3. Test Techniques and their Characteristics

IT6206 - Software Quality Assurance

Level III - Semester 6

© 2022 e-Learning Centre, UCSC

Overview

- Test design techniques help every software development project improve its overall quality and effectiveness.
- There are many different types of software testing techniques, each with its own strengths and weaknesses.
- Each individual technique is good at finding particular types of defect and relatively poor at finding other types.
- Categories of Test Design Techniques
    - Black Box Test Techniques
    - White Box Test Techniques
    - Experience Based Test Techniques

© 2022 e-Learning Centre, UCSC

Intended Learning Outcomes

At the end of this lesson, you will be able to;

- Describe the concept and importance of black-box test techniques, white-box test techniques, and experience-based test techniques
- Explain the characteristics, commonalities, and differences between different test techniques.
- Apply test techniques to derive test cases from given requirements.
- Identify test conditions, test cases, and test data according to their applicability to certain situations and test levels.

© 2022 e-Learning Centre, UCSC

List of subtopics

© 2022 e-Learning Centre, UCSC

List of subtopics

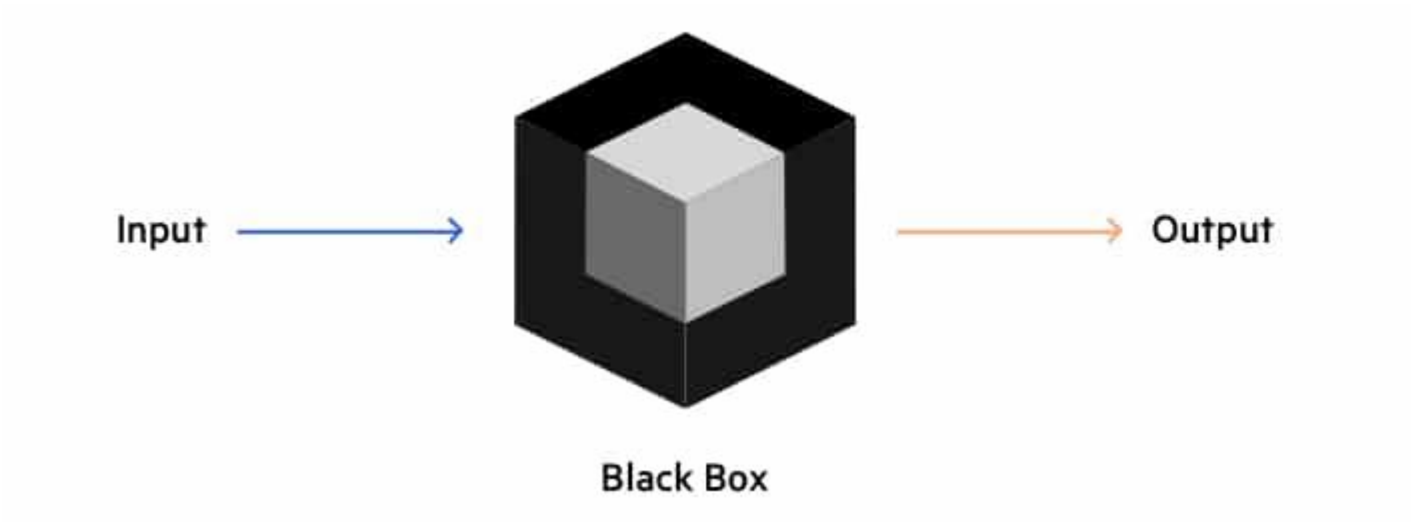
© 2022 e-Learning Centre, UCSC

3.1 Black-box Test Techniques

© 2022 e-Learning Centre, UCSC

3.1 Black-box Test Techniques

- The technique of testing without having any knowledge of the interior workings of the application is called Black Box testing.
- Black box testing techniques view the software as a black-box with inputs and outputs, but they have no knowledge of how the system or component is structured inside the box.
- Black-box techniques are appropriate at all levels of testing where a specification exists.



© 2022 e-Learning Centre, UCSC

3.1 Black-box Test Techniques

- Characteristics
    - Test conditions, test cases, and test data are 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 requirement and the implementation of the requirement as well as deviations from the requirements.
    - Coverage is measured based on the items tested in the test basis.

© 2022 e-Learning Centre, UCSC

3.1 Black-box Test Techniques

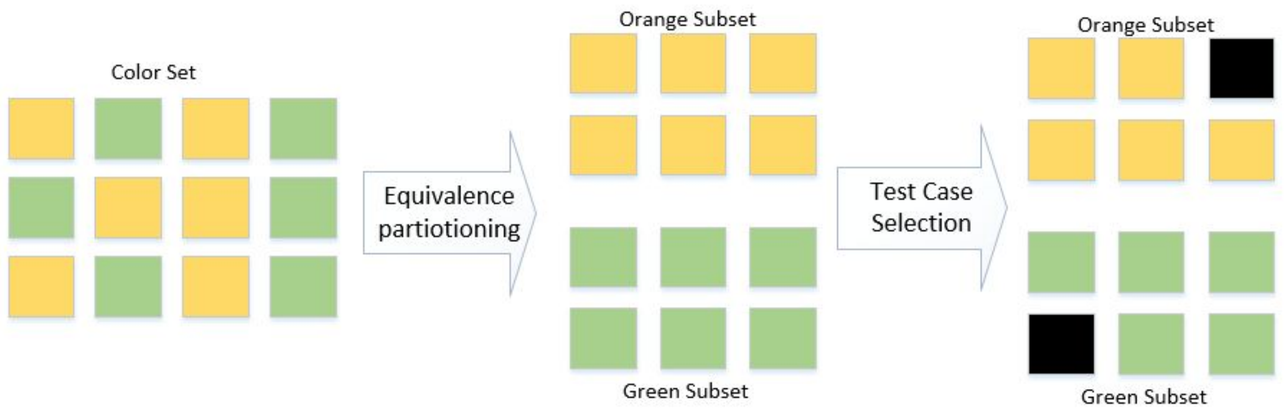
- Techniques
  - Equivalence Partitioning
  - Boundary Value Analysis
  - Decision Table Testing
  - State Transition Testing
  - Use Case Testing

TEST CASE INPUT
APPLICATION
TEST CASE OUTPUT
ZERO KNOWLEDGE

© 2022 e-Learning Centre, UCSC

3.1.1 Equivalence Partitioning

- Equivalence partitioning (EP) is a black-box technique which can be applied at any level of testing and is often a good technique to use first.
- The idea behind the technique is to divide or partition a set of test conditions into groups or sets that can be considered the same.
- Equivalence partitions are also known as equivalence classes.



Source : https://www.maniuk.net/search/label/equivalent%20partitioning

© 2022 e-Learning Centre, UCSC

3.1.1 Equivalence Partitioning

- In equivalence-partitioning, only one condition from each partition is tested assuming that all the conditions in one partition will be treated in the same way by the software.
- If one condition in a partition works, it is assumed that all of the conditions in that partition will work, and so there is little point in testing any of these others.
- Similarly, if one of the conditions in a partition does not work, it is assumed that none of the conditions in that partition will work so again there is little point in testing any more in that partition.

© 2022 e-Learning Centre, UCSC

3.1.1 Equivalence Partitioning

Example :

A savings account in a bank has following rates of interest depending on the account balance. In order to test the software that calculates the interest due, we can identify the ranges of balance values that earn the different rates of interest.

Rate of Interest	Account Balance
3%	$0 to $100
5%	$100 to $1000
7%	$1000 and above

Initially, three valid equivalence partitions and one invalid partition can be identified from this example as shown in the next slide.

© 2022 e-Learning Centre, UCSC

3.1.1 Equivalence Partitioning

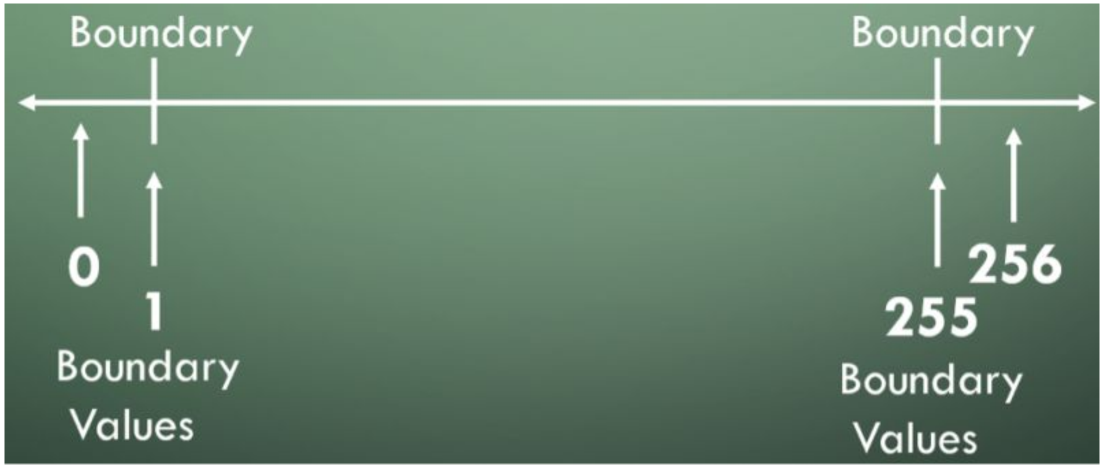
Invalid partition	Valid partition (for 3% interest)	Valid partition (for 5% interest)	Valid partition (for 7% interest)
$-0.01	$0.00 $100.00	$100.01 $999.99	$1000.00

- Here four partitions have been identified, even though the specification only mentions three.
- This highlights that a tester should not only test what is in the specification, but should also think about things that haven’t been specified.
- When designing the test cases for this software it should be ensured that all the three valid equivalence partitions are covered once, and that the invalid partition is also tested at least once.

© 2022 e-Learning Centre, UCSC

3.1.2 Boundary Value Analysis

- Boundary value analysis (BVA) is based on testing at the boundaries between partitions.
- In BVA, both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions) are identified.
- Boundary values are defined as those values on the edge of a partition.



© 2022 e-Learning Centre, UCSC

3.1.2 Boundary Value Analysis

Example 01:

Consider a printer that has an input option of the number of copies to be made, from 1 to 99. To apply boundary value analysis, we will take the minimum and maximum (boundary) values from the valid partition (1 and 99 in this case) together with the first or last value respectively in each of the invalid partitions adjacent to the valid partition (0 and 100 in this case).

Invalid partition	Valid partition	Invalid partition
0	**1** **99**	**100**

In this example, there are three equivalence partitioning tests (one from each of the three partitions) and four boundary value tests.

© 2022 e-Learning Centre, UCSC

3.1.2 Boundary Value Analysis

Boundary value analysis can also be applied to the whole of a string of characters (e.g. a name or address).

Example 02: The number of characters in the string is a partition

Between 1 and 30 characters is the valid partition with valid boundaries of 1 and 30. The invalid boundaries would be 0 characters (null, just hit the Return key) and 31 or above characters. Both of these should produce an error message.

Invalid partition	Valid partition	Invalid partition
0	1 30	31

© 2022 e-Learning Centre, UCSC

3.1.2 Boundary Value Analysis

Exercise

Consider the bank system example described in the previous section in equivalence partitioning and try to identify the boundary values.

© 2022 e-Learning Centre, UCSC

3.1.3 Decision Table Testing

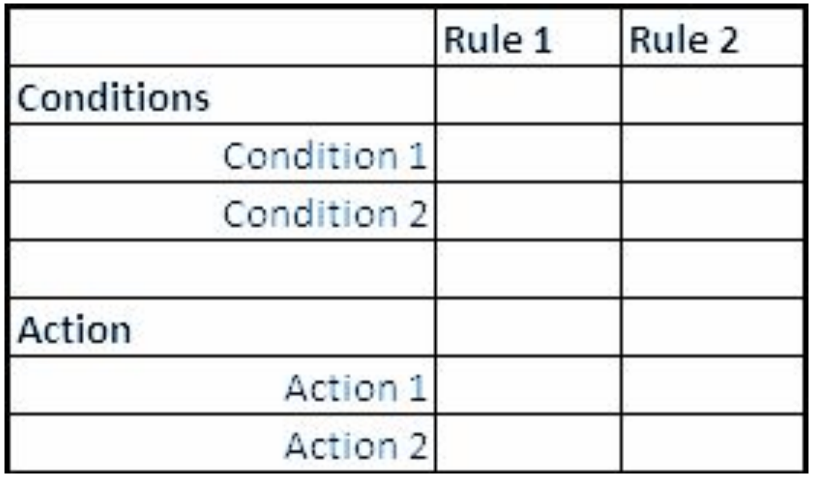
- Decision tables provide a systematic way of stating complex business rules, which is useful for developers as well as for testers.
- This technique is sometimes also referred to as a 'cause-effect' table.
- Decision tables aid the systematic selection of effective test cases and can have the beneficial side-effect of finding problems and ambiguities in the specification.
- They help testers explore the effects of combinations of different inputs and other software states that must correctly implement business rules.
- It is a technique that works well in conjunction with equivalence partitioning.

© 2022 e-Learning Centre, UCSC

3.1.3 Decision Table Testing

How to Use decision tables for test designing?

- Identify a suitable function or subsystem which reacts according to a combination of inputs or events. The system should not contain too many inputs otherwise the number of combinations will become unmanageable.
- Once the aspects that need to be combined are identified, they are put into a table listing all the combinations of True and False for each of the aspects.



	Rule 1	Rule 2
Conditions		
Condition 1		
Condition 2		
Action		
Action 1		
Action 2		

Source : https://www.maniuk.net/2016/09/qa-decision-tables.html

© 2022 e-Learning Centre, UCSC

3.1.3 Decision Table Testing

Example

In a loan application, you can enter the amount of the monthly repayment or the number of years you want to take to pay it back (the term of the loan). If you enter both, the system will make a compromise between the two if they conflict. The two conditions are the loan amount and the term, so we put them in a table as follows.

Condition	Rule 01	Rule 02	Rule 03	Rule 04
Repayment amount has been entered				
Term of loan has been entered				

© 2022 e-Learning Centre, UCSC

3.1.3 Decision Table Testing

Next we have to identify all of the combinations of True and False. With two conditions, each of which can be True or False, we will have four combinations (two to the power of the number of things to be combined).

Condition	Rule 01	Rule 02	Rule 03	Rule 04
Repayment amount has been entered	T	T	F	F
Term of loan has been entered	T	F	T	F

Legend
Rule 01 : Customer has entered both repayment amount and term of loan
Rule 02 : Customer has entered only repayment amount
Rule 03 : Customer has entered only term of loan
Rule 04 : Customer has not entered anything in either of the two fields.

© 2022 e-Learning Centre, UCSC

3.1.3 Decision Table Testing

Next step is to identify the correct outcome for each combination.

Condition	Rule 01	Rule 02	Rule 03	Rule 04
Repayment amount has been entered	T	T	F	F
Term of loan has been entered	T	F	T	F
Actions / Outcomes				
Result	Error Message	Process Repayment Amount	Process term of loan	Error Message

© 2022 e-Learning Centre, UCSC

3.1.3 Decision Table Testing

Decision Table Interpretation:

- **Case 01** : Expected result should be an error as the customer is not allowed to enter both repayment and term of loan
- **Case 02** : Process the repayment amount
- **Case 03** : Process the term of loan
- **Case 04** : Expected result should be an error as the customer has not entered anything into the two fields

© 2022 e-Learning Centre, UCSC

3.1.3 Decision Table Testing

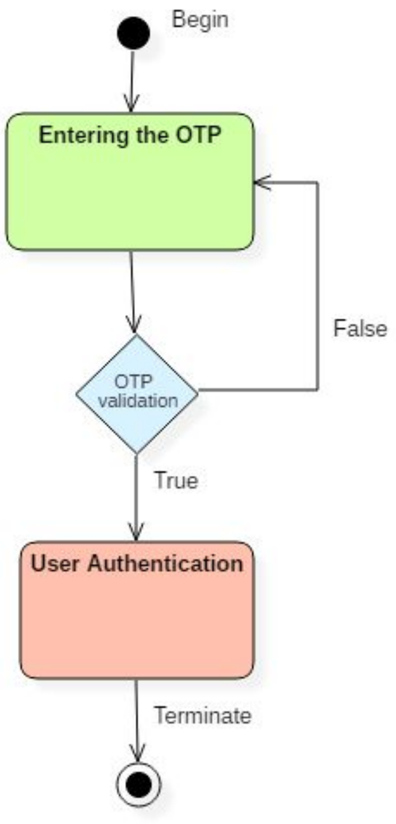
- The final step of this technique is to write test cases to exercise each of the four rules in our table.
- However, if there are a lot of combinations, it may not be possible or sensible to test each and every combination.
- Therefore it is always better to prioritize and test the most important combinations. Having the full table helps to decide which combinations should be tested and which should not to tested.

© 2022 e-Learning Centre, UCSC

3.1.4 State Transition Testing

- State transition testing is used where some aspect of the system can be described in what is called a ‘finite state machine’.

- Any system where you get a different output for the same input, depending on what has happened before, is a finite state system and a finite state system is often shown as a state diagram.



Source : https://www.guru99.com/state-machine-transition-diagram.html

© 2022 e-Learning Centre, UCSC

3.1.4 State Transition Testing

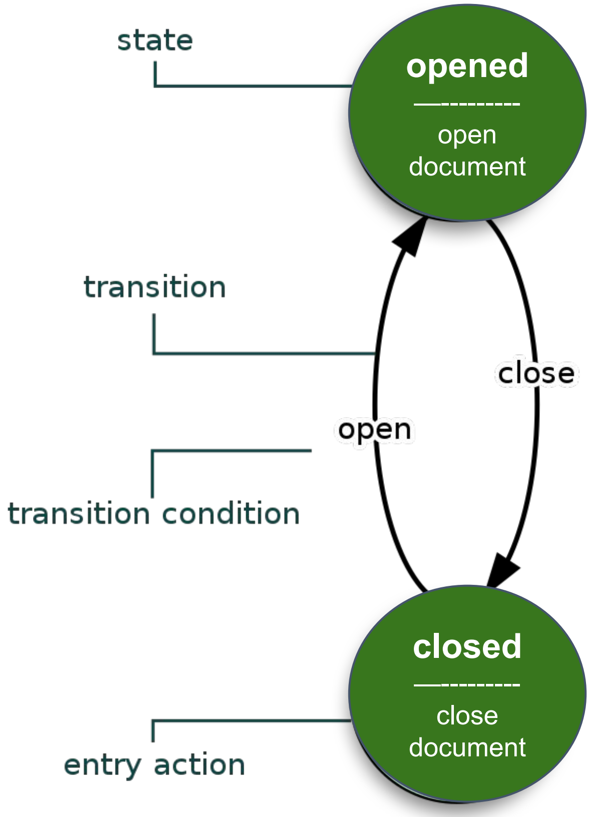
- One of the advantages of the state transition technique is that the model can be as detailed or as abstract as you need it to be.

Example :

In word processor, if a document is open, you are able to close it. If no document is open, then 'Close' is not available.

After you choose 'Close' once, you cannot choose it again for the same document unless you open that document.

A document thus has two states: open and closed

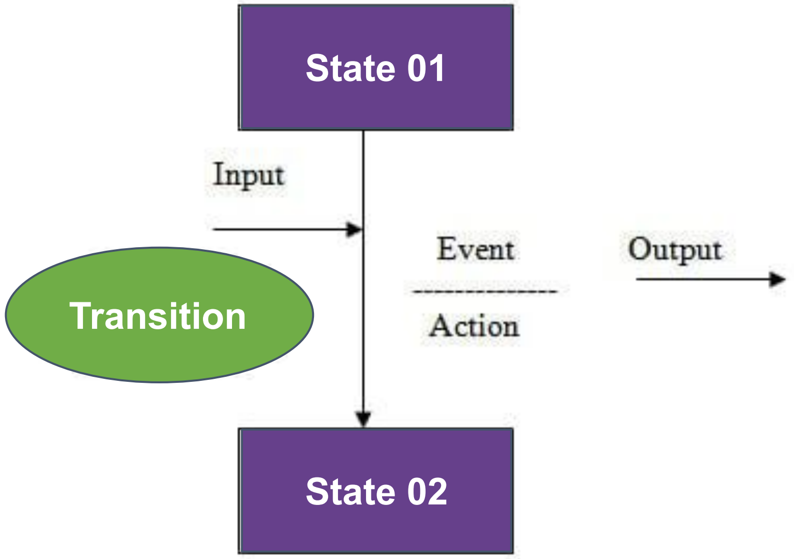


© 2022 e-Learning Centre, UCSC

3.1.4 State Transition Testing

A state transition model has four basic parts:

- The states that the software may occupy (open/closed or funded/insufficient funds);
- The transitions from one state to another (not all transitions are allowed);
- The events that cause a transition (closing a file or withdrawing money);
- The actions that result from a transition (an error message or being given your cash).

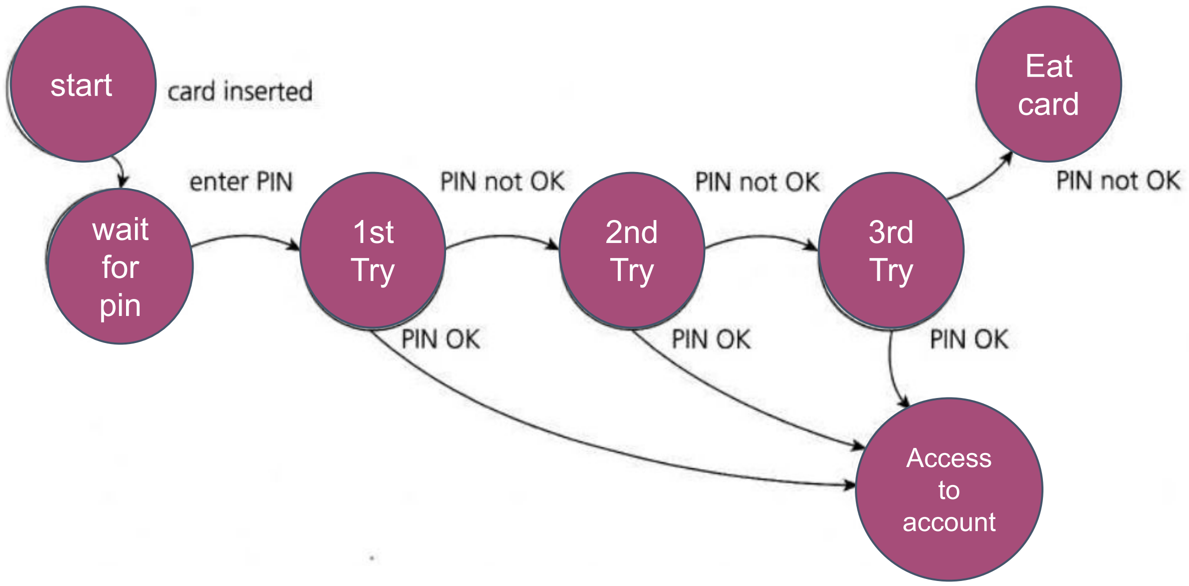


Source : https://www.softwaretestinghelp.com/state-transition-testing-technique-for-testing-complex-applications/

© 2022 e-Learning Centre, UCSC

3.1.4 State Transition Testing

Below state diagram shows an example of entering a Personal Identity Number (PIN) to a bank account. The states are shown as circles, the transitions as lines with arrows and the events as the text near the transitions.



It shows seven states but only four possible events (Card inserted, Enter PIN, PIN OK and PIN not OK).

© 2022 e-Learning Centre, UCSC

3.1.4 State Transition Testing

Test conditions can be derived from the state graph in various ways.

Each state or transition can be noted as a test condition.

Possible test cases from the previous state diagram

- Case 01 : Normal situation, where the correct PIN is entered in the first try.
- Case 02 : Entering an incorrect PIN in each 3 tries and the system eats the card.
- Case 03 : PIN was incorrect the in the first try but OK in the second try, and gets access to the account.

Note that a transition does not need to change to a different state (although all of the transitions shown above do go to a different state). So there could be a transition from ‘access account’ which just goes back to ‘access account’ for an action such as ‘request balance’.

© 2022 e-Learning Centre, UCSC

3.1.4 State Transition Testing

What is a state table?

- Deriving tests only from a state graph or state chart is very good for seeing the valid transitions, but we may not easily see the negative tests, where we try to generate invalid transitions.
- In order to see the total number of combinations of states and transitions, both valid and invalid, a state table is useful.
- The state table lists all the states down one side of the table and all the events that cause transitions along the top (or vice versa).
- The content of each cell indicates which state the system will move to, when the corresponding event occurs while in the associated state.
- This will include possible erroneous events - events that are not expected to happen in certain states. (negative test conditions)

© 2022 e-Learning Centre, UCSC

3.1.4 State Transition Testing

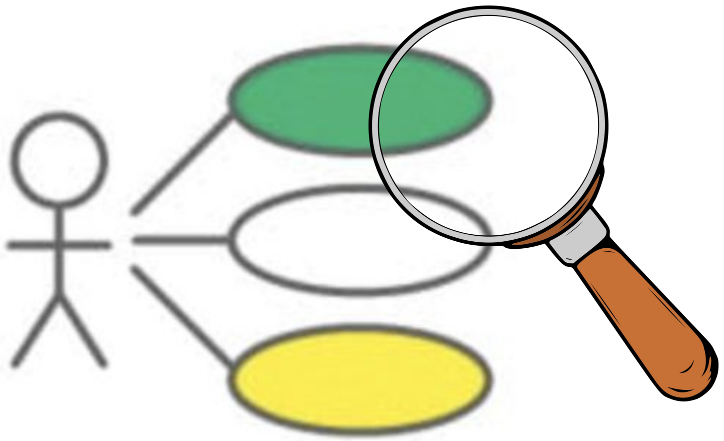
- Following is the state table for the previous ATM example which lists all the states in the first column and the possible inputs across the top row.
- For example, if the system is in State 1, inserting a card will take it to State 2. If we are in State 2, and a valid PIN is entered, we go to State 6 to access the account.
- We have put a dash in the cells that should be impossible, i.e. they represent invalid transitions from that state.

State	Insert card	Valid pin	Invalid PIN
S1) Start state	S2	-	-
S2) Wait for PIN	-	S6	S3
S3) 1st try invalid	-	S6	S4
S4) 2nd try invalid	-	S6	S5
S5) 3rd try invalid	-	-	S7
S6) Access account	-	?	?
S7) Eat card	S1 (For new card)	-	-

© 2022 e-Learning Centre, UCSC

3.1.5 Use Case Testing

- Use case testing is a technique that helps to identify test cases that exercise the whole system on a transaction by transaction basis from start to finish.
- Use case testing helps to identify gaps in software application that might not be found by testing individual software components.



© 2022 e-Learning Centre, UCSC

3.1.5 Use Case Testing

What is a use case?

- A use case is a description of a particular use of the system by an actor (a user of the system).
- Each use case describes the interactions the actor has with the system in order to achieve a specific task
- They serve as the foundation for developing test cases mostly at the system and acceptance testing levels.
- Each use case usually has a mainstream scenario and sometimes additional alternative branches (eg: special cases or exceptional conditions).
- Each use case must specify any preconditions and postconditions that need to be met for the use case to work successfully.

© 2022 e-Learning Centre, UCSC

3.1.5 Use Case Testing

The PIN example that we used for state transition testing could also be defined in terms of a use case, as shown below.

Main success scenario A: Actor S: System	Step	Description
	1	A.Inserts card
	2	S.Validates card and asks for PIN
	3	A.Enters PIN
	4	S.Validates PIN
	5	S.Allows access to account
Extensions	2a	Card not valid S.Display message and reject card
	4a	PIN not valid S.Display message and ask for re-try (twice)
	4b	PIN invalid 3 times S.Eat card and exit

© 2022 e-Learning Centre, UCSC

3.1.5 Use Case Testing

- The use case from PIN example denotes both successful and unsuccessful scenarios.
- In that diagram we can see the interactions between the A (actor) and S (system).
- Step 1 to step 5 is a success scenario as it shows that the card and pin both got validated and it allows the actor to access the account.
- But in extensions there can be seen three other cases as 2a, 4a, 4b.
- For use case testing, we would have a test of the success scenario and one testing for each extension.
- In this example, extension 4b will be given higher priority than 4a from a security point of view.

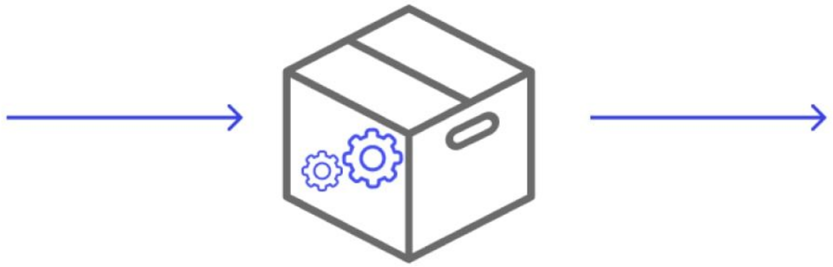
© 2022 e-Learning Centre, UCSC

3.2 White-box Test Techniques

© 2022 e-Learning Centre, UCSC

3.2 White-box Test Techniques

- White-box testing techniques use the internal structure of the software to derive test cases.
- They are commonly called 'glass-box' techniques since they require knowledge of how the software is implemented, that is, how it works.
- These techniques can also be used at all levels of testing.
- White-box test techniques are a good way of generating additional test cases that are different from existing tests.
- Developers use White-box Test Techniques in component testing and component integration testing, especially where there is good tool support for code coverage.



© 2022 e-Learning Centre, UCSC

3.2 White-box Test Techniques

- Characteristics
  - Test conditions, test cases, and test data are derived from a test basis that may include code, software architecture, detailed design, or any other source of information regarding the sthe structure of the software
  - Coverage is measured based on the items tested within a selected structure (e.g. the code or interfaces)
  - Specifications are often used as an additional source of information to determine the expected outcome of test cases.
- Techniques
  - Statement testing and coverage
  - Decision Testing and coverage

© 2022 e-Learning Centre, UCSC

3.2 White-box Test Techniques

What is test coverage?

- Test coverage measures in some specific way the amount of testing performed by a set of tests.
- Wherever we can count things and can tell whether or not each of those things has been tested by some test, then we can measure coverage. The basic coverage measure is

$$\text{Coverage} = \frac{\text{Number of coverage items exercised}}{\text{Total number of coverage items}} \times 100\%$$

- Coverage techniques measure only one dimension of a multi-dimensional concept. i.e. it measures coverage of what has been written, and it cannot say anything about the software that has not been written.

© 2022 e-Learning Centre, UCSC

3.2.1 Statement Testing and Coverage

- Statement Testing is a white box testing technique in which test cases are designed to execute each statement of a program at least once.
- The statement coverage is also known as line coverage or segment coverage.
- Through statement coverage we can identify the statements executed and where the code is not executed because of blockage.
- The statement coverage covers only the true conditions and it can be calculated as shown below:

$$\text{Statement coverage} = \frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100\%$$

© 2022 e-Learning Centre, UCSC

3.2.1 Statement Testing and Coverage

Example

```
READ X
READ Y
IF X>Y
  THEN Z = 0
ENDIF
```

To achieve 100% statement coverage of the above code segment just one test case is required, one which ensures that variable X contains a value that is greater than the value of variable Y

Test Case	Values of X and Y	No of statements executed	Statement coverage

Test 01	X = 5
	Y = 3
		5	5/5*100 = 100%

© 2022 e-Learning Centre, UCSC

3.2.2 Decision Testing and Coverage

- Decision testing is a white-box testing technique which exercises the decisions in the code and tests the code that is executed based on the decision outcomes.
- A decision is an IF statement, a loop control statement (e.g. DO-WHILE or REPEAT-UNTIL), or a CASE statement, where there are two or more possible exits or outcomes from the statement.
- With an IF statement, the exit can either be TRUE or FALSE, depending on the value of the logical condition that comes after IF statement.
- With a loop control statement, the outcome is either to perform the code within the loop or not - again a True or False exit.

© 2022 e-Learning Centre, UCSC

3.2.2 Decision Testing and Coverage

- Decision coverage is calculated by:

$$\text{Decision coverage} = \frac{\text{Number of decision outcomes exercised}}{\text{Total number of decision outcomes}} \times 100\%$$

3.2.2 Decision Testing and Coverage

Example

Earlier in the example provided in statement coverage, only one test case was required to achieve 100% statement coverage.

However, decision coverage requires each decision to have both a True and False outcome. Therefore, to achieve 100% decision coverage, a second test case is necessary where X is less than which ensures that the decision statement 'IF X > Y' has a False outcome.

Test Case	Values of X and Y	No of statements executed	Decision Coverage

Test 01	X = 5
	Y = 3
		5	5/5*100 = 100%

Test 02	X = -2
	Y = 8
		4	4/5*100 = 80%

This now covers both of the decision outcomes, True (with Test 01) and False (with Test 02)

© 2022 e-Learning Centre, UCSC

3.2.3 The Value of Statement and Decision Testing

- Statement coverage helps to find defects in code that was not exercised by other tests.
- When 100% statement coverage is achieved, it ensures that all executable statements in the code have been tested at least once, but it does not ensure that all decision logic has been tested.
- Decision coverage helps to find defects in code where other tests have not taken both true and false outcomes.
- When 100% decision coverage is achieved, it executes all decision outcomes, which includes testing the true outcome and also the false outcome, even when there is no explicit false statement.

© 2022 e-Learning Centre, UCSC

3.2.3 The Value of Statement and Decision Testing

- Out of the two white-box techniques discussed so far, statement testing may provide less coverage than decision testing.
- Achieving 100% decision coverage guarantees 100% statement coverage (but not vice versa).

© 2022 e-Learning Centre, UCSC

3.3 Experience-based Test Techniques

© 2022 e-Learning Centre, UCSC

3.3 Experience-based Test Techniques

- In experience-based test techniques, the test cases are derived from the tester’s skill and intuition, and their experience with similar applications and technologies.
- These techniques can be helpful in identifying tests that were not easily identified by other more systematic techniques.
- Depending on the tester’s approach and experience, these techniques may achieve widely varying degrees of coverage and effectiveness.
- Techniques
  - Error Guessing
  - Exploratory Testing
  - Checklist-based Testing

© 2022 e-Learning Centre, UCSC

3.3.1 Error Guessing

- Error guessing is a technique used to anticipate the occurrence of errors, defects, and failures, based on the tester’s knowledge
- The success of error guessing is very much dependent on the skill of the tester, as good testers know where the defects are most likely to lurk.
- It is a technique that should always be used as a complement to other more formal techniques.
- There are no rules for error guessing. The tester is encouraged to think of situations in which the software may not be able to cope.
- Typical conditions to try include division by zero, blank (or no) input, empty files and the wrong kind of data (e.g. alphabetic characters where numeric are required).

© 2022 e-Learning Centre, UCSC

3.3.1 Error Guessing

- A structured approach to the error-guessing technique is to list possible defects or failures and to design tests that attempt to produce them.
- These defect and failure lists can be built based on the tester's own experience or that of other people, available defect and failure data, and from common knowledge about why software fails.

© 2022 e-Learning Centre, UCSC

3.3.2 Exploratory Testing

- Exploratory testing is a hands-on approach in which testers are involved in minimum planning and maximum test execution.
- The planning involves the creation of a test charter, a short declaration of the scope of a short 1 to 2 hour) time-boxed test effort, the objectives and possible approaches to be used.
- The test design and test execution activities are performed in parallel typically without formally documenting the test conditions, test cases or test scripts.
- A key aspect of exploratory testing is learning where tester explores and learns about the software, its use, its strengths and its weaknesses.

© 2022 e-Learning Centre, UCSC

3.3.2 Exploratory Testing

- The tester is constantly making decisions about what to test next and where to spend the (limited) time.
- This is an approach that is most useful when there are no or poor specifications and when time is severely limited.
- It can also serve to complement other, more formal testing, helping to establish greater confidence in the software
- It can be used as a check on the formal test process by helping to ensure that the most serious defects have been found.

© 2022 e-Learning Centre, UCSC

3.3.1 Checklist-based Testing

- In checklist-based testing, testers design, implement, and execute tests to cover test conditions found in a checklist.
- As part of analysis, testers create a new checklist or expand an existing checklist, but testers may also use an existing checklist without modification.
- Such checklists can be built based on experience, knowledge about what is important for the user, or an understanding of why and how software fails.
- In the absence of detailed test cases, checklist-based testing can provide guidelines and a degree of consistency.
- As these are high-level lists, some variability in the actual testing is likely to occur, resulting in potentially greater coverage but less repeatability.

© 2022 e-Learning Centre, UCSC

Summary

- There are three types of test design techniques as; Black-box testing, White box testing and Experience based techniques. All categories are useful and the three are complementary.
- Black-box test techniques are based on an analysis of the appropriate test basis (e.g., formal requirements documents, specifications, use cases, user stories, or business processes).
- There are various test case design techniques applied for black-box testing: equivalence partitioning, boundary value analysis, decision tables, state transition testing and use case testing
- Black-box test techniques are appropriate at all levels of testing (component testing through to acceptance testing) where a specification exists.

© 2022 e-Learning Centre, UCSC

Summary

- White-box test techniques are based on an analysis of the architecture, detailed design, internal structure, or the code of the test object.
- White-box test techniques can also be used at all levels of testing.
- The techniques used to derive test cases in white box testing are statement coverage and decision coverage.

© 2022 e-Learning Centre, UCSC

Summary

- Experience-based test techniques leverage the experience of developers, testers and users to design, implement, and execute tests.
- Experience-based techniques are used to complement white box and black box techniques, and are also used when there is no specification, or if the specification is inadequate or out of date.
- Types of experienced based testing are error guessing, exploratory testing and checklist-based testing.

© 2022 e-Learning Centre, UCSC