UNICAM
Università di Camerino

XVIII. Software Testing

Laurea Triennale in Informatica – Corso di Ingegneria del Software I – A.A. 2006/2007
Andrea Polini

Objective

- General discussion on Testing
- Testing Phases
- Approaches to testing
- Structural testing
- Functional testing
- Testing non functional behaviour
- Adequacy

What is testing?

- Software testing consist of the **dynamic verification of the behavior** of a program on a **finite set of test cases**, suitably selected from the usually **infinite execution domain**, against the specified expected behavior”

[Antonia Bertolino – SWEBOK]

Implied tasks:

designing the test cases

executing the software with those test cases

examining the produced results

Error/Fault/Failure

- **Error**
  - Introduction of a wrong behaviour in the code
- **Fault**
  - The emergence of a wrong condition within the system caused by the error
- **Failure**
  - The observation of a wrong situation in the system behaviour

Interaction Constraints

- Testing is striclty related to two concepts:
  - **What we can control – e.g. Object interfaces**
  - **What we can observe – e.g. Return value**
- Platforms can be extended with support for testing...increase control and observability
- Certainly investments should be directed to observations for which we can provide an “oracle”
- Testability: what is it? How can be improved?

The Oracle Problem

- **Is it the result provided by the System Under Test (SUT) correct or not?**
- Manual derivation of Oracles
- Usage of models to define oracles
  - **In general weaker relation among the inputs and the outputs permit to highlight errors**
- oracles can be derived from history of executions of **available systems** providing same functionality
- Deriving an oracle is a really expensive task and should not consist in the implementation of another system
- Relevant theoretical results limit the possibility of automatic derivation of oracles

Testing Strategy

- **Testing cannot reveal the absence of errors and can only show their presence**
  - Develop strategy that maximise the chance of discovering bugs
- Testing as fishing
- **Systematic** exploration of a system characteristics
- Testing should be an adaptive phase
  - Different technology show different problems
    - OO - Late binding, inheritance
    - CBS – software reused in different context and produced by third parties
  - Testing strategy should exploit characteristics of a technology

Testing phases

- Unit Testing
- Integration Testing
- System Testing
- Regression Testing

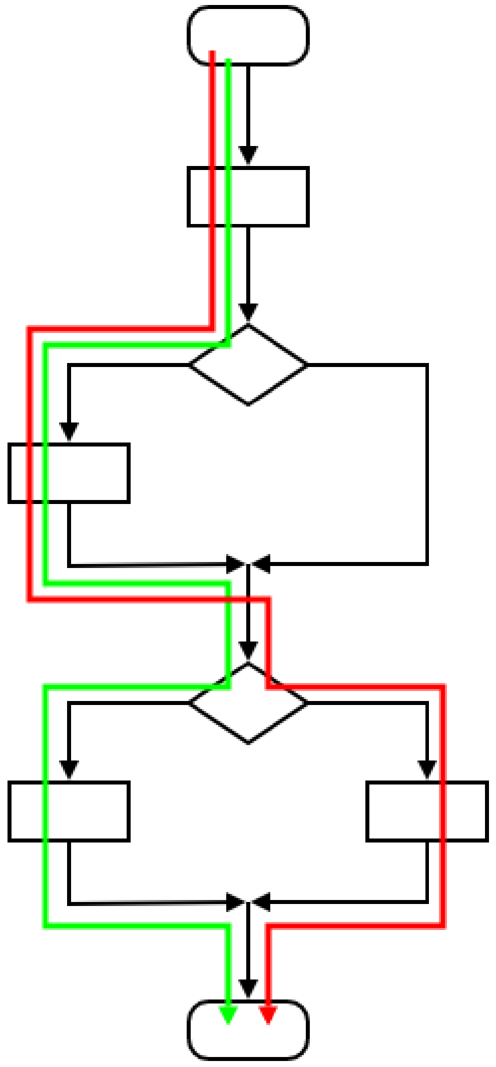
Structural Testing

- Also known as **white-box** testing
- Is based on the assumption that only when executed an error can manifest itself. i.e. a wrong statement must be executed
- Requires the availability of the source code
- Different code characteristics can be considered:
  - Data
    - Definition and Usage
  - Statements
    - Conditional
- Increasing level of complexity

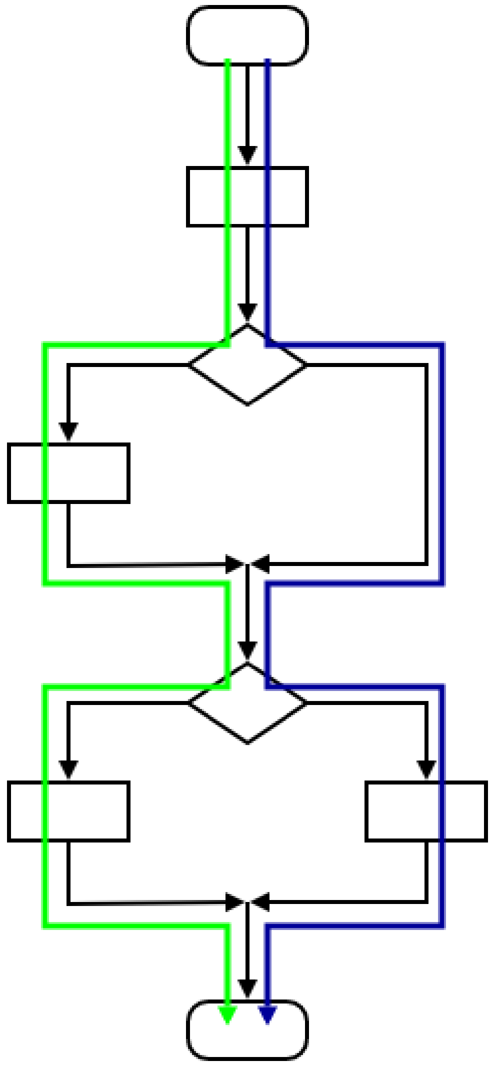
Structural Testing

- All Statements Coverage:
  - All statements should be executed at least once
- All Branches Coverage
  - All branches should be executed at least once
- All Paths Coverage
  - All paths should be executed at least once

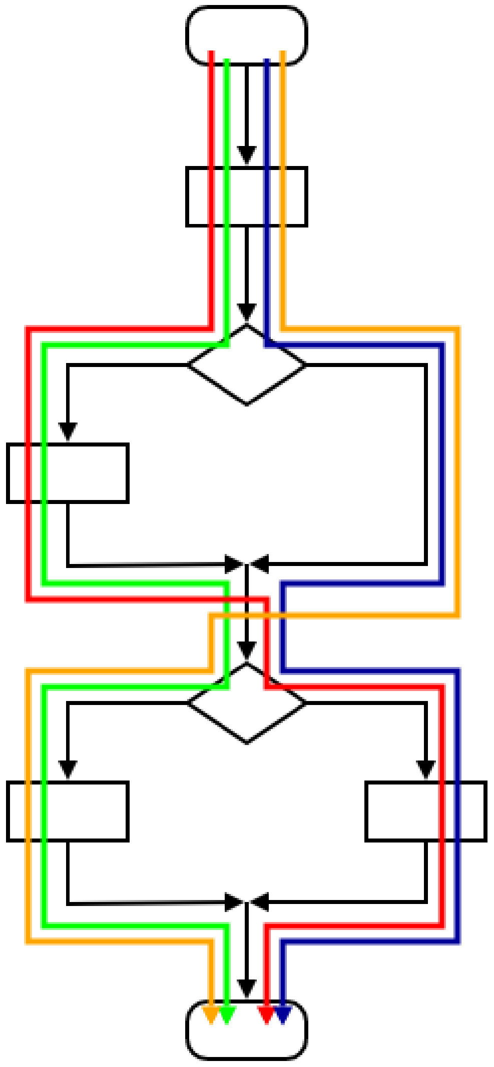
Coverage examples



Statement Coverage
(aka line coverage)



Branch Coverage
(aka condition coverage)



Path Coverage

Functional Testing

- Also known as Black-box testing
- **Assume the availability of a model for the system (the specification)**
- Requirements based testing
- Data Models
  - Several methodologies to derive tests:
    Boundary conditions, **Category Partitions**
- Behavioural Models
  - Availability of state machines and definition of invocation sequences to observe that traces
  - Coverage on behavioural models

Automatic support for the testing phase

- Availability of Models permits the development of tools for automatic derivation of test cases
- This permit to management to save a lot o money
- **Model Based Testing**

Testing non functional behaviour

- Inspection cannot be used to show lack of QoS
- Testing can help the developer to derive QoS correct systems
- **Platforms reproducing the environment are required**
- **Simulator of real usage are required (workload generator)**

Adeguacy Criteria and Assesment

- Traditional notion of test adequacy: A *test adequacy criterion* is a **systematic method** used to determine whether a test suite provides an **“adequate” amount of testing** for a program [component] under test
- When to stop testing:
  - related to required level of confidence
  - related to budget
- How can be defined?
- A fault based approach: fault injection

JUnit

- “*Every programmer knows they should write tests for their code. Few do. The universal response to "Why not?" is "I'm in too much of a hurry." This quickly becomes a vicious cycle- the more pressure you feel, the fewer tests you write. The fewer tests you write, the less productive you are and the less stable your code becomes. The less productive and accurate you are, the more pressure you feel.*”
- “***code a little, test a little, code a little, test a little***”
- Provide a framework to define, group and (re-)execute test cases