SE-3002
SOFTWARE QUALITY ENGINEERING

RUBAB JAFFAR
RUBAB.JAFFAR@NU.EDU.PK

Part II-Software Testing

Software Testing

Lecture # 10,11, 12

TODAY'S OUTLINE

- Software testing
- Testing levels
- Testing strategies
- Limitations of testing
- Functional testing
- Boundary value analysis

ENGINEERING PROCESS

- Sophisticated tools
  - amplify capabilities
  - but do not remove human error
- Engineering disciplines pair
  - construction activities with
  - activities that check intermediate and final products
- Software engineering is no exception:
- construction of high quality software requires
  - construction and
  - verification activities

TESTING

- Testing is a critical element of software development life cycles
- Often termed as software quality control or software quality assurance
- basic goals: validation and verification
- validation: are we building the right product?
- verification: does "X" meet its specification?
  - where "X" can be code, a model, a design diagram, a requirement, …
- At each stage, we need to verify that the thing we produce accurately represents its specification

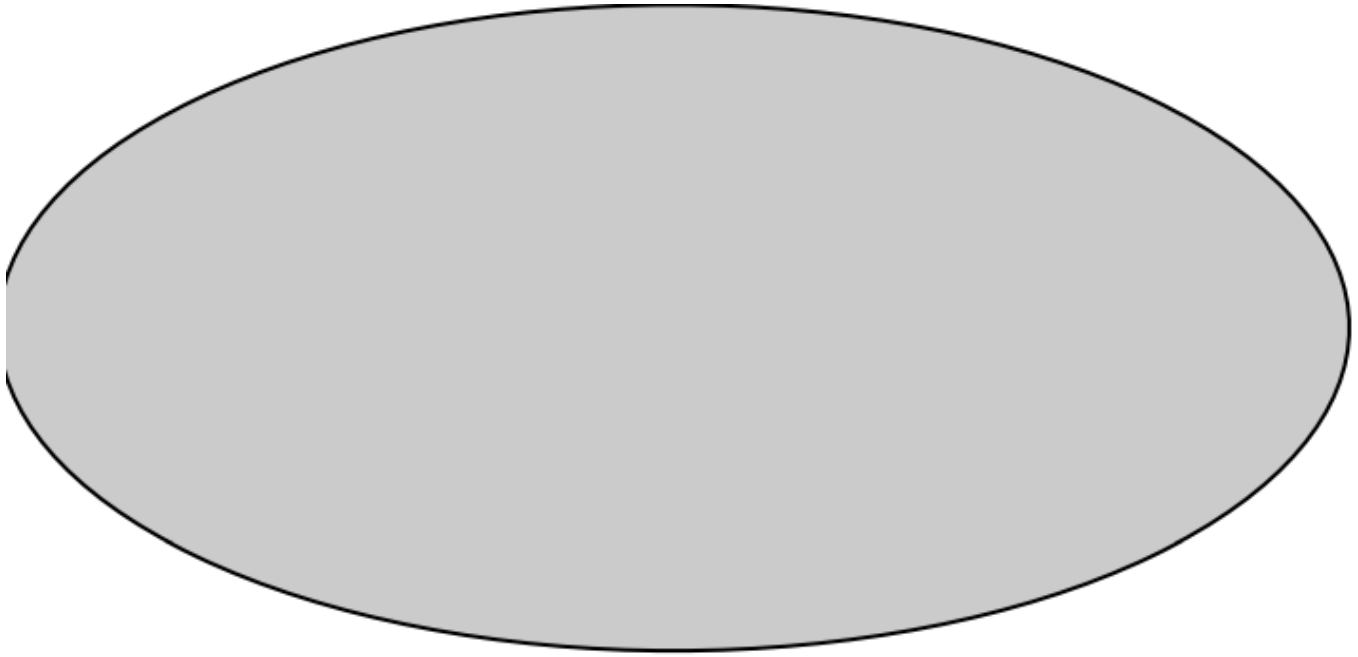
SOME TERMINOLOGIES

- An error
  - often a misunderstanding of a requirement or design specification
- A fault
- A failure
- The failure may occur immediately (crash!) or much, much later in the execution
- Testing attempts to surface failures in our software systems
- Debugging attempts to associate failures with faults so they can be removed from the system
  - If a system passes all of its tests, is it free of all faults?

NO!!

- •Faults may be hiding in portions of the code that only rarely get executed
  - "Testing can only be used to prove the existence of faults not their absence" or "Not all faults have failures"
- Sometimes faults mask each other resulting in no visible failures!
- However, if we do a good job in creating a test set that
  - covers all functional capabilities of a system
  - and covers all code using a metric such as "branch coverage"
- Then, having all tests pass increases our confidence that our system has high quality and can be deployed

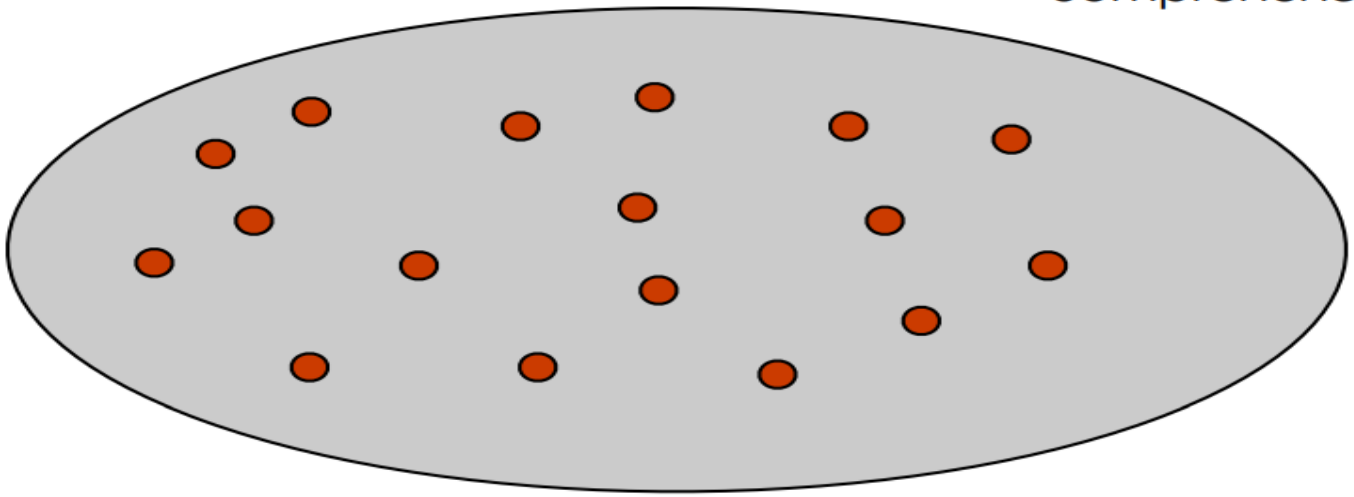
LOOKING FOR FAULTS



All possible states/behaviors of a system

LOOKING FOR FAULTS

As you can see, its not very comprehensive

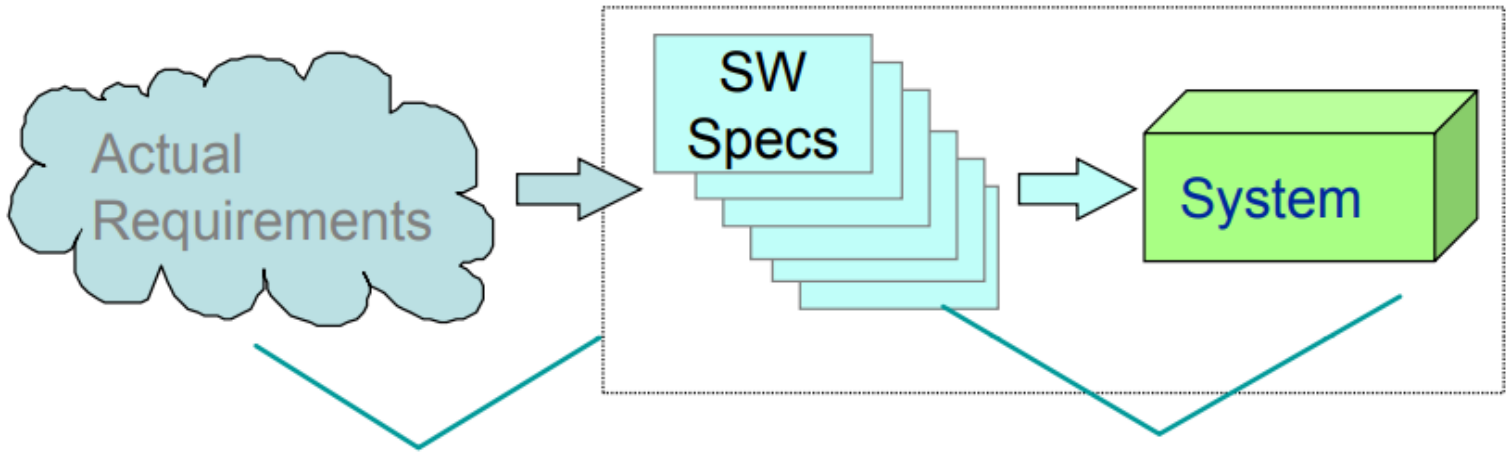


Tests are a way of sampling the behaviors of a software system, looking for failures

PECULIARITIES OF SOFTWARE

- Software has some characteristics that make V&V particularly difficult:
  - Many different quality requirements
  - Evolving (and deteriorating) structure
  - Uneven distribution of faults
- Example
- If an elevator can safely carry a load of 1000 kg, it can also safely carry a smaller load;
- If a procedure correctly sorts a set of 256 elements, it may fail on a set of 255 or 53 or 12 elements, as well as on 257 or 1023.

VALIDATION AND VERIFICATION



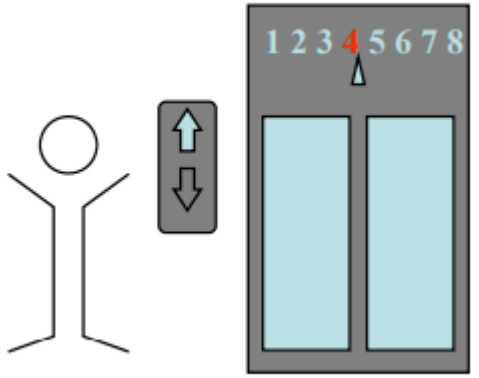
Validation

Includes usability testing, user feedback

Verification

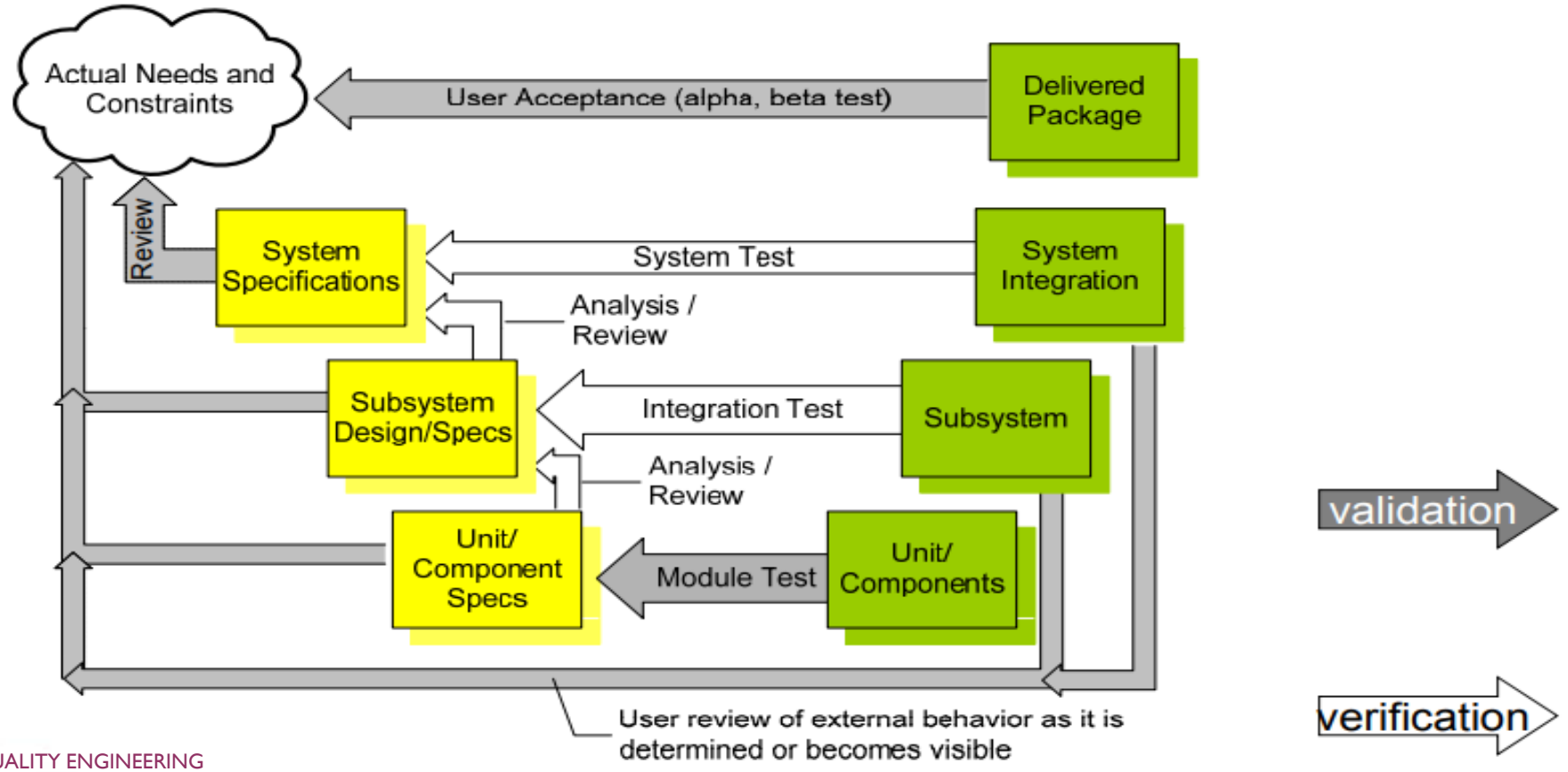
Includes testing, inspections, static analysis

VERIFICATION OR VALIDATION DEPENDS ON THE SPECIFICATION

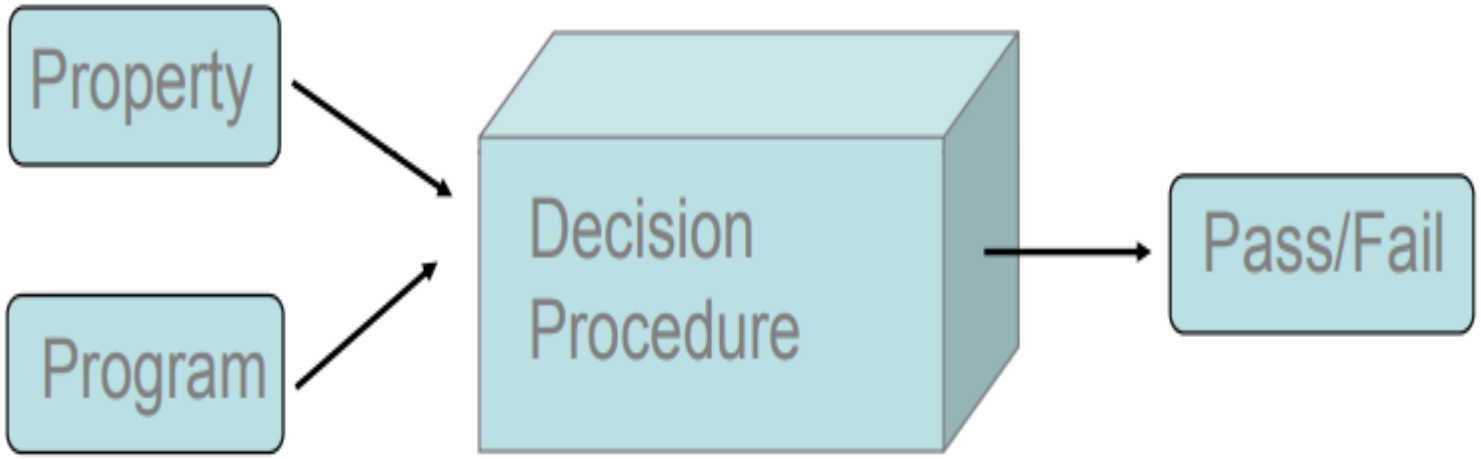


- Example: elevator response
- Unverifiable (but validatable) spec: ...
- if a user presses a request button at floor i, an available elevator must arrive at floor i soon...
- Verifiable spec: ...
- if a user presses a request button at floor i, an available elevator must arrive at floor i within 30 seconds...

VALIDATION AND VERIFICATION ACTIVITIES



You can't ~~always~~ *ever* get what you want



Correctness properties are undecidable

the halting problem can be embedded in almost every property of interest

VARIETY OF APPROACHES FOR TESTING

- There are no fixed recipes
- Test designers must
  - choose and schedule the right blend of techniques
    - to reach the required level of quality
    - within cost constraints
- – design a specific solution that suits
  - the problem
  - the requirements
  - the development environment

FIVE BASIC QUESTIONS

- When do verification and validation start? When are they complete?
- What particular techniques should be applied during development?
- How can we assess the readiness of a product?
- How can we control the quality of successive releases?
- How can the development process itself be improved?

WHEN DO VERIFICATION AND VALIDATION START? WHEN ARE THEY COMPLETE?

- Test is not a (late) phase of software development
  - Execution of tests is a small part of the verification and validation process
- V&V start as soon as we decide to build a software product, or even before
- V&V last far beyond the product delivery as long as the software is in use, to cope with evolution and adaptations to new conditions

EARLY START: FROM FEASIBILITY STUDY

- The feasibility study of a new project must take into account the required qualities and their impact on the overall cost
- At this stage, quality related activities include
  - risk analysis
  - measures needed to assess and control quality at each stage of development.
  - assessment of the impact of new features and new quality requirements
  - contribution of quality control activities to development cost and schedule

LONG LASTING: BEYOND MAINTENANCE

- Maintenance activities include
  - analysis of changes and extensions
  - generation of new test suites for the added functionalities
  - re-executions of tests to check for non regression of software functionalities after changes and extensions
  - fault tracking and analysis

WHAT PARTICULAR TECHNIQUES SHOULD BE APPLIED DURING DEVELOPMENT?

- No single A&T technique can serve all purposes
- The primary reasons for combining techniques are:
  - Effectiveness for different classes of faults
  - Applicability at different points in a project
    - Example: inspection for early requirements validation
  - Differences in purpose
    - example: statistical testing to measure reliability
  - Tradeoffs in cost and assurance
    - example: expensive technique for key properties

HOW CAN WE ASSESS THE READINESS OF A PRODUCT?

- A&T during development aim at revealing faults
- We cannot reveal or remove all faults
- A&T cannot last indefinitely: we want to know if products meet the quality requirements
- We must specify the required level of dependability and determine when that level has been attained.

MEASURES OF DEPENDABILITY

- Availability measures the quality of service in terms of running versus down time
- Mean time between failures (MTBF) measures the quality of the service in terms of time between failures
- Reliability indicates the fraction of all attempted operations that complete successfully

EXAMPLE OF DIFFERENT DEPENDABILITY MEASURES

- Web application:
- 50 interactions terminating with a credit card charge.
- The software always operates flawlessly up to the point that a credit card is to be charged, but on half the attempts it charges the wrong amount.
- What is the reliability of the system?
- If we count the fraction of individual interactions that are correctly carried, only one operation in 100 fail: The system is 99% reliable.
- If we count entire sessions, only 50% reliable, since half the sessions result in an improper credit card charge

ASSESSING DEPENDABILITY

- Randomly generated tests following an operational profile
- Alpha test : tests performed by users in a controlled environment, observed by the development organization
- Beta test: tests performed by real users in their own environment, performing actual tasks without interference or close monitoring

HOW CAN WE CONTROL THE QUALITY OF SUCCESSIVE RELEASES?

- Software test and analysis does not stop at the first release.
- Software products operate for many years, and undergo many changes:
  - They adapt to environment changes
  - They evolve to serve new and changing user requirements.
- Quality tasks after delivery
  - test and analysis of new and modified code
  - re-execution of system tests
  - extensive record-keeping

HOW CAN THE DEVELOPMENT PROCESS ITSELF BE IMPROVED?

- The same defects are encountered in project after project
- An important goal of the improving the quality process is to improve the process by
  - identifying and removing weaknesses in the development process
  - identifying and removing weaknesses in test and analysis that allow them to remain undetected

A FOUR STEP PROCESS TO IMPROVE FAULT ANALYSIS AND PROCESS

- Define the data to be collected and implementing procedures for collecting them
- Analyze collected data to identify important fault classes
- Analyze selected fault classes to identify weaknesses in development and quality measures
- Adjust the quality and development process

AN EXAMPLE OF PROCESS IMPROVEMENT

- Faults that affect security were given highest priority
- During A&T we identified several buffer overflow problems that may affect security
- Faults were due to bad programming practice and were revealed late due to lack of analysis
- Action plan: Modify programming discipline and environment and add specific entries to inspection checklists

TEST, TEST CASE AND TEST SUITE

- Test and test case terms are synonyms and may be used interchangeably
- A test case consists of inputs given to the program and its expected outputs. Inputs may also contain
  - precondition(s) (circumstances that hold prior to test case execution), if any, and
  - actual inputs identified by some testing methods. Expected output may contain post-condition(s) (circumstances after the execution of a test case), if any, and outputs which may come as a result when selected inputs are given to the software.
- Every test case will have a unique identification number.
- During testing, we give selected inputs to the program and note the observed output(s).
- Comparison of observed output(s) with the expected output(s)
  - if they are the same, the test case is successful.
  - If they are different, that is the failure condition with selected input(s) and this should be recorded properly in order to find the cause of failure.

TEST CASE TEMPLATE

Table 1.7. Test case template

Test Case Identification Number:

Part I (Before Execution)

1.	Purpose of test case:
2.	Pre-condition(s): (optional)
3.	Input(s) :
4.	Expected Output(s) :
5.	Post-condition(s) :
6.	Written by :
7.	Date of design :

Part II (After Execution)

1.	Output(s) :
2.	Post-condition(s) : (optional)

TEST CASE TEMPLATE

Part II (After Execution)

3. Pass / fail :
4. If fails, any possible reason of failure (optional) :
5. Suggestions (optional)
6. Run by :
7. Date of suggestion :

TEST SUITE

- The set of test cases is called a test suite.
  - test suite of all test cases,
  - test suite of all successful test cases and
  - test suite of all unsuccessful test cases.
  - All test suites should be preserved as we preserve source code and other documents.

TESTING THE SYSTEM (I)

- Unit Tests
  - Tests that cover low-level aspects of a system
  - For each module, does each operation perform as expected
  - For method foo(), we'd like to see another method testFoo()
- Integration Tests
  - Tests that check that modules work together in combination
  - Most projects on schedule until they hit this point
  - All sorts of hidden assumptions are surfaced when code written by different developers are used in tandem.
  - Lack of integration testing has led to spectacular failures

TESTING THE SYSTEM (II)

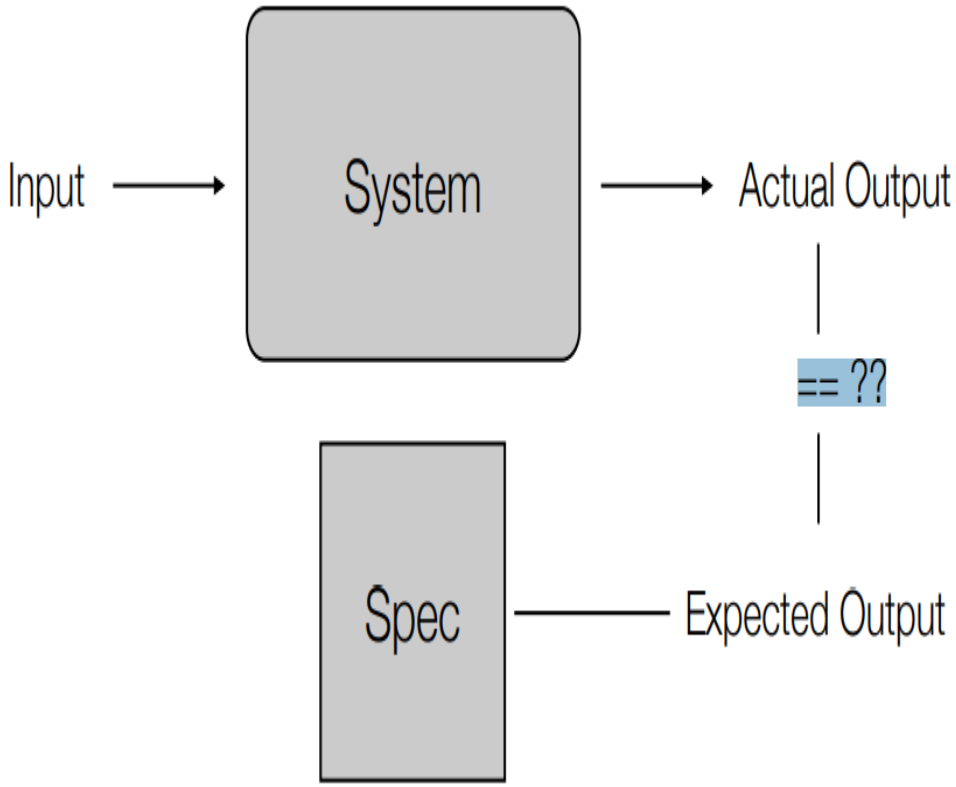
- System Tests
  - Tests performed by the developer to ensure that all major functionality has been implemented
  - Have all user stories been implemented and function correctly?
- Acceptance Tests
  - Tests performed by the user to check that the delivered system meets their needs
  - In large, custom projects, developers will be on-site to install system and then respond to problems as they arise

MULTI-LEVEL TESTING

- Once we have code, we can perform three types of tests
- Black Box Testing
  - Does the system behave as predicted by its specification
- Grey Box Testing
  - Having a bit of insight into the architecture of the system, does it behave as predicted by its specification
- White Box Testing
  - Since, we have access to most of the code, lets make sure we are covering all aspects of the code: statements, branches, …

BLACK BOX TESTING

- A black box test passes input to a system, records the actual output and compares it to the expected output.
- if actual output == expected output
  - TEST PASSED
- else
  - TEST FAILED
- Process
  - Write at least one test case per functional capability
  - Iterate on code until all tests pass
  - Need to automate this process as much as possible



GREY BOX TESTING

- Use knowledge of system's architecture to create a more complete set of black box tests
- Verifying auditing and logging information
  - for each function is the system really updating all internal state correctly
- System-added information (timestamps, checksums, etc.)
- "Looking for Scraps"
  - Is the system correctly cleaning up after itself
  - temporary files, memory leaks, data duplication/deletion

WHITE BOX TESTING

- Writing test cases with complete knowledge of code
  - Format is the same: input, expected output, actual output
- But, now we are looking at
  - code coverage
  - proper error handling
  - working as documented (is method "foo" thread safe?)
  - proper handling of resources
    - how does the software behave when resources become constrained?

TESTING, QUALITY ASSURANCE AND QUALITY CONTROL

- Most of us feel that these terms are similar and may be used interchangeably.
- The purpose of testing is to find faults and find them in the early phases of software development. We remove faults and ensure the correctness of removal and also minimize the effect of change on other parts of the software.
- The purpose of QA activity is to enforce standards and techniques to improve the development process and prevent the previous faults from ever occurring. The QA group monitors and guides throughout the software development life cycle. This is a defect prevention technique.
  - Examples are reviews, audits, etc.
- Quality control attempts to build a software system and test it thoroughly. If failures are experienced, it removes the cause of failures and ensures the correctness of removal. It concentrates on specific products rather than processes as in the case of QA. This is a defect detection and correction activity which is usually done after the completion of the software development.
  - An example is software testing at various levels.

STATIC AND DYNAMIC TESTING

- Static testing refers to testing activities without executing the source code. All verification activities like inspections, walkthroughs, reviews, etc. come under this category of testing.
- Dynamic testing refers to executing the source code and seeing how it performs with specific inputs. All validation activities come in this category where execution of the program is essential.

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.

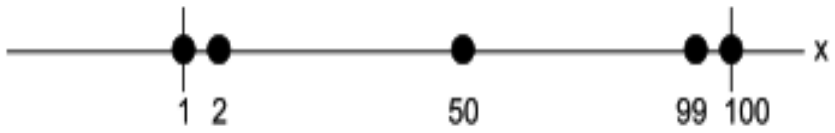
FUNCTIONAL TESTING

- 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

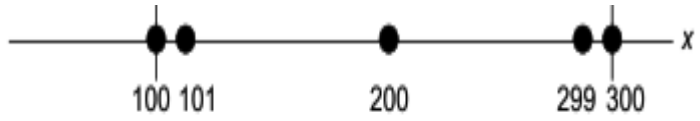
Test cases for the 'Square' program

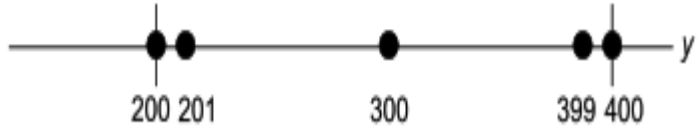
Test Case	Input x	Expected output
1.	1	1
2.	2	4
3.	50	2500
4.	99	9801
5.	100	10000

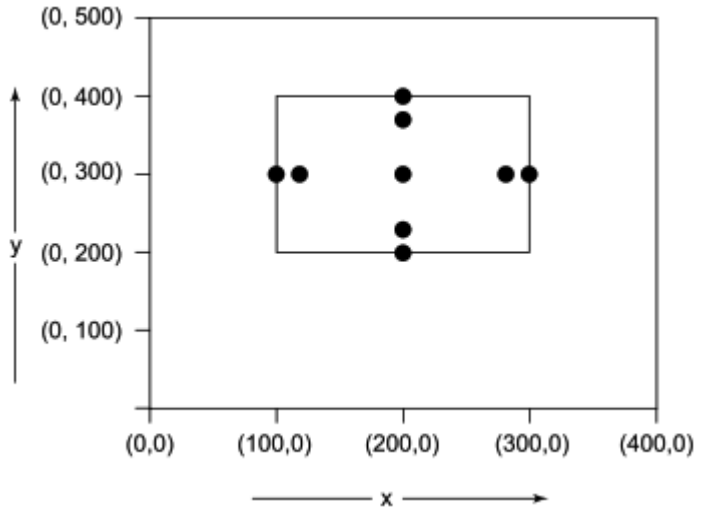
- The number of inputs selected by this technique is 4n + 1 where 'n' is the number of inputs.

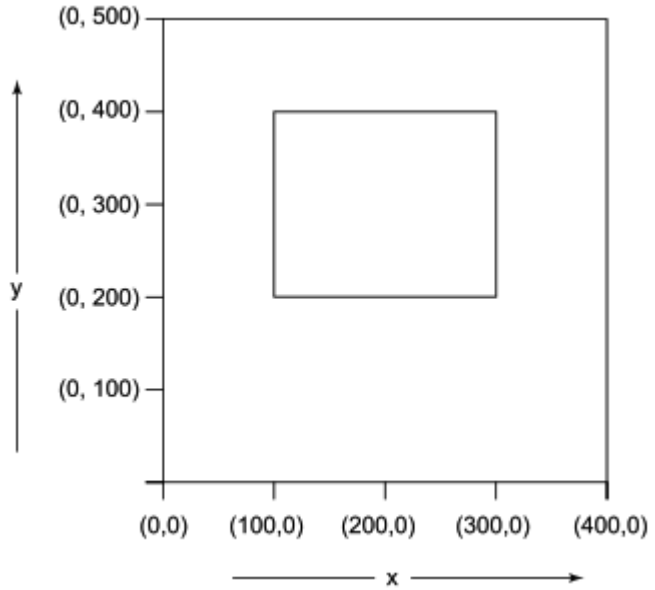
ANOTHER EXAMPLE

- 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.

YOUR TURN:

- Suppose you are working as a quality engineer in a software company. You are given the task for designing the test cases for an application program "Next Date" that takes three inputs, such as month, date, and year. It displays the date of the next day. The current date is used as the input date.
- The parameters are
- C1: 1 ≤ month ≤ 12
- C2: 1 ≤ day ≤ 30
- C3: 1900 ≤ year ≤ 2025.
- The Next Date program takes date as input and checks it for validity. If it is valid, it returns the next date as its output. This function outputs "Invalid Input Date" if any one of conditions C1, C2, or C3 fails.
- The test cases should be designed by using the boundary value analysis. How many test cases will be required to exercise the program functionality at the boundaries?

That is all