SE-3002
SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR
RUBAB.JAFFAR@NU.EDU.PK

Part II-Software Testing

Functional Testing

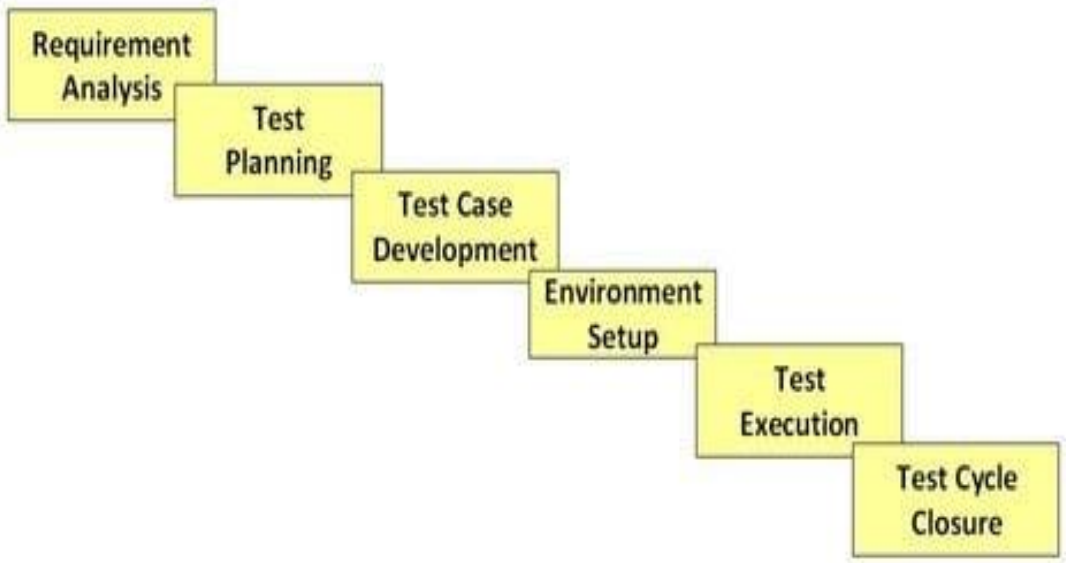
Lecture # 13, 14, 15

TODAY'S OUTLINE

- STLC
- Functional testing
- Boundary value analysis
- Equivalence class testing
- Revision

STLC (SOFTWARE TESTING LIFE CYCLE)

- A sequence of specific activities conducted during the testing process to ensure software quality goals are met. STLC involves both verification and validation activities.
- Each of these stages has a definite Entry and Exit criteria, Activities & Deliverables associated with it.
- **Entry Criteria:** Entry Criteria gives the prerequisite items that must be completed before testing can begin.
- **Exit Criteria:** Exit Criteria defines the items that must be completed before testing can be concluded



REQUIREMENT PLANNING PHASE

- Requirement Phase Testing also known as Requirement Analysis in which test team studies the requirements from a testing point of view to identify testable requirements and the QA team may interact with various stakeholders to understand requirements in detail.
- Requirements could be either functional or non-functional.
- Activities in Requirement Phase Testing
  - Identify types of tests to be performed.
  - Gather details about testing priorities and focus.
  - Prepare Requirement Traceability Matrix (RTM).
  - Identify test environment details where testing is supposed to be carried out.
- Deliverables of Requirement Phase Testing
  - RTM

Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

TEST PLANNING

- Test Planning in STLC is a phase in which a Senior QA manager determines the test plan strategy along with efforts and cost estimates for the project. Moreover, the resources, test environment, test limitations and the testing schedule are also determined. The Test Plan gets prepared and finalized in the same phase.
- Test Planning Activities
  - Preparation of test plan/strategy document for various types of testing
  - Test tool selection
  - Test effort estimation
  - Resource planning and determining roles and responsibilities.
  - Training requirement
- Deliverables of Test Planning
  - Test plan/strategy document.
  - Effort estimation document.

TEST CASE DEVELOPMENT PHASE

- The Test Case Development Phase involves the creation, verification and rework of test cases & test scripts after the test plan is ready. Initially, the Test data is identified then created and reviewed and then reworked based on the preconditions. Then the QA team starts the development process of test cases for individual units.

- Test Case Development Activities
  - Create test cases, automation scripts (if applicable)
  - Review and baseline test cases and scripts
  - Create test data (If Test Environment is available)
- Deliverables of Test Case Development
  - Test cases/scripts
  - Test data

TEST ENVIRONMENT SETUP

- Test Environment Setup decides the software and hardware conditions under which a work product is tested. It is one of the critical aspects of the testing process and can be done in parallel with the Test Case Development Phase. Test team may not be involved in this activity if the development team provides the test environment. The test team is required to do a readiness check (smoke testing) of the given environment.
- Test Environment Setup Activities
  - Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
  - Setup test Environment and test data
  - Perform smoke test on the build
- Deliverables of Test Environment Setup
  - Environment ready with test data set up
  - Smoke Test Results.

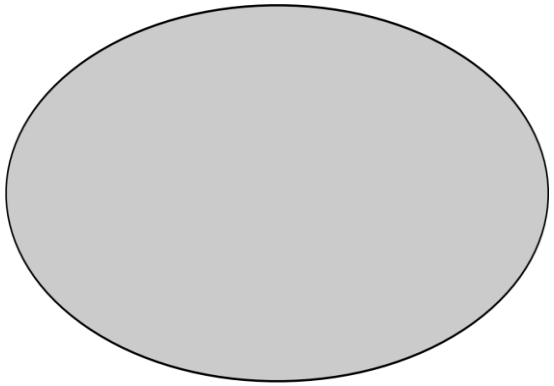
TEST EXECUTION PHASE

- **Test Execution Phase** is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting. If bugs are reported then it is reverted back to development team for correction and retesting will be performed.
- **Test Execution Activities**
  - Execute tests as per plan
  - Document test results, and log defects for failed cases
  - Map defects to test cases in RTM
  - Retest the Defect fixes
  - Track the defects to closure
- **Deliverables of Test Execution**
  - Completed RTM with the execution status
  - Test cases updated with results
  - Defect reports

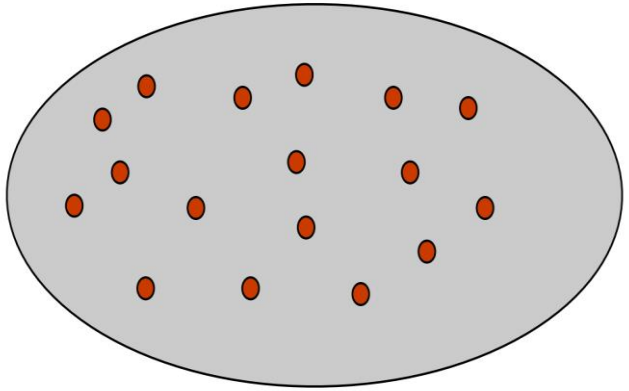
TEST CYCLE CLOSURE

- **Test Cycle Closure** phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.
- **Test Cycle Closure Activities**
  - Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality
  - Prepare test metrics based on the above parameters.
  - Document the learning out of the project
  - Prepare Test closure report
  - Qualitative and quantitative reporting of quality of the work product to the customer.
  - Test result analysis to find out the defect distribution by type and severity.
- **Deliverables of Test Cycle Closure**
  - Test Closure report
  - Test metrics

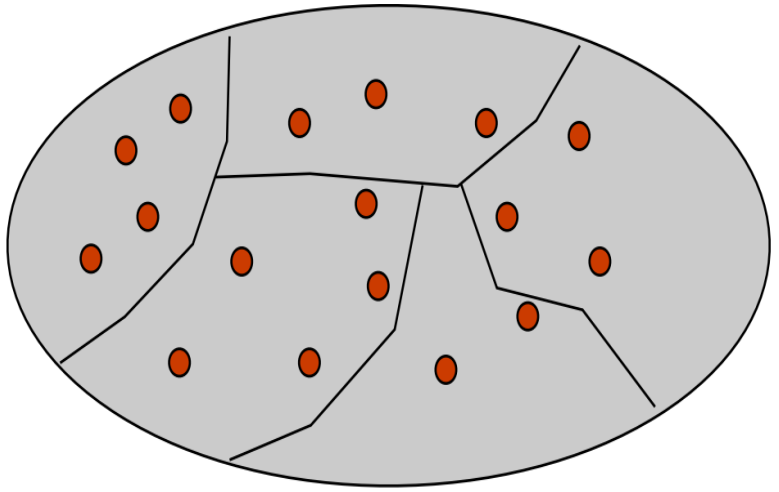
FUNCTIONAL TESTING



All possible states/behaviors of a system



Tests are a way of sampling the behaviors of a software system, looking for failures



The testing literature advocates folding the space into equivalent behaviors and then sampling each partition

FUNCTIONAL TESTING

- • Consider a simple example like the greatest common denominator function

  •int gcd(int x, int y)

  • At first glance, this function has an infinite number of test cases

  • But lets fold the space

  • x=6 y=9, returns 3, tests common case

  • x=2 y=4, returns 2, tests when x is the GCD

  • x=3 y=5, returns 1, tests two primes

  • x=9 y=0, returns ?, tests zero

  • x=-3 y=9, returns ?, tests negative

Completeness

From this discussion, it should be clear that "completely" testing a system is impossible

• So, we settle for heuristics

• attempt to fold the input space into different functional categories

• then create tests that sample the behavior/output for each functional partition

• As we will see, we also look at our coverage of the underlying code; are we hitting all statements, all branches, all loops?

LIMITATIONS OF TESTING

- Test everything before giving the software to the customers.
- 'EVERYTHING'
  - Execute every true and false condition
  - Execute every statement of the program
  - Execute every condition of a decision node
  - Execute every possible path
  - Execute the program with all valid inputs
  - Execute the program with all invalid inputs

- **impossible to achieve due to time and resource constraints**

FUNCTIONAL TESTING

- Main intent of software testing is to search of such test cases which may make the software fail.
- Functional testing techniques attempt to design those test cases which have a higher probability of making a software fail.
- These techniques also attempt to test every possible functionality of the software.
- Test cases are designed on the basis of functionality and the internal structure of the program is completely ignored.

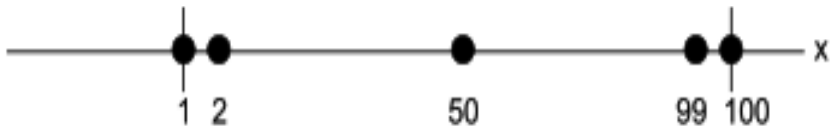
FUNCTIONAL TESTING

- Real life testing activities performed with only black box knowledge???
- In functional testing techniques, execution of a program is essential and hence these testing techniques come under the category of 'validation'.
- These techniques can be used at all levels of software testing like unit, integration, system and acceptance testing.

BOUNDARY VALUE ANALYSIS

- Popular functional testing technique that concentrate on input values and design test cases with input values that are on or close to boundary values.
- Write a program 'Square' which takes 'x' as an input and prints the square of 'x' as output. The range of 'x' is from 1 to 100.
- How to test this program???
- One possibility is to give all values from 1 to 100 one by one to the program and see the observed behavior.
- In boundary value analysis, we select values on or close to boundaries and all input values may have one of the following:
  - Just above minimum value
  - Minimum value
  - Maximum value
  - Just below maximum value
  - Nominal (Average) value

BOUNDARY VALUE ANALYSIS FOR SQUARE PROGRAM



Five values for input 'x' of 'Square' program

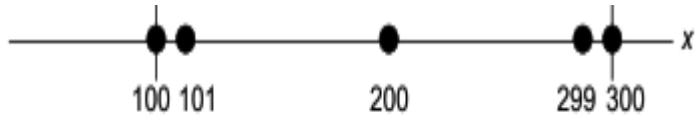
Test cases for the 'Square' program

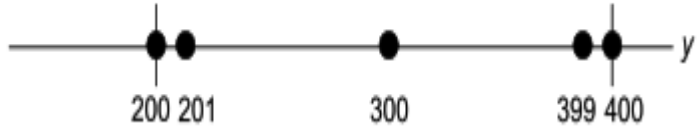
Test Case	Input x	Expected output
1.	1	1
2.	2	4
3.	50	2500
4.	99	9801
5.	100	10000

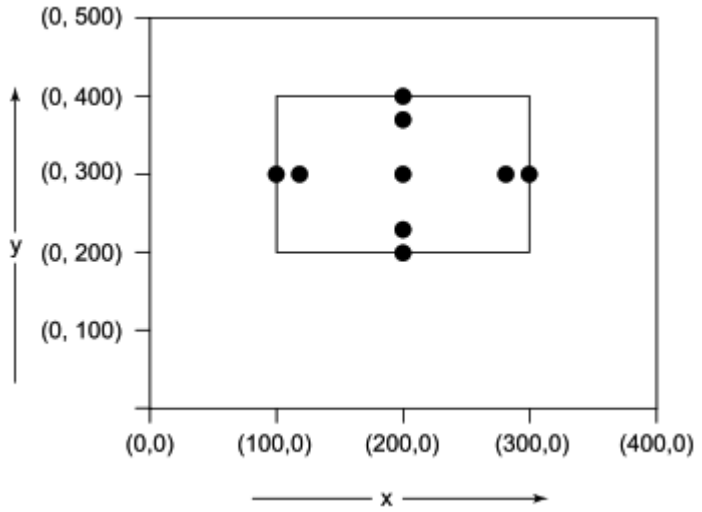
- The number of inputs selected by this technique is 4n + 1 where 'n' is the number of inputs.

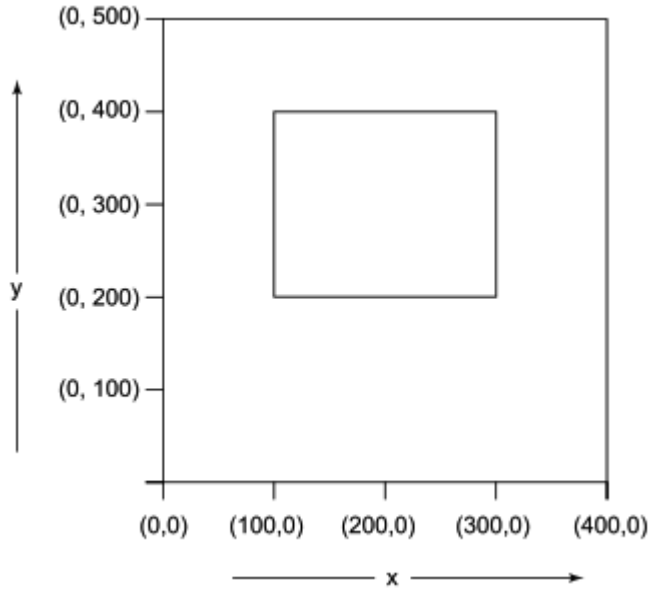
ANOTHER EXAMPLE

- Consider a program 'Addition' with two input values x and y and it gives the addition of x and y as an output. The range of both input values are given as:
  - 100<= x <= 300
  - 200 <= y <= 400









Test cases for the program 'Addition'

Test Case	x	y	Expected Output
1.	100	300	400
2.	101	300	401
3.	200	300	500
4.	299	300	599
5.	300	300	600
6.	200	200	400
7.	200	201	401
8.	200	300	500
9.	200	399	599
10.	200	400	600

APPLICABILITY

- This technique is suited to programs in which input values are within ranges or within sets and input values boundaries can be identified from the requirements.
- This is equally applicable at the unit, integration, system and acceptance test levels.
- This technique does not make sense for Boolean variables where input values are TRUE and FALSE only, and no choice is available for nominal values, just above boundary values, just below boundary values, etc.

EQUIVALENCE CLASS TESTING

- A large number of test cases are generated for any program. It is neither feasible nor desirable to execute all such test cases.
- We want to select a few test cases and still wish to achieve a reasonable level of coverage.
- We may divide input domain into various categories with some relationship and expect that every test case from a category exhibits the same behavior.
- If categories are well selected, we may assume that if one representative test case works correctly, others may also give the same results. This assumption allows us to select exactly one test case from each category and if there are four categories, four test cases may be selected.
- Each category is called an equivalence class and this type of testing is known as equivalence class testing.

CREATION OF EQUIVALENCE CLASSES

- The entire input domain can be divided into at least two equivalence classes: one containing all valid inputs and the other containing all invalid inputs.
- Each equivalence class can further be sub-divided into equivalence classes on which the program is required to behave differently.
- Square program

(i) $I_1 = \{ 1 \leq x \leq 100 \}$ (Valid input range from 1 to 100)
(ii) $I_2 = \{ x < 1 \}$ (Any invalid input where x is less than 1)
(iii) $I_3 = \{ x > 100 \}$ (Any invalid input where x is greater than 100)

Test cases for program 'Square' based on input domain

Test Case	Input x	Expected Output
I_1	0	Invalid Input
I_2	50	2500
I_3	101	Invalid Input

EQUIVALENCE CLASS TESTING

- A large number of test cases are generated for any program. It is neither feasible nor desirable to execute all such test cases.
- Select a few test cases and still wish to achieve a reasonable level of coverage.
- Divide input domain into various categories with some relationship and expect that every test case from a category exhibits the same behavior.
- If categories are well selected, we may assume that if one representative test case works correctly, others may also give the same results. This assumption allows us to select exactly one test case from each category and if there are four categories, four test cases may be selected.
- Each category is called an equivalence class and this type of testing is known as equivalence class testing.

CREATION OF EQUIVALENCE CLASSES

- The entire input domain can be divided into at least two equivalence classes: one containing all valid inputs and the other containing all invalid inputs.
- Each equivalence class can further be sub-divided into equivalence classes on which the program is required to behave differently.
- Square program

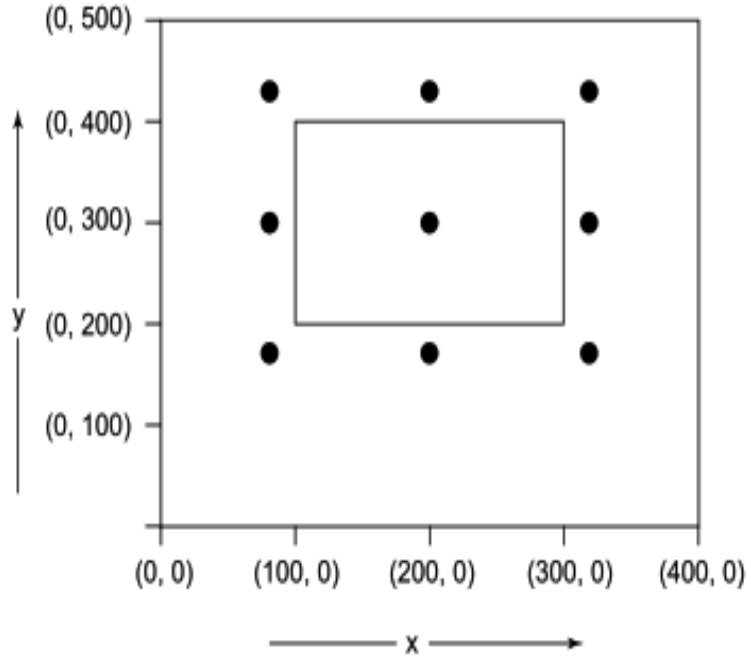
(i) $I_1 = \{ 1 \leq x \leq 100 \}$ (Valid input range from 1 to 100)
(ii) $I_2 = \{ x < 1 \}$ (Any invalid input where x is less than 1)
(iii) $I_3 = \{ x > 100 \}$ (Any invalid input where x is greater than 100)

Test cases for program 'Square' based on input domain

Test Case	Input x	Expected Output
I_1	0	Invalid Input
I_2	50	2500
I_3	101	Invalid Input

EQUIVALENCE CLASSES FOR ADDITION PROGRAM

(i) $I_1 = \{ 100 \leq x \leq 300 \text{ and } 200 \leq y \leq 400 \}$ (Both x and y are valid values)
(ii) $I_2 = \{ 100 \leq x \leq 300 \text{ and } y < 200 \}$ (x is valid and y is invalid)
(iii) $I_3 = \{ 100 \leq x \leq 300 \text{ and } y > 400 \}$ (x is valid and y is invalid)
(iv) $I_4 = \{ x < 100 \text{ and } 200 \leq y \leq 400 \}$ (x is invalid and y is valid)
(v) $I_5 = \{ x > 300 \text{ and } 200 \leq y \leq 400 \}$ (x is invalid and y is valid)
(vi) $I_6 = \{ x < 100 \text{ and } y < 200 \}$ (Both inputs are invalid)
(vii) $I_7 = \{ x < 100 \text{ and } y > 400 \}$ (Both inputs are invalid)
(viii) $I_8 = \{ x > 300 \text{ and } y < 200 \}$ (Both inputs are invalid)
(ix) $I_9 = \{ x > 300 \text{ and } y > 400 \}$ (Both inputs are invalid)



TEST CASES FOR ADDITION

Test cases for the program 'Addition'

Test Case	x	y	Expected Output
I_1	200	300	500
I_2	200	199	Invalid input
I^3	200	401	Invalid input
I_4	99	300	Invalid input
I_5	301	300	Invalid input
I_6	99	199	Invalid input
I_7	99	401	Invalid input
I_8	301	199	Invalid input
I_9	301	401	Invalid input

APPLICABILITY

- Applicable at unit, integration, system and acceptance test levels.
- The basic requirement is that inputs or outputs must be partitioned based on the requirements and every partition will give a test case.
- If one test case catches a bug, the other probably will too. If one test case does not find a bug, the other test cases of the same equivalence class may also not find any bug.
- The design of equivalence classes is subjective and two testing persons may design two different sets of partitions of input and output domains.

EXAMPLE

- Consider a program for the determination of the largest amongst three numbers. Its input is a triple of positive integers (say x, y and z) and values are from interval [1, 300].
- Design the boundary value test cases.
- Design the equivalence classes

That is all