CODE TESTING

Principles and Alternatives

Testing - Basics
- goal - find errors
  - focus is the source code (executable system)
  - test team wants to achieve an appropriate level of confidence in the product
- style - run test-cases
  - input: specification and source code
  - action: derive test-cases & evaluate results
  - have enough test-cases been executed?

How “complete” is Testing?
- If testing includes all possible input combinations, it is equivalent to a proof.
- In all other cases, testing establishes only a measure of confidence, a belief, that the systems works according to specification and within quality parameters.

The Art of Testing
- Find a good set of test-cases
  - cover all aspects of the system
  - derive test-cases in a structured way
- Know when to stop testing
  - trade-off between completeness & cost
  - “measure” relative completeness

When to Stop
- “completeness”:
  - You have established test-cases which cover all aspects in sufficient quantity
  - You have executed all test-cases and achieved valid results
- “test the test”
  - e.g. Monte Carlo method

Test-Cases - Basics
- test-cases are derived during all phases of the development cycle
- determine expected results before running a test-case
- never throw away test-cases & results
- look for errors where you found errors before
- allocate time and money
Types of Test-Cases

- derive test-cases for
  - valid, expected inputs
  - invalid unexpected inputs (garbage)
  - test, if the system does less than specified
  - test, if the system does more than specified

Module Testing

- part of the implementation phase
- establish confidence (correctness) in single, isolated module
- test module's functionality
- test module's interface from the perspective of the module (from the internal perspective)
- Two major forms:
  - black box & white box testing

Integration Testing

- major task of the testing phase
- establish confidence (correctness) in the complete, assembled system
- test the system structure (design)
- focus on control, communication, interfaces, performance (other system qualities)
- use of stubs, test-beds, data generators

Black Box Testing - The Concept

- module level
- no knowledge of actual implementation
  - "black box"
- driven by the module's external appearance
  - based on specification & interface
Black Box Testing - Discussion

• advantages:
  – finds missing functionality
  – independent from code-size and complexity (?)
  – finds some coding errors (by chance)

• constraints:
  – no systematic search for low level errors
  – specification errors not found

Black Box Testing - Handling

• partition test-cases into functional groups
• follow the suggestions for “types of test-cases”
  – boundary values
  – null values
  – extremes
• generate additional random test-cases

White Box Testing - The Concept

• module level
• focus on structure of the actual implementation (source code)
  – “white box”
• does not take into account the specification of the module’s external behaviour

White Box Testing - Discussion

• advantages:
  – finds errors on the code level
  – typically based on a very systematic approach, covering the complete internal module structure

• constraints:
  – does not find missing / additional functionality
  – does not really check the interface
  – difficult for large and complex modules

White box and black box testing complement each other.

Always use both forms of module testing for a complete test.

End of Section 12c