"

Strategies of testing

- White box testing (mainly Verification)
  - Tester focuses on the internal structure of the code
  - We use JUnit and TDD
- Black box testing
  - Tester has no access to code
  - We use acceptance tests

Problems in testing

- The number of possible inputs is large:
  - Black Box testing is hard
- The structure of the code is very complex
  - White Box testing is hard

Levels of testing

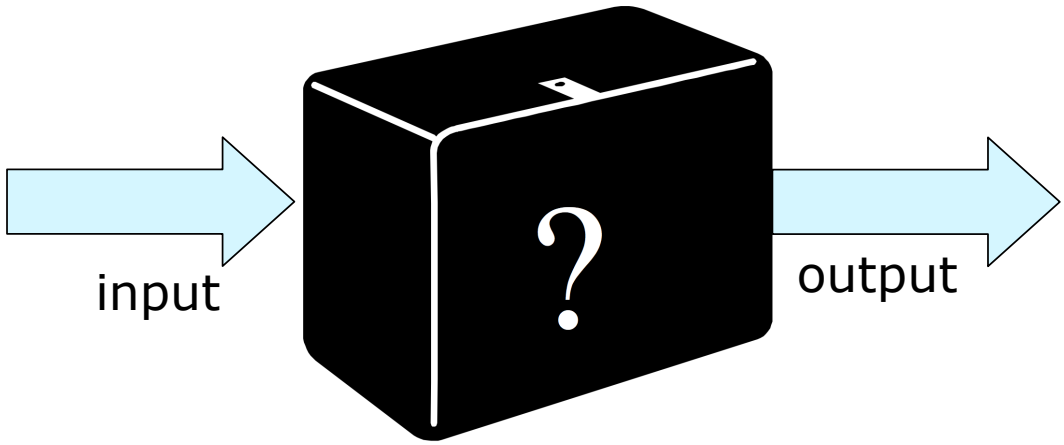
Testing Level	Tests Based Upon	Kind of Testing
Unit	Low-Level Design Actual Code Structure	White Box
Integration	Low-Level Design High-Level Design Smooth components integration	White Box Black Box
Functional and System	High-Level Design Requirements Analysis Functional: testing specific functionality System: testing in different environment	Black Box
Acceptance	Requirements Performed by customers	Black Box
Regression	Change Documentation High-Level Design Spot check throughout testing cycles	White Box Black Box

Test Case

- A test case is a set of conditions under which a piece of software works
  - A **good** test case has a **high probability** of finding an **as-yet** undiscovered error
  - A **successful** test is one that **uncovers** an **as-yet** undiscovered error
- Can be used TCs with WBT and BBT
- To design a test case we first need to reflect on its input and expected output

Black box testing

Black Box testing



Black Box testing

- **Aka: Functional testing or behavioural testing**
- Attempts to find 5 types of errors:
  - Incorrect or missing functions
  - Interface errors
  - Errors in data structures used by interfaces
  - Behaviour or performance errors
  - Initialisation and termination errors

Test case for BBT

- A test case consists of input-output paths
- Two types
  - Success paths - no error conditions in test cases
  - Failure paths - error conditions added in test cases

Exercise

- Write a test case for an enrolment service of the student information system that you use
  - isolate the functionality
  - identify all possible outputs
  - identify all possible inputs
  - identify boundaries of input (if any)
  - associate input and output (success paths and failure paths)

Example: log-in

» **Input: user name and password**

» **Output: not/accessing the system**

- Check actual behaviour against “access successful”
  - If credentials are correct the access the student enters the system
- Then check the actual behaviour for “access not permitted”
  - If credentials are wrong or do not correspond to existing user, student cannot enter the system
- To test the reason of failure, for example, whether the user is already logged in or the login session is expired we need WBT

Create a test case

- Isolate a functionality (e.g., log-in)
- Identify the **input variables**
- For each input variable describe the range of values and divide them into classes of **expected output** (equivalence partitioning)
- Select representatives for the classes or generate random values for the input variables
- Combine the values you chose for all the variables and define a test case

Example

logInTest		
Input	Description	Output (behaviors)
username= initials of name+surname password= characters, alpha numeric, bounded length (8) environmental input (DB) status= student	Preconditions: the user has accessed to the general site of the university The user introduces username and password Postcondition: the user is logged in	logged in not logged in

Exercise

- Select representatives for the classes or generate random values for the input variables
- Write the test cases

Equivalence partitioning

- The idea is to partition the input into classes that give equivalent output
  - To identify the equivalence classes
  - To select an input from within the equivalence class and a set from outside

Three types of input

- Input: certain value(s)
  - Valid class for each valid value and several invalid classes
- Input: a range of values
  - Two invalid classes for both ends and one valid class for the middle
- Input: membership of a set or a Boolean
  - One valid class and one invalid class

Examples

- Input: certain value(s)
  - f(n)=1/cos(π/2 *n) , n = natural number is not defined for odd n
- Input: a range of values
  - f(x)= sqrt(1-x^2)
- Input: membership of a set or a Boolean
  - xϵ{1,2,3}

Boundary value testing

- Example:
  - Range a..b
    - test cases: a, b, just above a, just below b
  - Number of values:
    - test cases: max, min, just below min, just above max
- Boundaries of externally visible data structures shall be checked (e.g. arrays)

Black box test case automation

- FitNess tool http://www.fitnesse.org/
- You can find the Eclipse plug-in