Test Design using Test Models

Introduction

Over the last 20 years, the test design process – how we generate test cases – has been defined and re-defined by several different bodies, including ISO and ISTQB. However, there has been no agreement on a common test design process, and there is confusion on the use of test conditions in these processes, as many practitioners are not sure where they fit into their way of generating tests.

This paper introduces a simplified test design process that discards the use of test conditions for the simpler concept of a 'test model', from which test coverage items, and subsequently test cases, can be easily derived. This test design process is supported by a full set of definitions and is aligned with the existing test planning process in the ISO/IEC/IEEE 29119 set of standards. To show how the process works, examples of it working with the most popular test design techniques are provided, along with descriptions of how it supports risk-based testing by the tester deriving the test cases.

Test Design Process

The new test design process is shown in Figure 1. This process is expected to be performed by anyone generating test cases for subsequent dynamic test execution.

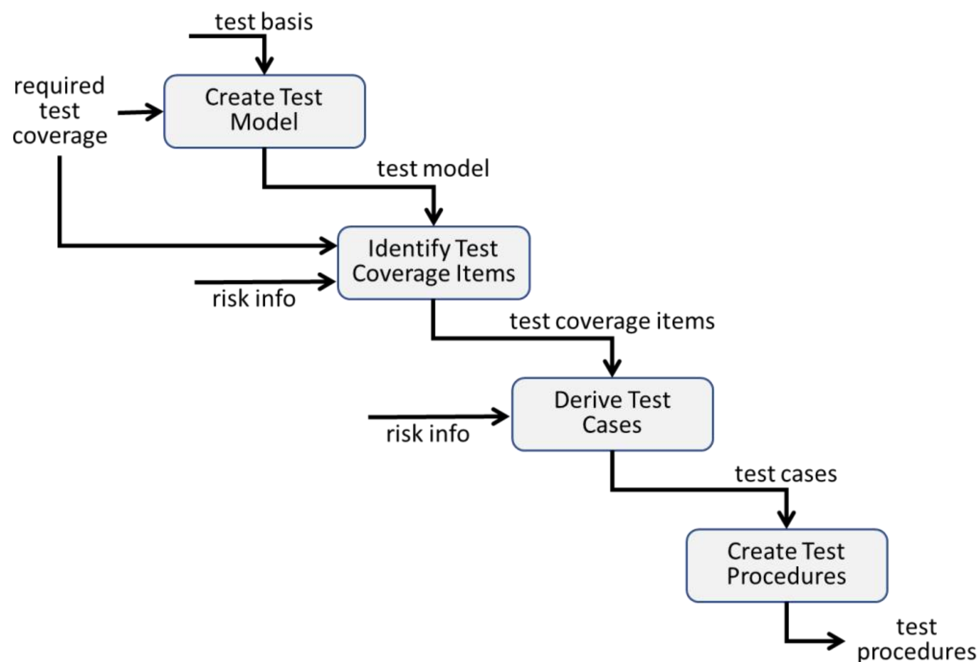

Figure 1: Test Design Process

The four activities that make up the test process are described below. If you struggle to follow the descriptions or prefer examples, these are provided in the next section. Definitions of terms are provided at the end of the paper.

 ©Stuart Reid, 2018

Create Test Model

1. **Understand the required behaviour of the test item from the test basis (including talking to stakeholders).**
    - *Its always a good idea to fully understand what you're testing. The test basis should also provide you with enough information to determine expected results, which you will need later when you derive the test cases. In many situations, the documentation of the test basis is lacking (e.g. incomplete or out-of-date), therefore it's always a good idea to talk to the relevant stakeholders who can tell you what the test item is really supposed to do.*
2. **Decide the notation of the test model based on the required test coverage.**
    - *We will later use the test model to identify test coverage items, and by exercising enough of these test coverage items, we will achieve the required test coverage – thus, we want the test model to be in a form that matches the test coverage items and the required test coverage. The required test coverage should be defined as part of the test completion criteria in the test strategy – this relationship can be seen in Figure 2.*
3. **Create the test model for the test item focused on the required test coverage.**
    - *Having determined the appropriate notation, we can create a test model for the test item that we can subsequently use to identify test coverage items. For instance, if the required test coverage is exercising all the transitions of a state transition diagram (and so the test coverage items are the transitions), then a suitable test model would be a state transition diagram.*

Identify Test Coverage Items

1. **Use risk information to identify the most important parts of the test model**
    - *Often, we won't have the time or resources to exercise all of the test model, which means we focus on the most important parts. The risk information used to determine importance may come from specifications (e.g. requirements may be labelled using the MoSCoW method for classification - Must have, Should have, Could have, and Won't have), or it may come from the experience of the tester, or it may come from talking to stakeholders.*
2. **Identify test coverage items to achieve required test coverage covering the most important parts of the test model**
    - *Starting with the most important parts, we identify test coverage items from the test model. For instance, if we are using decision table testing, we will identify the most important rules (test coverage items) from the decision table (test model). We continue to do this until we have what we believe are enough to achieve the required test coverage. Often this is set at 100% for the given measure and so we must exercise the whole model (e.g. all the identified boundaries between equivalence classes for boundary value analysis, or all the branches for branch testing).*

Derive Test Cases

1. **Derive test cases to exercise the identified test coverage items**

 ©Stuart Reid, 2018

- *The test coverage items tell us what we must cover with our tests, but they do not provide sufficient detail to run a test. Thus, we expand on the test coverage item (our rationale for this test) in the form of a test case (e.g. providing actual input values and values for expected results).*

2. **Use risk information to decide the number of test cases per test coverage item**

    - *When following a risk-based approach to the testing, we will not always limit ourselves to a single test case for each test coverage item. Sometimes we will know that a particular test coverage item is targeted at a part of the software that is far more important than another part. In such situations, we can decide to spend more effort on testing the more important parts, and we can do this by deriving more test cases to cover these important test coverage items. For instance, we could decide to exercise what we believe are important decision outcomes more often than other decision outcomes in the code when performing decision testing.*

Create Test Procedures

1. **Create test procedures from the derived test cases, based on constraints on the execution of the test cases and dependencies between the test cases**

    - *Typically, we don't run each test case separately (although we could if we wanted). Instead, we create test procedures (sometimes known as test scripts) that are sequences of related test cases. So, how do we decide the order for executing these test cases? Test cases often have preconditions and we can sometimes satisfy these preconditions by running another test case first. A simple example is where we have test cases to login, logout and register on the system. If we want to order these test cases into a test procedure, then we would normally order them so that we register first, then login and finally logout.*

The test design process shown in Figure 1 will typically be repeated for each different type of required test coverage included in the test completion criteria. For instance, if the test completion criteria for a test item included two forms of required test coverage - full boundary coverage <u>and</u> 100% statement coverage - then we would require the process to be followed twice.

We do not need to always follow this test design process in a completely sequential manner. We could, for instance, identify only some of the test coverage items and then derive test cases and test procedures for them (and even execute them) before we go back and identify the remaining test coverage items – and then perform the subsequent activities for these.

Figure 2 shows how the test management activity of test planning follows a risk-based approach to determine the test strategy (part of the test plan). The test completion criteria form part of this test strategy and typically include one or more types of required test coverage (e.g. full boundary coverage and 100% branch coverage), as well as other criteria, such as a maximum number of known unfixed critical defects. Some completion criteria may also include resource limits on the testing, such as the maximum amount of time or money that may be spent on the testing. The required test coverage for a test item is a major driver for the test design process, as this determines the form of

the test model and the test coverage items, as well as the minimum number of test coverage items (and test cases) that need to be identified and derived.

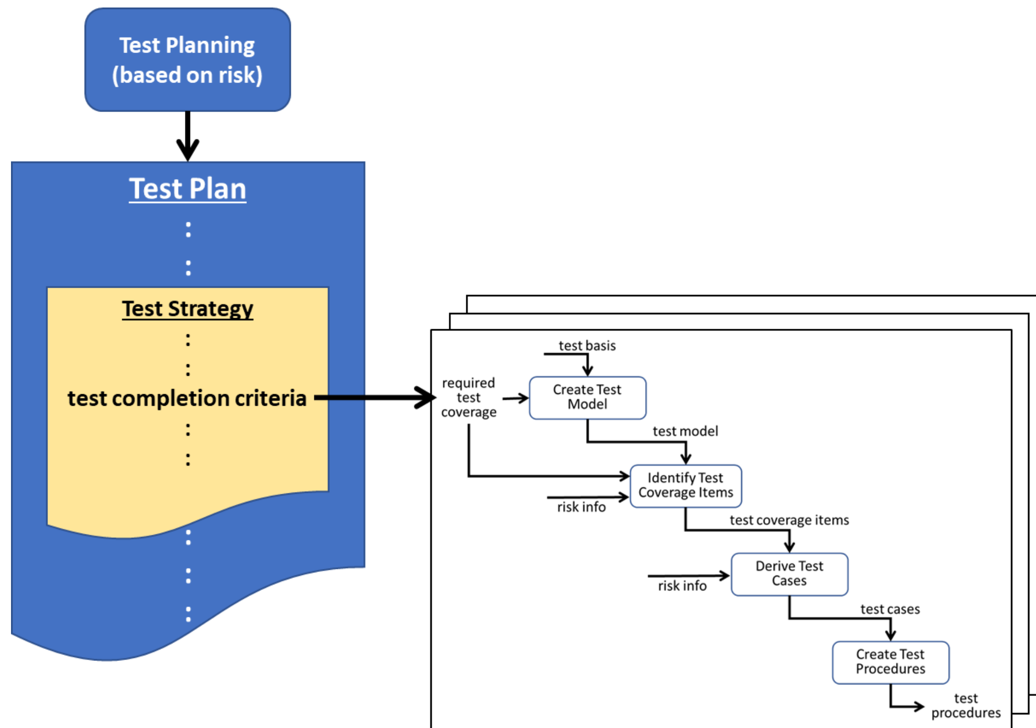

Figure 2: Test Planning and Test Design

A risk-based approach to test planning is typically used by the test manager or test lead to determine the test strategy, however, when performing the test design, the tester also perform risk-based testing at this lower level when they use risk information to decide how much of the test model to cover when identifying test coverage items and when they decide how many test cases to derive for each test coverage item.

We will see the use of risk at this lower level in some of the examples of applying the test design process with actual test design techniques next.

Application with different Test Design Techniques

We will next consider the most popular test design techniques and demonstrate how the test design process (see Figure 2) supports them, by describing the expected forms of the 'Required Test Coverage', the 'Test Model' and the 'Test Coverage Items' for each of these techniques.

Requirements-Based Testing

- Required Test Coverage
    - Coverage of specified requirements.

      Typically, the required test coverage for requirements-based testing is 'all specified requirements', although where requirements can be classified in terms of importance then sometimes lower coverage criteria may be set (see example below).

- Test Model
  - The set of 'atomic' requirements (requirements that can be individually tested).

    The generation of the set of individually testable requirements that make up the test model can be difficult when requirements are poorly-documented, and this activity may require extensive interaction with project stakeholders.
- Test Coverage Items
  - The test coverage items are the individual 'atomic' requirements considered important enough to test based on available risk information.

    Where there is available information on the relative importance of different requirements, it may be necessary to select only a subset as test coverage items to be exercised by the testing.

Example - Requirements-Based Testing

- Required Test Coverage
  - All high importance requirements shall be covered.
- Test Basis
  - The system shall do X when input speed does not exceed 50 km/h, and if the speed is higher than 120 km/h it shall do Y, otherwise it shall do Z. Valid input speeds are from 0 km/h to 220 km/h – invalid speeds will cause an error message. All speeds are provided as integers.
- Test Model (including importance based on known risks)
  - The system shall do X when input speed is between 0 km/h and 50 km/h inclusive (high importance).
  - The system shall do Y when input speed is higher than 120 km/h and no greater than 220 km/h (high importance).
  - The system shall do Z when input speed is higher than 50 km/h and no greater than 120 km/h (high importance).
  - The system shall generate an error message for invalid speeds outside the range from 0 km/h to 220 km/h (low importance).
- Test Coverage Items
  - The system shall do X when input speed is between 0 km/h and 50 km/h inclusive.
  - The system shall do Y when input speed is higher than 120 km/h h and no greater than 220 km/h.
  - The system shall do Z when input speed is higher than 50 km/h and no greater than 120 km/h.

 ©Stuart Reid, 2018

Note that only those requirements in the test model that are classified as 'high importance' are used to identify test coverage items (the low importance requirement will not be exercised by a test), due to the specified required test coverage in this example.

Equivalence Partitioning (EP)

- Required Test Coverage
    - Coverage of equivalence classes.

      Normally, all identified equivalence classes are expected to be exercised when performing equivalence partitioning.
- Test Model
    - The test model represents the different functionality within the test item (e.g. A, B and C in Figure 3), where each is expected to process a separate class of inputs in a similar fashion.

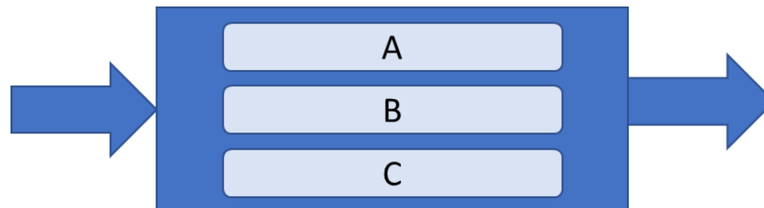

Figure 3: Representation of Equivalent Functionality (EP Test Model)

    - Functionality to deal with invalid inputs is also normally included in the test model (and different invalid input types may be dealt with separately, e.g. out of range inputs may require separate functionality from non-integer inputs).
    - The classes of inputs that cause specific functionality to run are known as equivalence classes. For instance, for the situation shown in Figure 3 there would typically be equivalence classes corresponding to A, B and C.
- Test Coverage Items
    - The test coverage items for EP are the equivalence classes.

Example - Equivalence Partitioning

- Required Test Coverage
    - All identified equivalence partitions shall be covered.
- Test Basis
    - The system shall do X when input speed does not exceed 50 km/h, and if the speed is higher than 120 km/h it shall do Y, otherwise it shall do Z. Valid input speeds are from 0 km/h to 220 km/h – invalid speeds will cause an error message. All speeds are provided as integers.
- Test Model
    - See Figure 4.

©Stuart Reid, 2018

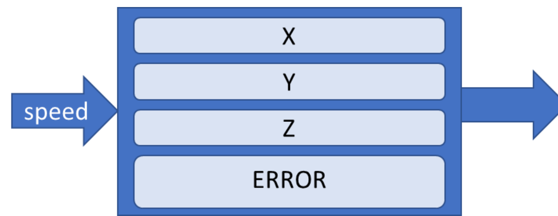

Figure 4: Example EP Test Model

- Test Coverage Items
  - 0 ≤ speed ≤ 50 (X)
  - 120 < speed ≤ 220 (Y)
  - 50 < speed ≤ 120 (Z)
  - speed < 0 or speed > 220 (ERROR)

The above format is typical for equivalent classes. Note that they are defined in terms of the input variable, speed, although the outputs of the test item are often used to identify the 'equivalent' functionality and thereby the corresponding input equivalence classes.

When subsequently deriving the test cases we could use risk information to decide that some of these equivalence classes are more important than others, and perhaps derive less test cases for those than for the other equivalence classes.

Boundary Value Analysis (BVA)

- Required Test Coverage
  - Coverage of identified boundaries.

    Normally, all identified boundaries are expected to be exercised when performing equivalence partitioning.
- Test Model
  - The test model (see Figure 5) represents the boundaries between the input equivalence classes identified as part of equivalence partitioning (see previous technique).

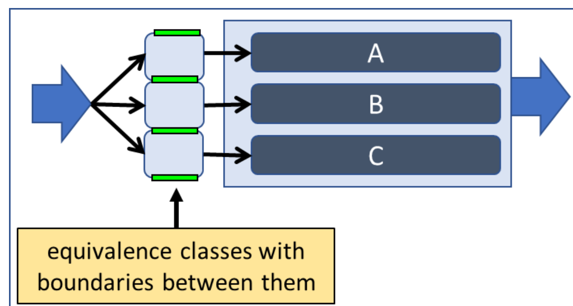

Figure 5: Representation of BVA Test Model

Note that to apply BVA you must first identify the test coverage items for EP.

- Test Coverage Items (two-value BVA)

 ©Stuart Reid, 2018

- The test coverage items are the upper and lower values of the equivalence classes.
- Where there is no adjoining equivalence class, then the next value outside the equivalence class is also a test coverage item.

Note that there is also a form of three-value BVA, but, for simplicity, we will only cover 2-value BVA here.

Example - Boundary Value Analysis (BVA)

- Required Test Coverage
    - All identified boundary values shall be covered.
- Test Basis
    - The system shall do X when input speed does not exceed 50 km/h, and if the speed is higher than 120 km/h it shall do Y, otherwise it shall do Z. Valid input speeds are from 0 km/h to 220 km/h – invalid speeds will cause an error message. All speeds are provided as integers.
- Test Model
    - We can derive the following input equivalence classes:
        - 0 ≤ speed ≤ 50 (X)
        - 120 < speed ≤ 220 (Y)
        - 50 < speed ≤ 120 (Z)
        - speed < 0 <u>or</u> speed > 220 (ERROR)

      Thus, the input speed boundaries, as shown in Figure 6, are:

        - between -1 and 0, 50 and 51, 120 and 121, 220 and 221.

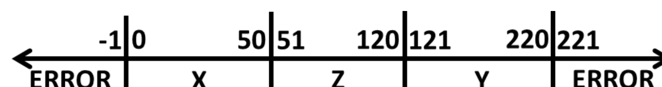

Figure 6: Example BVA Test Model

- Test Coverage Items (two-value BVA)
    - input speed = [-1]; [0, 50]; [51, 120]; [121, 220]; [221]

Decision Table Testing (DTT)

- Required Test Coverage
    - Coverage of feasible decision table rules.

      Infeasible rules are often found in decision tables with multiple conditions. For instance, if one condition is that age is less than 21, and another (seemingly) independent condition is that a car has been owned for over 10 years, then the rule corresponding to the combination of both conditions being true is (normally) going

to be infeasible. We typically allow infeasible rules to be discounted (ignored) when defining required test coverage for DTT.

- Test Model
    - Decision Table.

      May be available from the developers, but it is better practice to create your own test model, and, especially when deriving decision tables, you may find defects in the requirements.
- Test Coverage Items
    - Decision Table Rules.

      These rules are normally easily converted into test cases.

Example – Decision Table Testing

- Required Test Coverage
    - All feasible decision table rules shall be covered.
- Test Basis
    - Sims can rest if they're tired, unless they're at work. If they're not tired, then they don't rest.
- Test Model
    - See Figure 7.

		Rules			
		1	2	3	4
Condition(s)	Tired?	Yes	Yes	No	No
	At work?	Yes	No	Yes	No
Action(s)	Rest?	No	Yes	No	No

Figure 7: Example Decision Table

- Test Coverage Items (rules)
    - See the four rules in the decision table (Figure 7).

      Often, we can identify rules that are more likely than others – so making them higher risk (if the corresponding impact of failure is considered constant – or is unknown).
- Test Cases
    1. Inputs: Sim tired = yes; Location = office / Expected Result = Not resting
    2. Inputs: Sim tired = yes; Location = home / Expected Result = Resting
    3. Inputs: Sim tired = no; Location = factory / Expected Result = Not resting

 ©Stuart Reid, 2018

4. Inputs: Sim tired = no; Location = home / Expected Result = Not resting

If a Sim resting at work when tired was considered more risky, we could duplicate coverage of rule 1 in Figure 7 with an additional test case that covered the factory location.

State Transition Testing (STT)

- Required Test Coverage
    - There are several alternatives for test coverage when using state transition testing, and the choice should depend on the level of risk associated with test item, e.g. (in order of ascending risk/rigour):
        - Coverage of states
        - Coverage of single transitions (100% coverage ⇨ 0-switch coverage)
        - Coverage of pairs of transitions (100% coverage ⇨ 1-switch coverage)
- Test Model
    - State Model
        - e.g. state transition diagram, state table, UML statechart
- Test Coverage Items
    - States (required for state coverage)
    - Single transitions (required for 0-switch coverage)
    - Pairs of transitions (required for 1-switch coverage)

Example - State Transition Testing

- Required Test Coverage
    - All single transitions shall be covered (0-switch coverage).
- Test Basis
    - The fitness tracker shall either show the time or fitness info, depending on which is selected. The only exception is when the tracker is plugged-in for charging, when it shows charging information. When charging stops, the tracker automatically displays fitness information.
- Test Model (state transition diagram)
    - See Figure 8.

 ©Stuart Reid, 2018

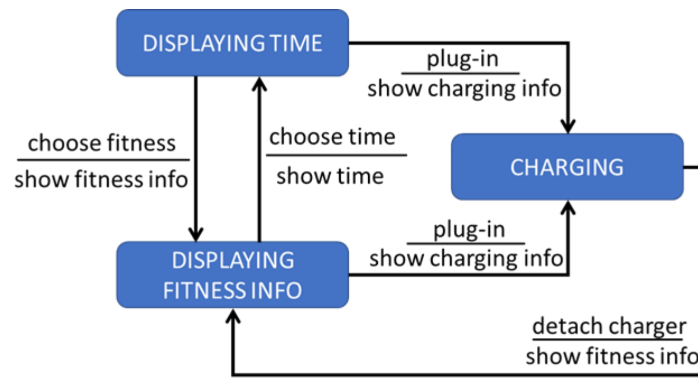

Figure 8: Example STT Test Model

- Test Coverage Items (transitions)
  - Displaying Time → Charging
  - Displaying Time → Displaying Fitness Info
  - Displaying Fitness Info → Charging
  - Displaying Fitness Info → Displaying Time
  - Charging → Displaying Fitness Info
- Test Cases

Test Case	Start State	Input	Expected action	End State
1	Displaying Time	plug-in	show charging info	Charging
2	Displaying Time	choose fitness	show fitness info	Displaying Fitness Info
3	Displaying Fitness Info	plug-in	show charging info	Charging
4	Displaying Fitness Info	choose time	show time	Displaying Time
5	Charging	detach charger	show fitness info	Displaying Fitness Info

If we applied risk-based testing at this level, we may determine that some transitions are higher risk than others, and so would create additional test cases to exercise the higher risk transitions.

Statement Testing

- Required Test Coverage
  - Statement coverage.

    Full (100%) coverage should ideally be required, if used.
- Test Model
  - Source code (typically of a software unit, module or component).

 ©Stuart Reid, 2018

- Test Coverage Items
    - Executable statements

Example - Statement Testing

- Required Test Coverage
    - 100% statement coverage shall be achieved.
- Test Basis
    - The program shall read in values for x and y. If x is the larger then x shall be set to 5 and y to zero. If y is the larger then y shall be set to 5 and x to zero. If x and y are equal they shall retain their original values.
- Test Model (source code)
    - See Figure 9.

```
1.    read (x)
2.    read (y)
3.    if x > y then
4.    comment:x is greater than y
5.       x := 5
6.       y := 0
7.    elsif y > x then
8.    comment:y is greater than x
9.       x := 0
10.      y := 5
11.   endif
```

Figure 9: Example Source Code

- Test Coverage Items (executable statements)
    - Lines 1, 2, 3, 5, 6, 7, 9, 10 and 11 in the source code (Figure 9).
    - Note that lines 4 and 8 are comments (non-executable)
- Test Cases

1. inputs: x=3, y = 6 / statements covered: 1-2-3-7-9-10-11 / expected outputs: x=0, y = 5
2. inputs: x=9, y = 2 / statements covered: 1-2-3-5-6-11 / expected outputs: x=5, y = 0

Branch Testing

- Required Test Coverage
    - Branch coverage.

      Full (100%) coverage should ideally be required, if used.
- Test Model
    - Control flow graph of the source code (typically of a software unit, module or component).

 ©Stuart Reid, 2018

- Test Coverage Items
    - Branches (arcs) in the control flow graph.

Example - Branch Testing

- Required Test Coverage
    - 100% branch coverage shall be achieved.
- Test Basis
    - The program shall read in values for x and y. If x is the larger then x shall be set to 5 and y to 0. If y is the larger then y shall be set to 5 and x to zero. If x and y are equal they shall retain their original values.
- Test Model
    - Control flow graph of the source code (see Figure 10).

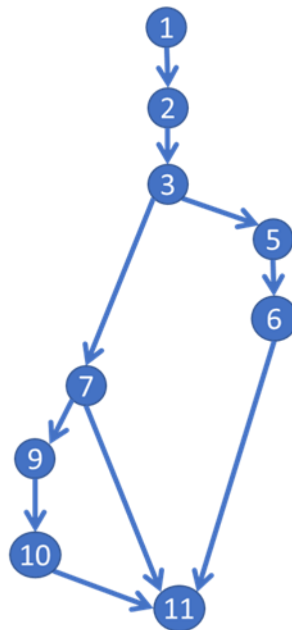

Figure 10: Example Control Flow Graph

- Test Coverage Items (branches)
    - 1 → 2 → 3
    - 3 → 5 → 6 → 11
    - 3 → 7
    - 7 → 9 → 10 → 11
    - 7 → 11
- Test Cases

1. inputs: x=3, y = 6 / 1 → 2 → 3 → 7 → 9 → 10 → 11 / expected outputs: x=0, y = 5

©Stuart Reid, 2018

2. inputs: x=9, y = 2 / 1 → 2 → 3 → 5 → 6 → 11 / expected outputs: x=5, y = 0
3. inputs: x=4, y = 4 / 1 → 2 → 3 → 7 → 11 / expected outputs: x=4, y = 4

Definitions

required test coverage

- level of test coverage set as a target for testing
- NOTE: Required test coverage is typically a major component of the test completion criteria.
- EXAMPLE: All identified boundary values, full statement coverage.

test basis

- body of knowledge used to determine the expected behaviour of a test item

test case

- set of preconditions, inputs, actions (where applicable), and expected results, derived to exercise test coverage items
- NOTE 1: Preconditions typically include the set-up of the test environment, databases, etc.
- NOTE 2: Expected results typically include outputs and postconditions, such as a changed system state

test completion criteria

- set of criteria used to determine that planned testing is complete
- NOTE: Typical test completion criteria include achieving a required test coverage and the maximum number of known defects remaining unfixed.
- EXAMPLE: All high importance requirements tested, 100% statement coverage achieved, and no outstanding critical or high severity defects.

test coverage

- the proportion of specified test coverage items exercised by a test case or test cases
- NOTE: Test coverage is typically expressed as a percentage (e.g. 40% decision coverage)

test coverage item

- measurable attribute of a test item that is the focus of testing
- EXAMPLE: Equivalence classes, transitions between states, executable statements.

test design technique

- procedure used to create or select a test model, identify test coverage items and derive corresponding test cases
- EXAMPLE: Equivalence partitioning, boundary value analysis, decision table testing, branch testing.
- NOTE: The test design technique is typically used to achieve a required test coverage.

 ©Stuart Reid, 2018

test item

- work product to be tested
- EXAMPLE: A software component, a system, a requirements document, a design specification, a user guide.

test model

- representation of an aspect of the test item, which is focused on the attributes used to define required test coverage
- NOTE 1: The test model and the required test coverage are used to identify test coverage items.
- NOTE 2: A separate test model may be required for each different type of required test coverage included in the test completion criteria.
- EXAMPLE: Requirements statements, equivalent functionality, boundaries between equivalence classes, state transition diagram, use case description, decision table, input syntax description, source code, control flow graph, classification tree.

test procedure

- sequence of test cases in execution order, with any associated actions required to set up preconditions and perform wrap up activities post execution

 ©Stuart Reid, 2018