UTCN
Computer Science Department
Software Design 2017

TESTING TECHNIQUES

==========

1. Theoretical Background

This section briefly presents the basic concepts regarding software testing. The provided information is based on [1].

Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems. ***Software testing*** consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior. The objectives of testing are the following:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as-yet-undiscovered error.
- A successful test is one that uncovers an as-yet-undiscovered error.

The attributes of a good test are the following:

- Has a high probability of finding an error.
- Is not redundant.
- Should be neither too simple nor too complex.

Software testing techniques can be classified into *white-box* and *black-box*.

1.1 White-box Testing

White-box testing uses the control structure of the procedural design to derive test cases that (*i*) guarantee that all independent paths within a module have been exercised at least once, and that (*ii*) exercise all logical decisions, all loops and internal data structures. Control structure testing is an example of white-box testing method which includes the following:

a) Condition testing

- Exercises the logical conditions in a program module.
- If a condition is incorrect, then at least one component of the condition is incorrect. Therefore, types of errors in a condition include the following:
    - Boolean operator error (incorrect/missing/extra Boolean operators).
    - Boolean variable error.

- Boolean parenthesis error.
- Relational operator error.
- Arithmetic expression error.

- Strategies:
  - *Branch testing* - for a compound condition C, the true and false branches of C and every simple condition in C need to be executed at least once.
  - *Domain testing* - for E1 <relational-operator> E2, 3 tests are required to make the value of E1 greater than, equal to, or less than that of E2.
  - *Branch and relational testing* - detects branch and relational operator errors in a condition if all Boolean variables and relational operators in the condition occur only once and have no common variables.

b) Loop testing

- Focuses exclusively on the validity of loop constructs (simple, concatenated, nested, unstructured).

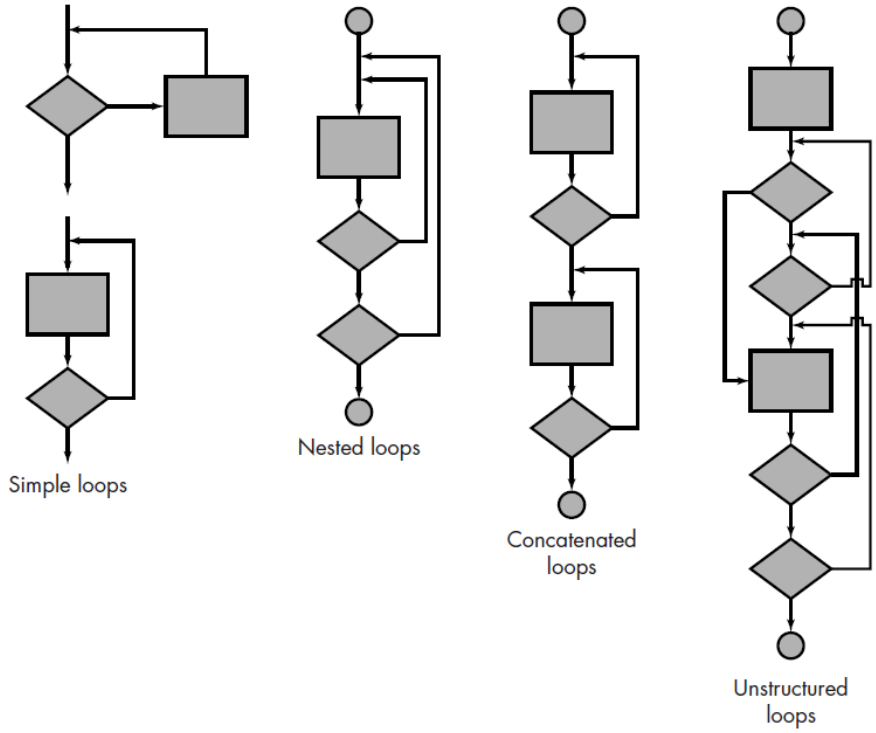

Figure 1. Types of loop constructs [1]

1.2 Black-box Testing

Black-box testing, also called behavioral testing, focuses on the functional requirements of the software and enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing is not an alternative to

white-box techniques; rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods. Black-box testing attempts to find errors in the following categories: (1) incorrect or missing functions, (2) interface errors, (3) errors in data structures or external data base access, (4) behavior or performance errors, and (5) initialization and termination errors. Black-box testing includes methods such as the following:

a) Equivalence partitioning

- Divides the input domain of a program into classes of data from which test cases can be derived.
- An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition. Equivalence classes may be defined according to the following guidelines:
    - If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
    - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
    - If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
    - If an input condition is Boolean, one valid and one invalid class are defined.

b) Boundary value analysis

- Complements equivalence partitioning by exercising bounding values. Rather than selecting any element of an equivalence class, the boundary value analysis leads to the selection of test cases at the "edges" of the class.

2. Task

Read and run the examples from the links below:

Java – JUnit: http://netbeans.org/kb/docs/java/junit-intro.html#Exercise_32.

C# - NUnit: http://www.dijksterhuis.org/setting-up-nunit-for-c-unit-testing-with-visual-studio-c-express-2008/

3. References

[1] R. Pressman, “Software Engineering – A Practitioner’s Approach”, McGraw Hill, 2005.