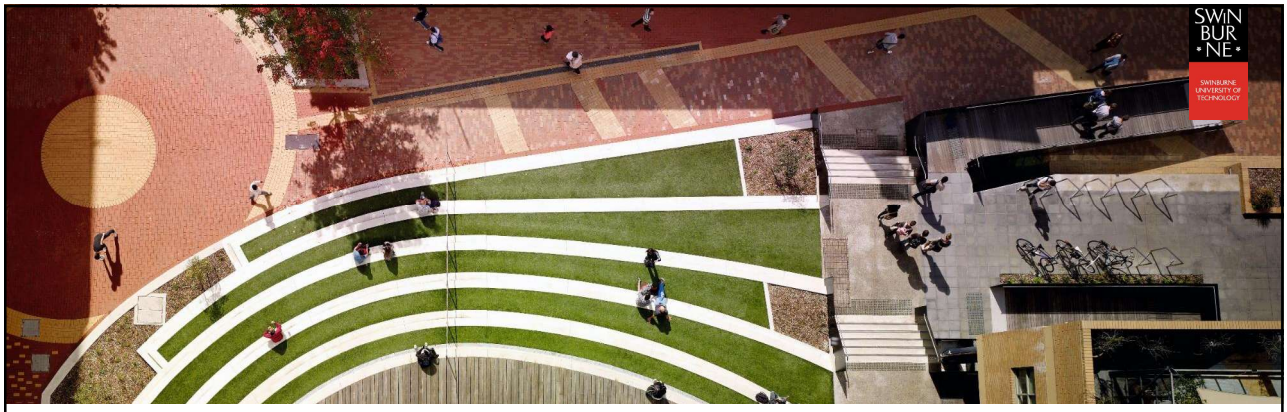


COS80022 – Software Quality and Testing

Week 7 – Black-Box Testing

Outline

- Categories of test techniques
- Black-box test techniques
  - Equivalent partitioning
  - Boundary value analysis
  - Decision table testing
  - State transition testing
  - Use case testing

Categories of Test Techniques

Categories of Test Techniques

- Black-box testing (this lecture)
- White-box testing (next lecture)
- Experience-based testing (next lecture)

- Purpose of a test technique
  - Test conditions
  - Test cases
  - Test data

Choosing Test Techniques

- Type of component or system
- Component or system complexity
- Regulatory standards
- Customer or contractual requirements
- Risk levels
- Risk types
- Test objectives
- Available documentation

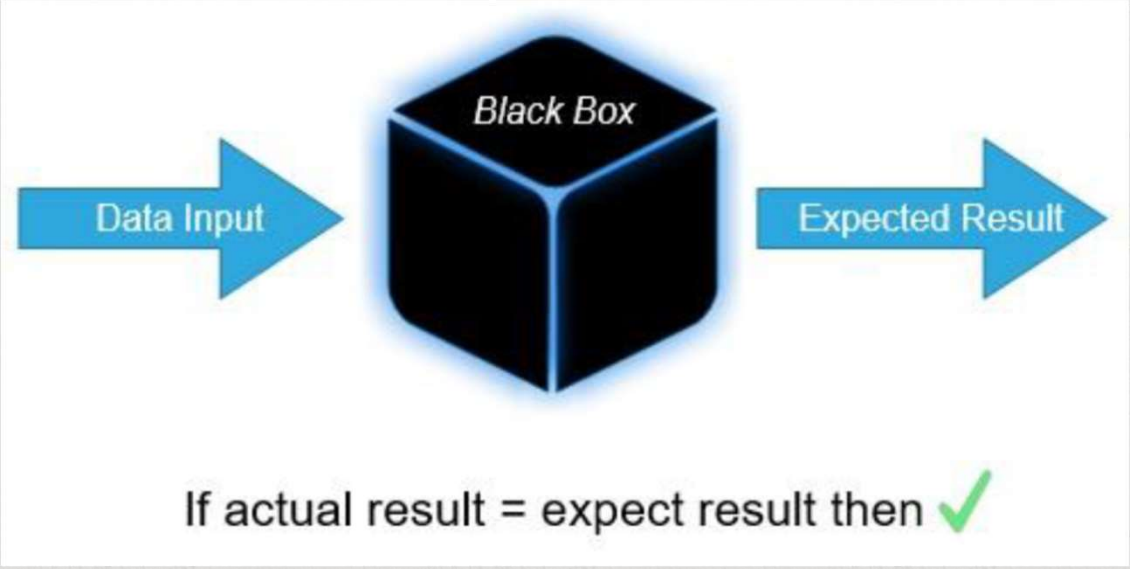
Choosing Test Techniques

- Tester knowledge and skills
- Available tools
- Time and budget
- Software development lifecycle model
- Expected use of the software
- Previous experience with using the test technique on the component or system to be tested
- The types of defects expected in the component or system

Black-Box Test Techniques

- Also called behavioural or behaviour-based techniques
- Based on an analysis of the appropriate test basis
  - Specification-based
- Applicable to both functional and non-functional testing

Black-Box Test Techniques



Common Characteristics of Black-Box Testing

- Test artefacts derived from a test basis that may include software requirements, specifications, use cases, and user stories
- Test cases may be used to detect gaps between the requirements and the implementation of the requirements, as well as deviations from the requirements
- Coverage is measured based on the items tested in the test basis and the technique applied to the test basis

Types of Black-Box Test Techniques

- Equivalence partitioning
- Boundary value analysis
- Decision table testing
- State transition testing
- Use case testing

Equivalence Partitioning

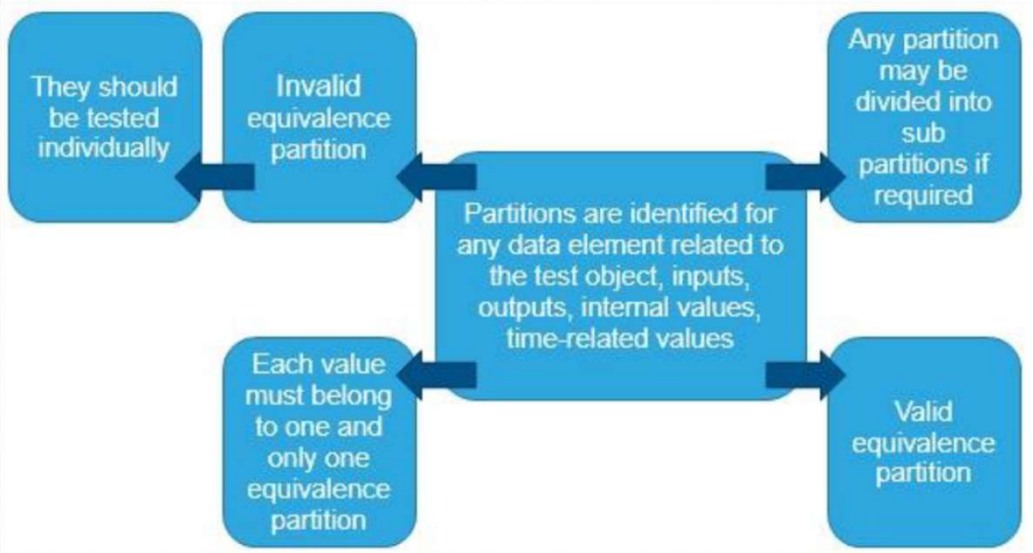
Equivalence Partitioning

- All possible inputs are divided into equivalence partitions
  - Also called equivalence classes
- A test case is selected for each partition
- Assumption – homogeneity
  - Inputs in the same "equivalence" partition cause similar behaviors

Equivalence Partitioning

- Two major steps
  - Identification of equivalence partitions
  - Selection of test inputs

Equivalence Partitioning



Equivalence Partitioning

- Criteria for identifying equivalence partitions
  - Coverage
  - Disjointedness
  - Representation
- A test input for each partition has to be identified that covers the partition
  - All elements of the partition are covered by the input

Equivalence Partitioning – A Simple Example

- A method that returns the number of days in a month, given the month and year

```
class MyGregorianCalendar {
...
    public static int getNumDaysInMonth(int month, int year) {…}
...
}
```

Equivalence Partitioning – A Simple Example

- Three equivalence class for month parameter
  - Months with 31 days (1, 3, 5, 7, 8, 10, 12)
  - Months with 30 days (4, 6, 9, 11)
  - February (2), that is, month with 28 or 29 days
- Two equivalence class for year parameter
  - Leap years
  - Non-leap years
- Total number of partitions: 3 * 2 = 6

Equivalence Partitioning – A Simple Example

Equivalence class	Value for month input	Value for year input
Months with 31 days, non–leap years	7 (July)	1901
Months with 31 days, leap years	7 (July)	1904
Months with 30 days, non–leap years	6 (June)	1901
Month with 30 days, leap year	6 (June)	1904
Month with 28 or 29 days, non–leap year	2 (February)	1901
Month with 28 or 29 days, leap year	2 (February)	1904

Equivalence Partitioning – Detailed Illustration

- Identifying partitions of the test object
- Valid and invalid partitions
- Test invalid partitions separately
- Correct invalid partitions tests
- Dividing a partition into sub partitions
- Test one data value from each partition
- Equivalence partition – tests
- Equivalence partitioning – defects & coverage

Equivalence Partitioning – Detailed Illustration

- An ordering system determines the size of a box needed when the quantity amount is entered. The field “order quantity” only accepts whole numbers; any other values will result in an error message “invalid value added, please add quantity in whole numbers starting from 1”.
- Any valid value will result in the order being accepted and a message displayed to the user of the system “The order is accepted <<size>> box needed”.

Equivalence Partitioning – Larger-Scale Example

Quantity	Size of box needed
Up to 10	Small box
Above 10, up to 50	Medium
Above 50, up to 100	Large
Above 100	Extra large box

Identifying Partitions of the Test Object

- Inputs – Order quantity field values
- Outputs – Messages

Quantity	Size of box needed
Up to 10	Small box
Above 10, up to 50	Medium
Above 50, up to 100	Large
Above 100	Extra large box

Valid and Invalid Partitions

Inputs	
Valid	**Invalid**
Whole Number	Alpha Characters
	Alpha Numeric
	Decimal
	Special
	Zero
	Negative

Outputs	
Valid	**Invalid**
Message "The order is accepted <<size>> box is needed"	Message "Invalid value added, please add quantity in whole numbers starting from 1"

Test Invalid Partitions Separately

- Counterexample to "separately"
  - Test step: Enter the following into the quantity field: A&-10.5
  - Expected result: Message "Invalid value added, please add quantity in whole numbers starting from 1"
  - Actual result: Message "The order is accepted A&-10.5 box is needed"
  - Status: Fail

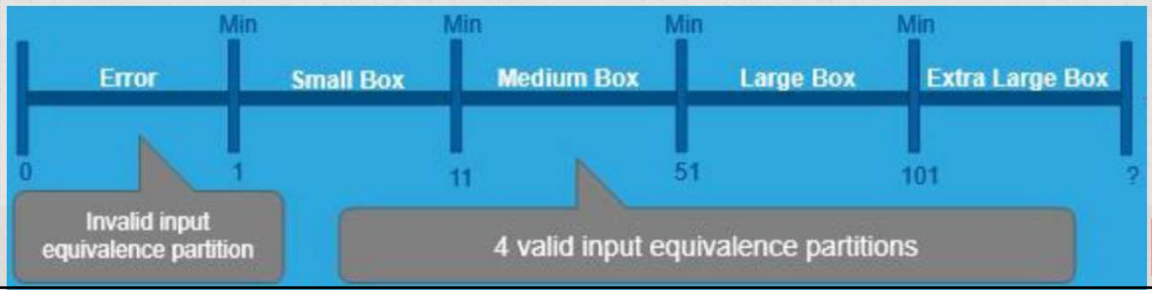
However, it is unknown which partition is failing the test step

Correct Invalid Partition Tests

Test Step	Expected Result	Actual Result	Status
Enter the following into the quantity field A	Message "Invalid value added, please add quantity in whole numbers starting from 1"	Message "The order is accepted A box is needed"	Fail
Enter the following into the quantity field &	Message "Invalid value added, please add quantity in whole numbers starting from 1"	Message "Invalid value added, please add quantity in whole numbers starting from 1"	Pass
Enter the following into the quantity field -10.5	Message "Invalid value added, please add quantity in whole numbers starting from 1"	Message "The order is accepted-10.5 box is needed"	Fail

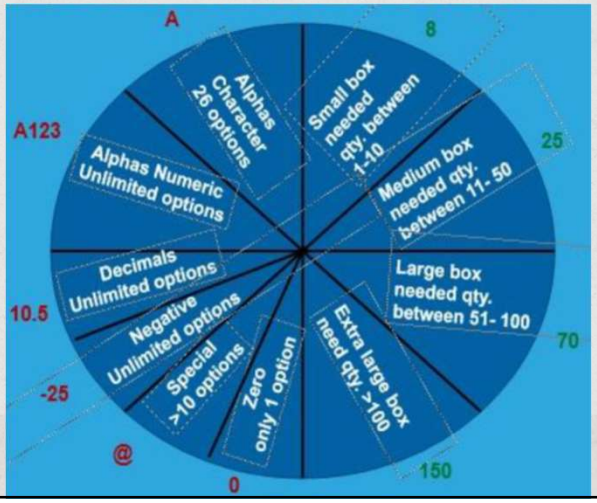
Dividing a Partition into Sub-Partitions

- As the sizes of the boxes change depending on the quantity entered the valid input partition whole numbers can be divided into sub-partitions
- The sub-partitions will be the ranges of quantity values that can be entered



Test One Data Value from Each Partition

- Total of 10 tests identified using equivalence partitioning



Equivalence Partitioning – Tests

No.	Input Partition	(In)valid	Test Data	Expected Results	
1	Alpha Character	Invalid	A	Error message "Invalid value added, please add quantity in whole numbers starting from 1"	
2	Alpha Numeric	Invalid	A123		
3	Zero	Invalid	0		
4	Decimal	Invalid	10.5		
5	Special Characters	Invalid	&		
6	Negative	Invalid	-25		
7	Whole No.s up to 10	Valid	8	Message displayed "The order is accepted	small size box needed"
8	Whole No.s above 10, up to 50	Valid	25		medium size box needed"
9	Whole No.s above 50, up to 100	Valid	70		large size box needed"
10	Whole No.s above 100	Valid	150		extra large size box needed"

Equivalence Partitioning – Defects & Coverage

- Likely to find defects in the handling of various data values
- To achieve 100% coverage with this technique, test cases must cover all identified partitions (including invalid partitions) by using a minimum of one value from each partition.
- Normally expressed as a percentage:

$$\frac{\text{No. of equivalence partitions tested by at least one value}}{\text{Total number of identified equivalence partitions}}$$

Equivalence Partitioning – Recap

- Input and output data can be split up into equivalence classes (a.k.a. equivalence partitions)
- Assumes that a test of any value from an equivalence class is equivalent to a test of any other value in that class
- Both valid (specified) and invalid (non-specified) equivalence classes are set up
- Equivalence classes are identified from both the inputs and the outputs of the software under test.

Equivalence Partitioning – Applicability

- Very versatile and commonly used, e.g. smoke testing
- Can be applied at all test levels
- All members of a set of values to be tested are expected to be handled in the same way
- The sets of values used by the application do not interact
- Valid and invalid partitions
- Usually used in combination with boundary value analysis.

Boundary Value Analysis

Boundary Value Analysis (BVA)

- Focus on the conditions at the boundaries of equivalence partitions
- Any element in the equivalence partition → elements selected from the edges
- Assumption:
  - Overlook of special cases
  - Domain error

BVA – A Simple Example

Equivalence class	Value for month input	Value for year input
Leap years divisible by 400	2 (February)	2000
Non–leap years divisible by 100	2 (February)	1900
Nonpositive invalid months	0	1291
Positive invalid months	13	1315

BVA

- "Boundary value analysis (BVA) is an extension of equivalence partitioning, but can only be used when the partition is ordered, consisting of numeric or sequential data. The minimum and maximum values (or first and last values) of a partition are its boundary values."
- BVA is applicable at all test levels

BVA – Detailed Illustration

- Identify minimum and maximum values for each partition
- BVA information in a table format
- Two point BVA
- Three point BVA
- BVA – defects
- BVA – coverage

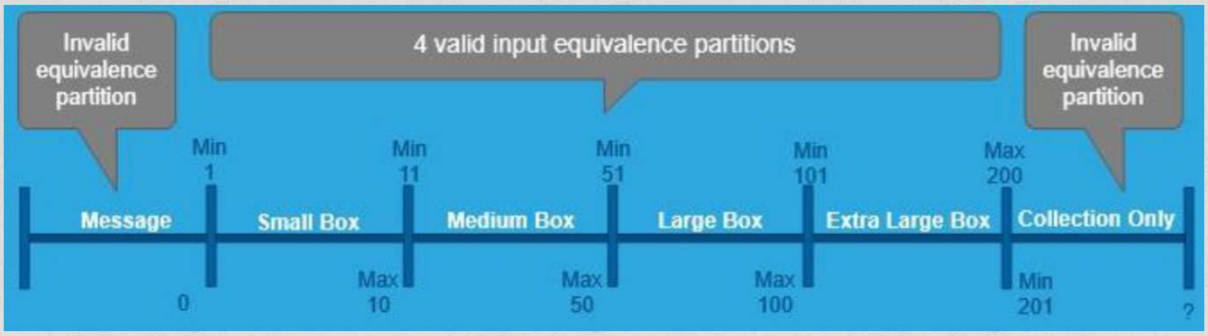
BVA – Detailed Illustration

- A new update to the requirement states the quantity field accepts only whole numbers as an input, using a keypad to limit inputs so that anything other than a whole number is impossible. The minimum quantity that can be entered is 1 and the maximum that an extra large box carries is 200. For anything over 200, a message will be displayed saying collection only.

BVA – Detailed Illustration

Quantity	Size of box needed
0	Message incorrect quantity
Up to 10	Small box
Above 10, up to 50	Medium box
Above 50, up to 100	Large box
Above 100, up to 200	Extra large box
Above 200	Collection only

Identify Min and Max for Each Partition



39

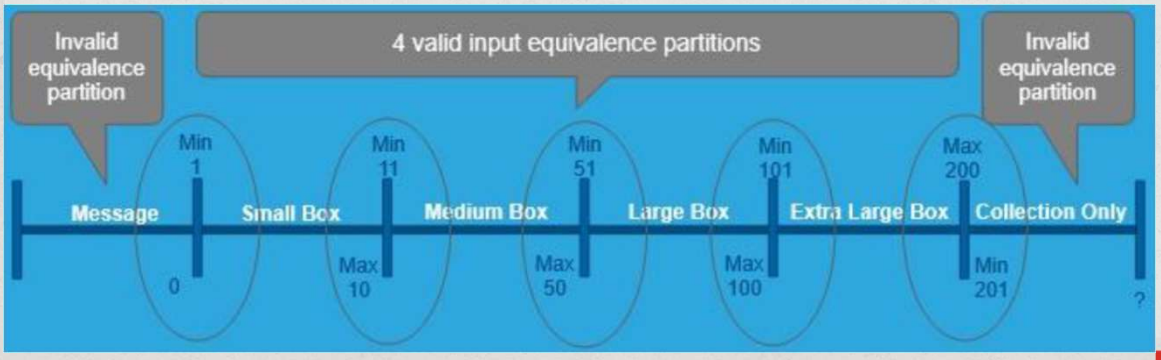
BVA Information in a Table Format

Equivalence Partition	Output	Min Boundary Value	Max Boundary Value
Invalid	Message invalid qty.	N/A	0
Valid	Small box	1	10
Valid	Medium box	11	50
Valid	Large box	51	100
Valid	Extra-large box	101	200
Invalid	(Collection only)	201	Unknown

40

Two Point BVA

- Identify the minimum and maximum for each partition → Identify two values where the partitions change



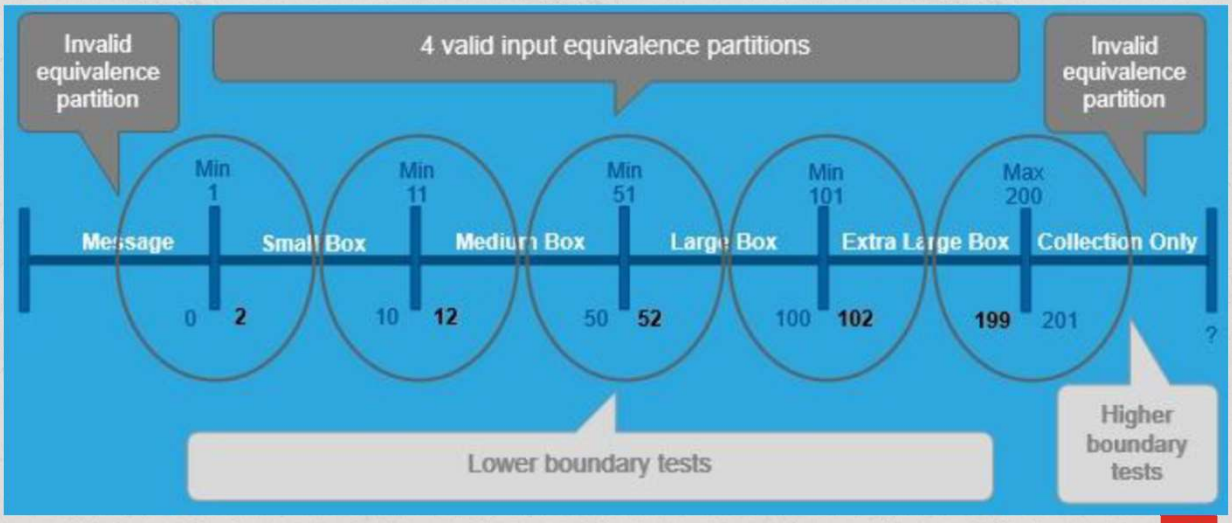
41

Two Point BVA

No	Boundary / Partition tested	Test Data	Expected Results
1	N/A Message incorrect qty	0	Error message "Invalid value added, please add quantity in whole numbers starting from 1"
2	Minimum / small box	1	Message displayed " The order is accepted small size box needed
3	Maximum / small box	10	Message displayed " The order is accepted small size box needed
4	Minimum / medium box	11	Message displayed " The order is accepted medium size box needed"
5	Maximum/ medium box	50	Message displayed " The order is accepted medium size box needed"
6	Minimum/ large box	51	Message displayed " The order is accepted large size box needed"
7	Maximum/ large box	100	Message displayed " The order is accepted large size box needed"
8	Minimum/ extra large box	101	Message displayed " The order is accepted extra large size box needed"
9	Maximum/ extra large box	200	Message displayed " The order is accepted extra large size box needed"
10	Minimum/ collection only	201	Message "Collection only"

42

Three Point BVA



BVA – Defects

- Behaviour at the boundaries of equivalence partitions is more likely to be incorrect than behaviour within the partitions
- BVA testing will reveal defects like these by forcing the software to show behaviours from a partition other than the one to which the boundary value should belong
- Example:
  - Input – 50
  - Output – "The order is accepted large size box needed"

BVA – Coverage

- Example: If only 5 test cases are executed from the two point BVA, then there would be 50% coverage

BVA – Recap

- Uses the same classes as equivalence partitioning
- Uses the principle that errors are made where values change and thus the program must treat inputs differently - so it's fault-based
- Tests are set up to exercise class boundaries and immediately adjacent to class boundaries for both input and output equivalence classes

BVA – Applicability

- Very versatile and can be applied at all test levels
- Ordered equivalence partitions must exist e.g.:
  - A range of numbers
  - Numeric attributes of non-numeric variables
  - Loops, including those in use cases
  - Stored data structures
  - Physical objects (including memory)
  - Time-determined activities;
- Not just limited to input values – e.g. count if cells in a spreadsheet.

Decision Table Testing

Decision Table Testing

- Ideal for describing situations where there are a number of different combinations of conditions
- Good for complex business rules
- Exercise combinations of conditions which may otherwise go untested
- Applicable to all test levels
- A decision table is made up of:
  - Conditions: Inputs
  - Actions: Outputs

A Decision Table – Example

	TC1	TC2	TC3	TC4
C1	T	F	T	F
C2	T	F	F	F
C3	F	T	F	F
A1	X			X
A2	X			
A3		X		
A4			X	

Conditions Inputs

Actions Outputs

For each set of inputs, the specified actions are determined

Each column represents a unique business rule and therefore a test case

Decision Table Notation

Condition:

- Y means the condition is true (may also be shown as T or 1)
- N means the condition is false (may also be shown as F or 0)
- – means the value of the condition doesn't matter (may also be shown as N/A)

Decision Table Notation

Actions:

- X means the action should occur (may also be shown as Y or T or 1)
- Blank means the action should not occur (may also be shown as – or N or F or 0)

Decision Table Notation

Collapsing full decision table by deleting columns

- Containing impossible combinations of conditions
- Containing possible but infeasible combinations of conditions
- That test combinations of conditions that do not affect the outcomes

Decision Table Testing

- Often perceived as a more specialised technique – It is useful for testing (amongst other things):
  - Rules engines
  - Any rule-based decisions
- Very useful for testing higher-risk situations where:
  - more than one input is required to occur at the same time
  - outputs are dependent on complex rules

Applicability of Decision Table Testing

- Useful for Integration, System and Acceptance testing (sometimes used in Component testing)
- Where requirements have complex business rules with multiple combinations of input conditions
- May already exist in the test basis or may have to be designed by the tester
- Should record defined and un-defined conditions

Decision Table – Example

- A new application for the management and preparation of tax returns has deductions functionality. There is a different deduction amount for each employee type: permanent, part-time or casual. Deductions are allowed for cars and study loans.
- Deduction amounts per year, per item:
  - For permanent employees it is $1,000;
  - For part-time employees it is $600;
  - For casual employees it is $250
- If an employee takes out a study loan then they need to record the average hours of study per week.

Identify Conditions and Actions

Conditions

- Employee type
  - Permanent
  - Part-time
  - Casual
- Deduction item
  - Car
  - Study loan

Actions

- Deduction amount
  - $1,000
  - $600
  - $250
- Record hours of study

Create Decision Table

	TC1	TC2	TC3	TC4	TC5	TC6
Permanent	T	T	F	F	F	F
Part-time	F	F	T	T	F	F
Casual	F	F	F	F	T	T
Car	T	F	T	F	T	F
Loan	F	T	F	T	F	T
$1,000	X	X	–	–	–	–
$600	–	–	X	X	–	–
$250	–	–	–	–	X	X
Hours of study	–	X	–	X	–	X

Decision Table – Defects

- Helps to identify all the important combinations of conditions, some of which might otherwise be overlooked
- It also helps in finding any gaps in the requirements

Decision Table – Coverage

- Coverage is measured as the number of decision rules tested by at least one test case, divided by the total number of decision rules, normally expressed as a percentage
- Example: 6 decision rules have been identified, but only 3 decision rules were executed. Then only 50% coverage would have been achieved.

State Transition Testing

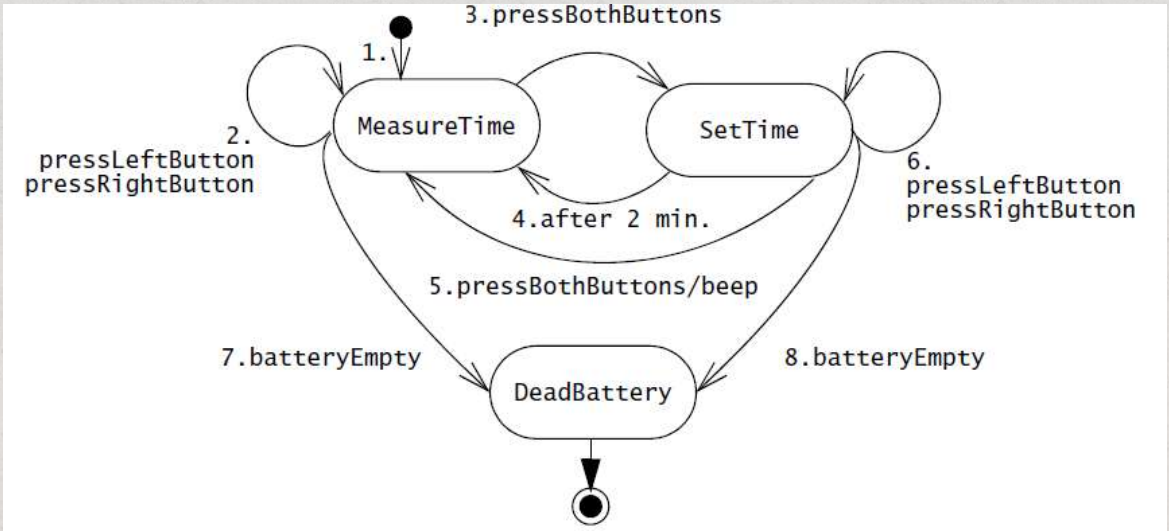
State Transition Testing

- Systems move from a set of states to another set based on the inputs that are applied
- Mapping these changes in states and their transitions are the basic parts of state transition testing
- State transition testing is often used for embedded systems and technical automation in general

State Transition Testing

- Compare the resulting state of the system with the expected state
  - Traditionally, compare the test results with oracle
- For each state, a representative set of stimuli is derived for each transition
- The attributes of the class are then instrumented and tested after each stimuli has been applied to ensure that the class has reached the specified state

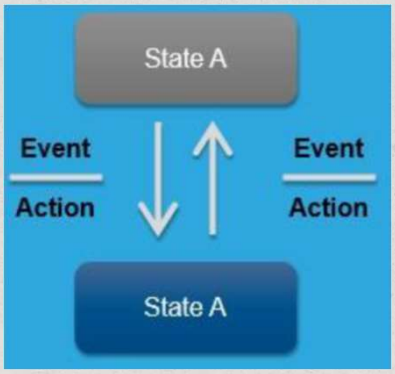
State Transition Testing – A Simple Example



State Transition Testing – A Simple Example

Stimuli	Transition tested	Predicted resulting state
Empty set	1. *Initial transition*	`MeasureTime`
Press left button	2.	`MeasureTime`
Press both buttons simultaneously	3.	`SetTime`
Wait 2 minutes	4. *Timeout*	`MeasureTime`
Press both buttons simultaneously	3. *Put the system into the SetTime state to test the next transition.*	`SetTime`
Press both buttons simultaneously	5.	`SetTime->MeasureTime`
Press both buttons simultaneously	3. *Put the system into the SetTime state to test the next transition.*	`SetTime`
Press left button	6. Loop back onto `MeasureTime`	`MeasureTime`

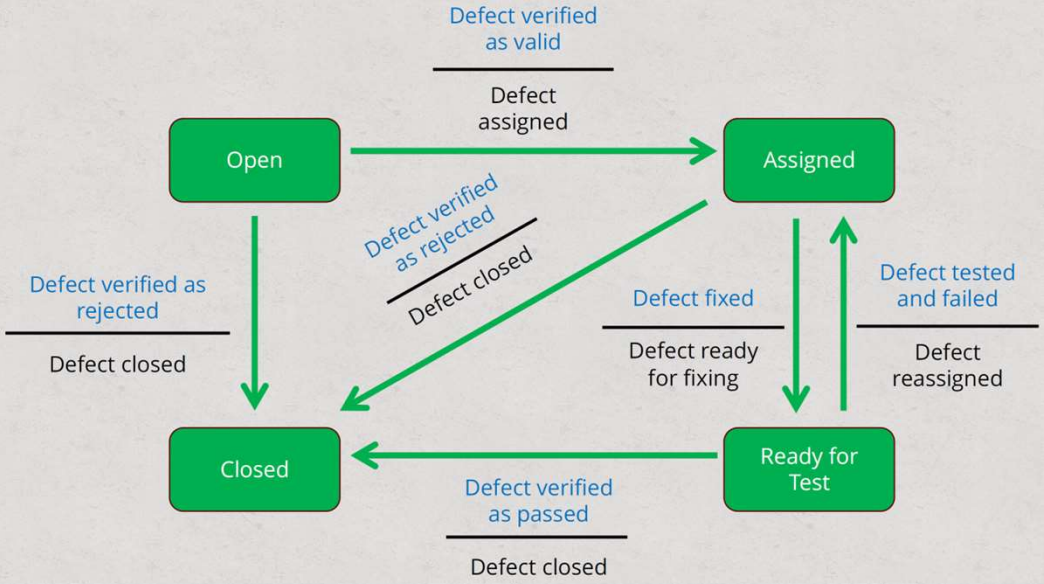
State Transition Testing – Detailed Illustration



State Transition Diagram

- The boxes specify the state or condition the system is in
- The arrows show the direction in which the transition may flow
- The event is the input which must applied to achieve the state
- The action is the outcome of the input applied

Create a State Transition Diagram



Create State Transition Tests

Tests can be designed to cover the following:

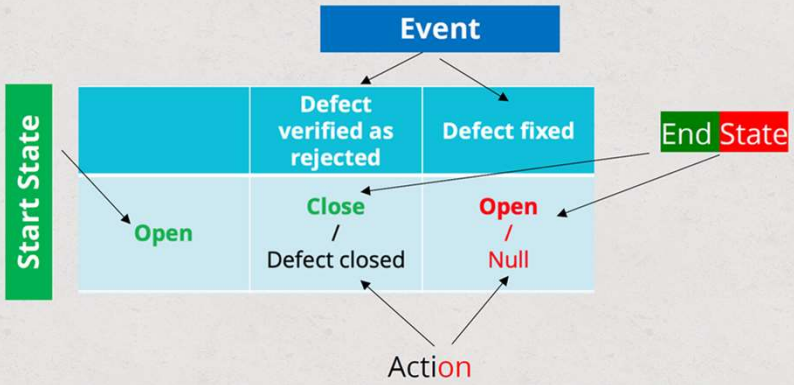
- A typical sequence of states
- To cover every state
- To exercise every single transition
- To exercise specific sequences of transitions
- To test invalid transitions

Test Cases for All Valid Single Transitions

Test case	1	2	3	4	5	6
Start State	**Open**	**Assigned**	**Open**	**Assigned**	**Ready for Test**	**Ready for Test**
Event	Defect verified as valid	Defect verified as rejected	Defect verified as rejected	Defect fixed	Defect tested and failed	Defect tested and passed
Action	Defect assigned	Defect closed	Defect closed	Defect ready for testing	Defect reassigned	Defect closed
End State	**Assigned**	**Closed**	**Closed**	**Ready for Test**	**Assigned**	**Closed**

Create State Tables

- The main purpose of the state table is to identify negative tests
- For example:
  - Start State: Open
  - Event: Defect Fixed
  - End State: **Open**
  - Action: **Null (a negative test)**



Create Invalid Tests using a State Table

Start State \ Event	Defect verified as rejected	Defect fixed	Defect tested and failed	Defect verified as valid	Defect tested and passed
Open	Closed / Defect closed	Open / Null	Open / Null	Assigned / Defect assigned	Open / Null
Assigned	Closed / Defect closed	Ready for Test / Defect ready for fixing	Assigned / Null	Assigned / Null	Assigned / Null
Closed	Closed / Null	Closed / Null	Closed / Null	Closed / Null	Closed / Null
Ready for Test	Ready for test / Null	Ready for test / Null	Assigned / Defect reassigned	Ready for test / Null	Closed / Defect closed

State Transition Testing – Defects

- Typical defects include incorrect processing in the current state, incorrect transitions and the need for states or transitions that do not exist
  - Wrong number of states
  - Wrong transition for given state
  - Wrong action for given transition
  - Dead states / black holes
  - Unreachable states
- Defects may also be found in the specification document

State Transition Testing – Coverage

- Typical defects include incorrect processing in the current state, incorrect transitions and the need for states or transitions that do not exist
- Defects may also be found in the specification document
- Coverage is commonly measured as the number of identified states or transitions tested, divided by the total number of identified states or transitions in the test object, normally expressed as a percentage

Use Case Testing

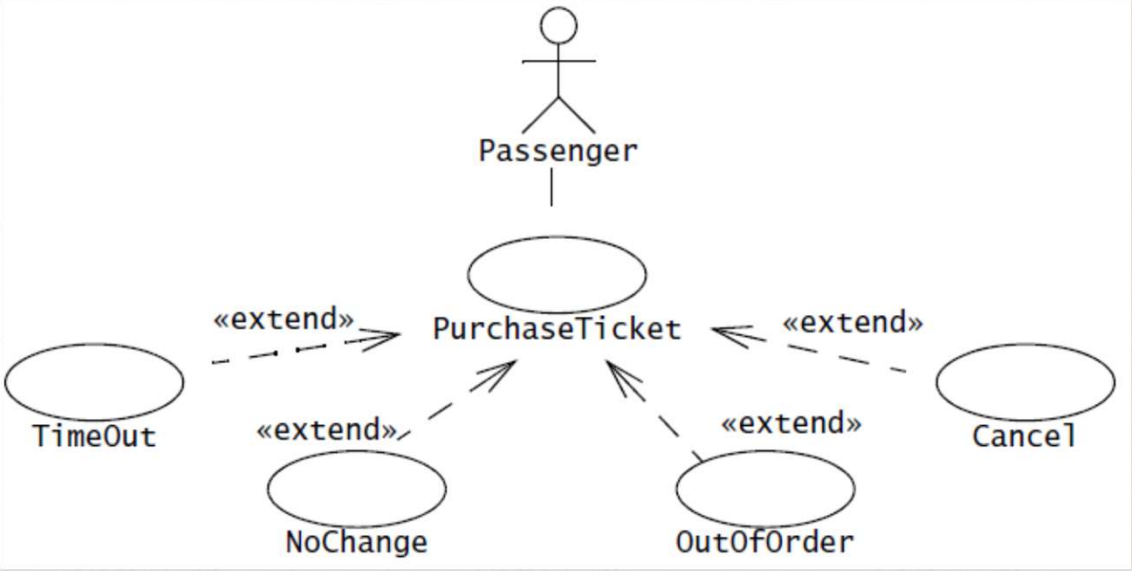
Use Case Testing

- Use cases are scenarios that describe the use of a systems and are associated with actors (users, external hardware or other systems)
- A scenario is a sequence of steps that describe the interaction between the actor(s) and the system

Use Case Testing

- Formal definition of
  - Use Case: A sequence of transactions in a dialogue between a user and the system with a tangible result. In other words, what the system has to do from the user's point of view.
  - Use Case Testing: A black box test design technique in which test cases are designed to execute user scenarios.
- Use case testing provides transactional, scenario-based tests that should emulate usage of the system.

Use Case Testing – Example in Lecture 5



Use Case Testing – Example in Lecture 5

Use case name	`PurchaseTicket`
Entry condition	The `Passenger` is standing in front of ticket `Distributor`. The `Passenger` has sufficient money to purchase ticket.
Flow of events	1. The `Passenger` selects the number of zones to be traveled. If the `Passenger` presses multiple zone buttons, only the last button pressed is considered by the `Distributor`. 2. The `Distributor` displays the amount due. 3. The `Passenger` inserts money. 4. If the `Passenger` selects a new zone before inserting sufficient money, the `Distributor` returns all the coins and bills inserted by the `Passenger`. 5. If the `Passenger` inserted more money than the amount due, the `Distributor` returns excess change. 6. The `Distributor` issues ticket. 7. The `Passenger` picks up the change and the ticket.
Exit condition	The `Passenger` has the selected ticket.

Use Case Testing – Example in Lecture 5

Test case name	`PurchaseTicket_CommonCase`
Entry condition	The `Passenger` standing in front of ticket `Distributor`. The `Passenger` has two $5 bills and three dimes.
Flow of events	1. The `Passenger` presses in succession the zone buttons 2, 4, 1, and 2. 2. The `Distributor` should display in succession $1.25, $2.25, $0.75, and $1.25. 3. The `Passenger` inserts a $5 bill. 4. The `Distributor` returns three $1 bills and three quarters and issues a 2-zone ticket. 5. The `Passenger` repeats steps 1–4 using his second $5 bill. 6. The `Passenger` repeats steps 1–3 using four quarters and three dimes. The `Distributor` issues a 2-zone ticket and returns a nickel. 7. The `Passenger` selects zone 1 and inserts a dollar bill. The `Distributor` issues a 1-zone ticket and returns a quarter. 8. The `Passenger` selects zone 4 and inserts two $1 bills and a quarter. The `Distributor` issues a 4-zone ticket. 9. The `Passenger` selects zone 4. The `Distributor` displays $2.25. The `Passenger` inserts a $1 bill and a nickel, and selects zone 2. The `Distributor` returns the $1 bill and the nickel and displays $1.25.
Exit condition	The `Passenger` has three 2-zone tickets, one 1-zone ticket, and one 4-zone ticket.

Create Use Case Tests

- Tests are designed to exercise the defined behaviours including:
  - Basic
  - Exceptional or alternative
  - Error handling
- Can be described by interactions, pre- and post-conditions
- May be represented by work flows, activity diagrams or business process models

Applicability of Use Case Testing

- Use case testing is usually applied at the system and acceptance testing levels.
- It may be used for integration testing depending on the level of integration and even component testing depending on the behavior of the component.
- Use cases are also often the basis for performance testing because they portray realistic usage of the system.
- The scenarios described in the use cases may be assigned to virtual users to create a realistic load on the system.

Use Case Testing – Defects

- Defects that may be found by use case testing include errors in business process scenarios, for example
  - missing an exceptional behaviour path
  - Inconsistencies and errors in requirements
  - Business process defects
  - Interoperability defects
  - Integration issues between processes and systems
  - Usability issues with different actors

Use Case Testing – Coverage

- Coverage can be measured by the percentage of use case behaviours tested divided by the total number of use case behaviours
- Decision table testing may be used for:
  - Rules coverage and syntax
- Equivalence Partitions and Boundary Values may be also used for:
  - Input values
  - Output Values

Summary

- Equivalence partitioning
- Boundary value analysis
- Decision table testing
- State transition testing
- Use case testing

References

- Sommerville, I. *Software Engineering* (Chapter 8)
- Bruegge, B. & Dutoit A.H. *Object-Oriented Software Engineering Using UML, Patterns, and Java* (Chapter 11)
- International Software Testing Qualifications Board. *ISTQB Foundation Level (Core) Syllabus*

Next

- Tutorial
  - Lab work: EP & BVA
- Next week
  - Guest Lecture – DevOps
  - NO tutorial
- Week 9
  - White-box testing