Software Systems Verification and Validation

Lecture 03 - White-box testing

Lect. dr. Andreea Vescan

Babeș-Bolyai University
Cluj-Napoca

2015-2016

Test case design

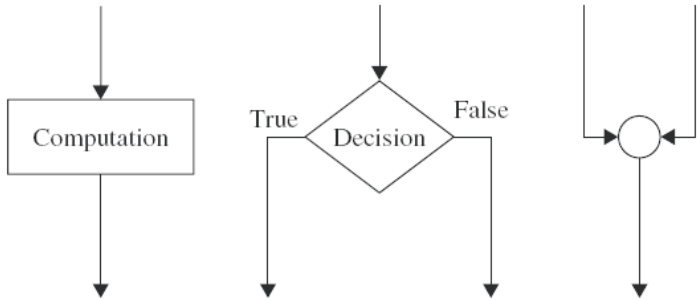
- Testing is the process of executing a program with the intent of finding errors. [Mye04]
- Achieve this goal:
  - Test cases
  - Test case: " A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement." [IEE90]
  - Black-box testing
  - White-box testing

Selecting test cases for White-box testing [Fre10]

- Based on control flow, i.e. using Control Flow Graph
- Based on data flow

Flow Graph Notation [Mye04] (chapter 3), citeNai08 (chapter 4), [PY08] (chapter 12), [Pat05] (chapter 6, 7)

- A Control Flow Graph (CFG) is a graphical representation of a program unit.
- A CFG has exactly one entry node and exactly one exit node.



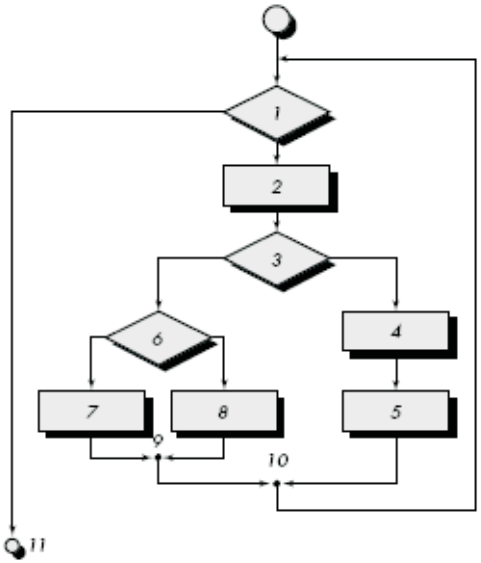
- Three symbols are used to construct a CFG
- nodes - sequential statements, decision and looping predicates
- edges - represent transfer of control
- Path in the CFG - is represented as a sequence of computation and decision nodes from the entry node to the exit node.
- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

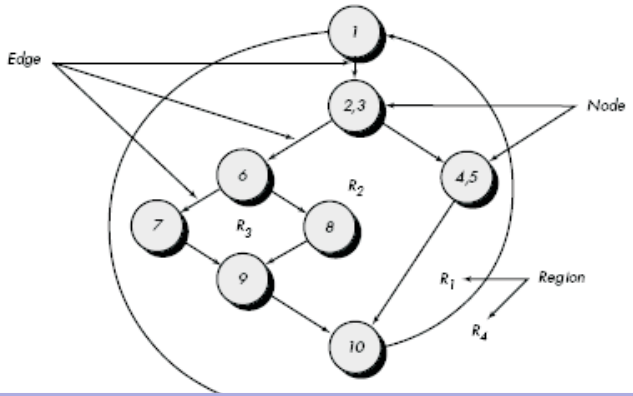
Cyclomatic complexity

- Cyclomatic complexity
  - The number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
  - CC = The number of regions of the flow graph.
  - CC = E - N + 2, where E - #edges, N - #nodes.
  - CC = P+1, where P - #predicate_nodes

Cyclomatic complexity - example

- CC
  - CC = four regions = 4.
  - CC = 11 edges - 9 nodes + 2 = 4.
  - CC = 3 predicate nodes + 1 = 4.
- A set of independent paths:
  - path 1: 1-11.
  - path 2: 1-2-3-4-5-10-1-11.
  - path 3: 1-2-3-6-8-9-10-1-11.
  - path 4: 1-2-3-6-7-9-10-1-11.





All-Path coverage criterion

- Testing every execution of the program, i.e. testing all paths of the CFG.
- It is difficult to achieve in practice $\Rightarrow$ reduced number of paths.
- Structural criteria are applied based on statements, edges and paths.

CFG coverage

- Select the minimum number of test cases such that we achieve:
  1. statement coverage
  2. branch/decision coverage
  3. condition coverage
  4. decision-condition coverage
  5. multiple-condition coverage
  6. path coverage
  7. loop coverage

1. Statement coverage (sc)

- Goal: to execute every statement in the program at least once.
- Complete statement coverage is *the weakest coverage criterion* in program testing.
  - Any test suite that achieves less than statement coverage for new software is considered to be unacceptable.
- We need to select a few feasible paths to cover all the nodes of the CFG. The selecting paths rules are:
  1. Select shorts paths.
  2. Select paths of increasingly longer length. Unfold a loop several times if there is a need.
  3. Select arbitrarily long, "complex" paths

2. Decision (branch) coverage (dc)

- A branch is an ongoing edge from a node.
  - All the rectangle nodes have at most one ongoing branch, except the exit node.
  - All the diamond nodes have two outgoing branches.
- Covering a branch means selecting a path that includes the branch.
- Complete branch coverage means selecting a number of paths such that every branch is at least one path.
  - selecting enough number of paths such that every condition evaluates to true at least once and to false at least once.

Decision coverage - issues

- Remark: dc⇒ sc
  - Why? Since every statement is on one subpath emanating from a branch statement or from the entry point of the program, every statement must be executed if every branch direction is executed.
  - Exceptions:
    - Programs with no decisions.
    - Programs with multiple entry points. A given statement might be executed only if the program is entered at a particular entry point.
- A branch with multiple conditions → some decisions may remain uncovered.

3. Condition coverage (cc)

- Goal: to write enough test cases to ensure that each condition in a decision takes on all possible outcomes at least once.
- cc $\Rightarrow$ dc (in general).
  - cc may cause (but does not always) every individual condition in a decision to be executed with both outcomes.
- Exceptions:
  - if $(A \&\& B) <$ *statement* $>$
  - cc $\Rightarrow$ TC_1 for A true, B false, and TC_2 for A false and B true
  - But the *statement* is not executed (dc for True is not covered!
  - $\Rightarrow$ there is a need for decision/condition coverage

4. Decision/condition coverage (dcc)

- Goal: requires sufficient test cases that:
  - each condition in a decision takes on all possible outcomes at least once;
  - each decision takes on all possible outcomes at least once;
  - each point of entry is invoked at least once.
- dc $\Rightarrow$ cc (in general)
- Exceptions:
  - When certain condition mask other conditions
  - Results of conditions in && and || expressions can mask or block the evaluation of other conditions (i.e. if an && condition is false then none of subsequent conditions in the expression need to be evaluated)
  - Thus, errors in logical expressions are not necessarily revealed by the condition-coverage and decision/condition coverage criteria

Hierarchy of strengths for sc, dc, cdc

- From weakest to strongest: sc, dc, cdc.
- The implication for this approach to test design is that the stronger the criterion, the more defects will be revealed by the tests.
- In most cases the stronger the coverage criterion, the larger the number of test cases that must be developed to ensure complete coverage.
- $\Rightarrow$ the tester must decide (based on the type of code, reliability requirements, resources available) which criterion to select!

5. Multiple condition coverage (mcc)

- Goal: write sufficient test cases that:
  - all possible combinations of condition outcomes in each decision, and
  - all points of entry are invoked at lest once.
- mcc $\Rightarrow$ dcc (in general)
- Remark: A set of test cases satisfying the multiple-condition criterion also satisfies the decision coverage, condition coverage, and decision/condition coverage criteria.

Minimum test criterion

- For programs containing only one condition per decision:
  - Test cases to evoke all outcomes of each decision at least once, and
  - Test cases to invoke each point of entry at least once, to ensure that all statements are executed at least once.
- For programs containing decisions having multiple conditions:
  - Test cases to evoke all possible combinations of condition outcomes in each decision, and
  - all points of entry to the program, at least once.

7. Loop coverage

- Simple loops - n is the maximum number of allowable passes through the loop:
  1. Skip the loop entirely.
  2. Only one pass through the loop.
  3. Two passes through the loop.
  4. m passes through the loop where $m < n$.
  5. n-1, n, n + 1 passes through the loop.
- Nested loops
  1. Start at the innermost loop. Set all other loops to minimum values.
  2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter .
  3. Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values.
  4. Continue until all loops have been tested

White-box testing

Advantages

- Code coverage
- Testing can be commenced at an earlier stage.
- Find the fault.

Disadvantages

- A skilled tester is needed to carry out this type of testing.
- No ambiguities in spec. may be found.
- After code is written.

Example - wbt

- Problem statement: Compute the number of appearances of the maximum value in an array of natural elements.
- Applied techniques:
  - Construction of the CFG.
  - Coverage: statements, conditions/decisions, paths, loops.
- See example files on SSVV lecture's homepage

Testlink

- Test management tools - Testlink. (Release 1.9.5)
- `http://www.scs.ubbcluj.ro/testlink/testlink-1.9.5/logir`
- Testlink Tutorial - SSVV lecture's homepage

Bibliografie I

[Fre10] M. Frentiu.
Verificarea și validarea sistemelor soft.
Presa Universitară Clujeană, 2010.

[IEE90] Ieee standard glossary of software engineering terminology.
https://standards.ieee.org/findstds/standard/610-1990.html, 1990.
Accessed: 2016-02-26.

[Mye04] G. Myers.
The Art of Software Testing, 2nd Edition.
John Wiley, 2004.

[Pat05] R. Patton.
Software Testing.
Sams Publishing, 2005.

[PY08] M. Pezzand and M. Young.
Software Testing and Analysis: Process, Principles and Techniques.
John Wiley and Sons, 2008.