



Black box Testing

Design of Test Cases

- Exhaustive testing of any non-trivial system is impractical:
 - input data domain is extremely large.
- Design an **optimal test suite**:
 - of reasonable size and
 - uncovers as many errors as possible.

Design of Test Cases

- If test cases are selected randomly:
 - many test cases would not contribute to the significance of the test suite,
 - would not detect errors not already being detected by other test cases in the suite.
- Number of test cases in a randomly selected test suite:
 - not an indication of effectiveness of testing.

Design of Test Cases



- Testing a system using a large number of randomly selected test cases:
 - does not mean that many errors in the system will be uncovered.
- Consider an example for finding the maximum of two integers x and y .

Design of Test Cases



- The code has a simple programming error:
- If $(x > y)$ $\max = x$;
 else $\max = x$;
- test suite $\{(x=3, y=2); (x=2, y=3)\}$ can detect the error,
- a larger test suite $\{(x=3, y=2); (x=4, y=3); (x=5, y=1)\}$ does not detect the error.

Design of Test Cases



- Systematic approaches are required to design an optimal test suite:
 - each test case in the suite should detect different errors.

Design of Test Cases



- There are essentially two main approaches to design test cases:
 - Black-box approach
 - White-box (or glass-box) approach

Black-box Testing

- Test cases are designed using only **functional specification** of the software:
 - without any knowledge of the internal structure of the software.
- For this reason, black-box testing is also known as **functional testing**.

White-box Testing

□ Designing white-box test cases:

- requires knowledge about the internal structure of software.

- white-box testing is also called structural testing.

- In this unit we will not study white-box testing.

Black-box Testing



- There are essentially two main approaches to design black box test cases:
 - Equivalence class partitioning
 - Boundary value analysis

Equivalence Class Partitioning



- Input values to a program are partitioned into **equivalence classes**.
- Partitioning is done such that:
 - **program behaves in similar ways to every input value belonging to an equivalence class.**

Why define equivalence classes?



- Test the code with just one representative value from each equivalence class:
 - as good as testing using any other values from the equivalence classes.

Equivalence Class Partitioning



- How do you determine the equivalence classes?
 - examine the input data.
 - few general guidelines for determining the equivalence classes can be given

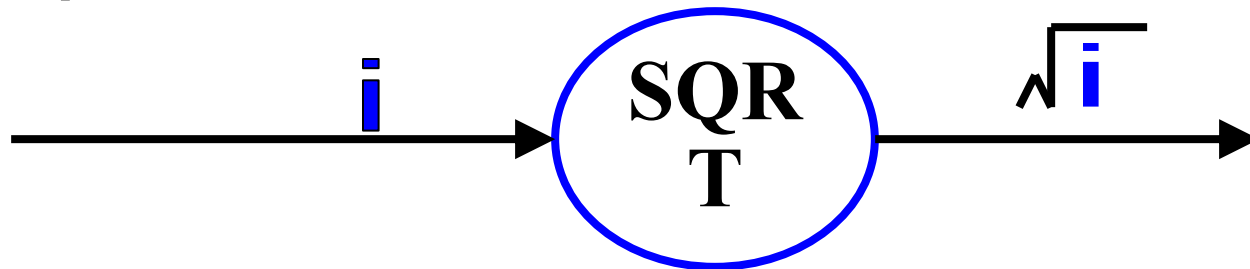
Equivalence Class Partitioning

- If the input data to the program is specified by a **range of values**:
 - e.g. numbers between 1 to 5000.
 - one valid and two invalid equivalence classes are defined.



Example

- A program reads an input value in the range of 1 and 5000:
- computes the square root of the input number



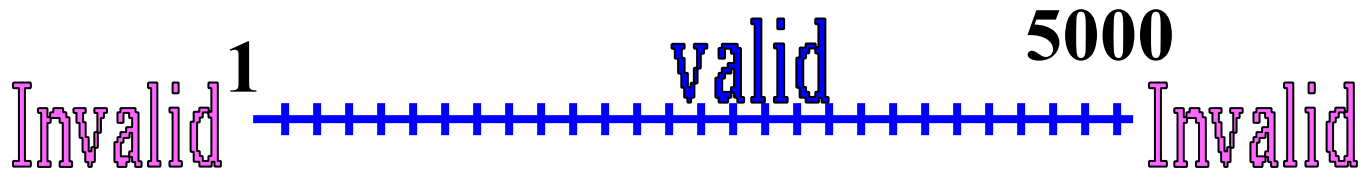
Example (cont.)

- There are three equivalence classes:
 - the set of negative integers,
 - set of integers in the range of 1 and 5000,
 - integers larger than 5000.



Example (cont.)

- The test suite must include:
 - representatives from each of the three equivalence classes:
 - a possible test suite can be: $\{-5, 500, 6000\}$.



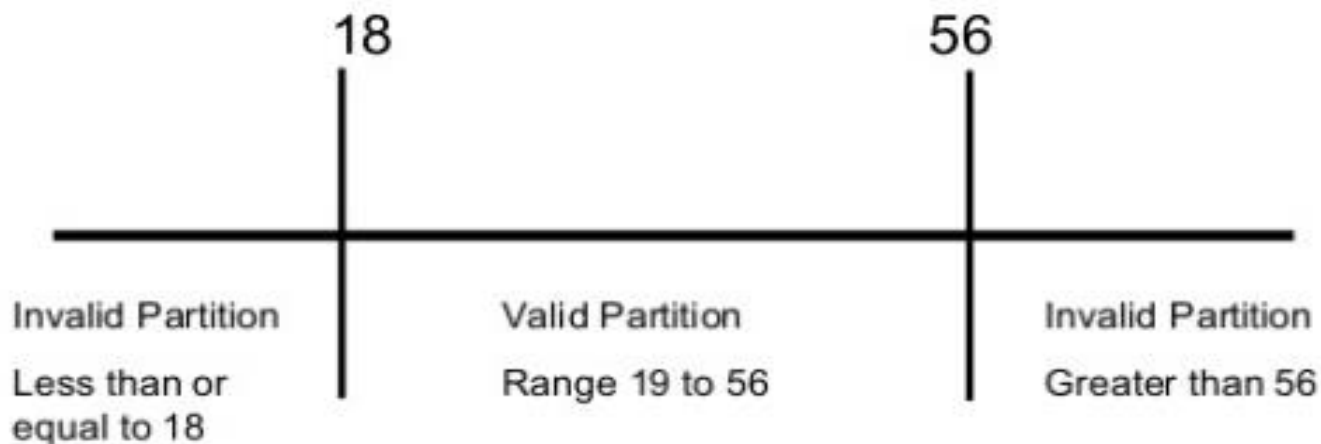
Equivalence Partitioning #3

- EP Example:
- Consider a requirement for a software system:
 - “The customer is eligible for a life assurance discount if they are at least 18 and no older than 56 years of age.”

For the exercise only consider integer years.

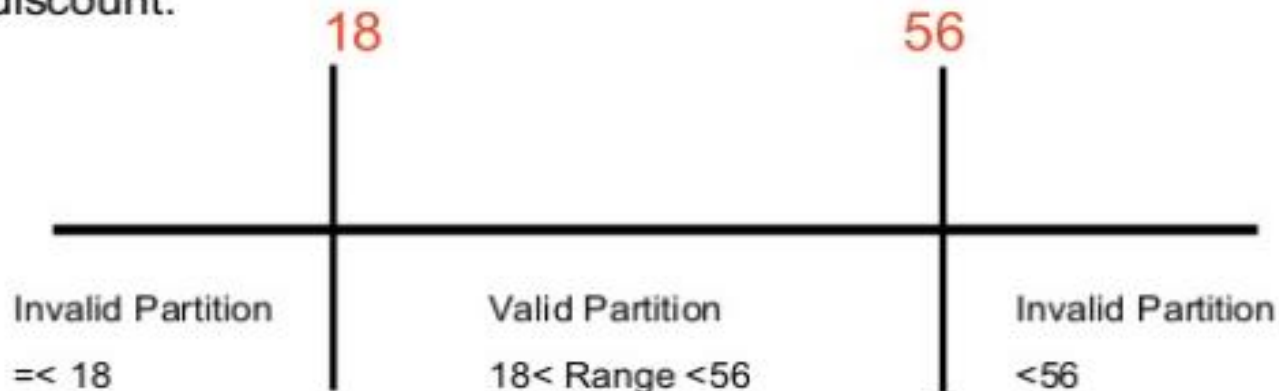
Equivalence Partitioning #4

- “The customer is eligible for a life assurance discount if they are at least 18 and no older than 56 years of age.”



Equivalence Partitioning #5

- What if our developer incorrectly interpreted the requirement as:
- "The customer is eligible for a life assurance discount if they are **over** 18 and **less** than 56 years of age."
- People aged exactly 18 or exactly 56 would now not get a discount.



Errors are more common at boundary values, either just below, just above or specifically on the boundary value.

Boundary Value Analysis



- Some typical programming errors occur:
 - at boundaries of equivalence classes
 - might be purely due to psychological factors.
- Programmers often fail to see:
 - special processing required at the boundaries of equivalence classes.

Boundary Value Analysis

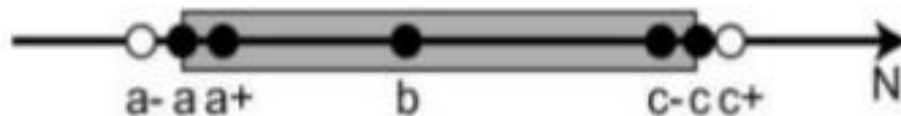


- Programmers may improperly use $<$ instead of \leq
- Boundary value analysis:
 - select test cases at the boundaries of different equivalence classes.

Boundary value analyze

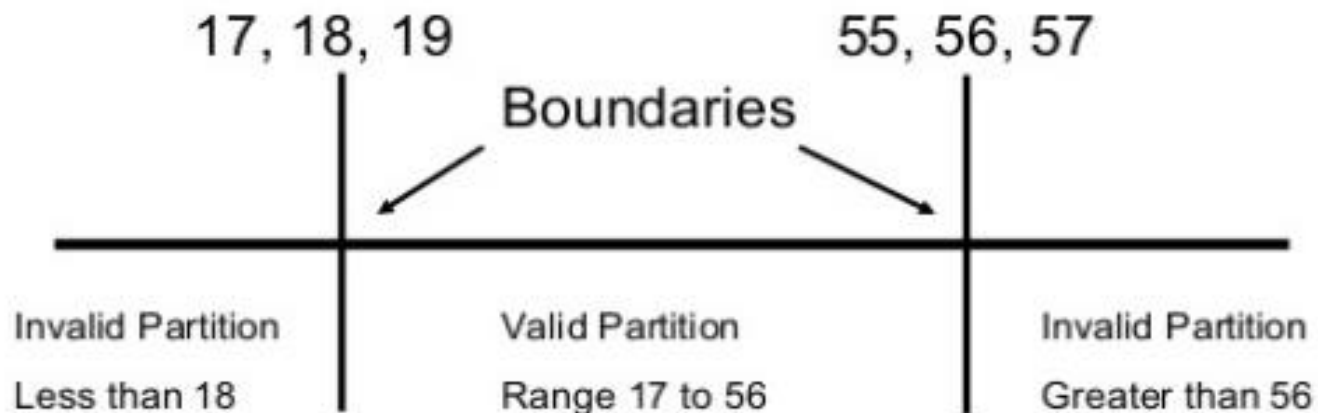
Test case design by BVA proceeds into 3 steps:

- Determine the range of values (usually it is equivalence class)
- Determine boundary values
- Check input variable value at the minimum, just above minimum, just below minimum, normal, at the maximum, just below maximum, just above maximum



Boundary Analysis #1

- "The customer is eligible for a life assurance discount if they are at least 18 and no older than 56 years of age."



Test values would be: 17, 18, 19, 55, 56 and 57.

This assumes that we are dealing with integers and so least significant digit is 1 either side of boundary.

Example

- For a function that computes the square root of an integer in the range of 1 and 5000:
- test cases must include the values: {0,1,5000,5001}.



Question 1

- One of the fields on a form contains a text box which accepts numeric values in the range of 18 to 26. Identify the invalid Equivalence class.
 - a) 17
 - b) 19
 - c) 25
 - d) 21

Question 2


- In an Examination a candidate has to score minimum of 25 marks in order to pass the exam. The maximum that he can score is 50 marks. Identify the Valid Equivalence values if the student passes the exam.
 - a) 22,24,27
 - b) 21,39,40
 - c) 29,30,31
 - d) 0,15,22

Performance Testing



- Addresses non-functional requirements.
 - May sometimes involve testing hardware and software together.
 - There are several categories of performance testing.

Stress testing



- Evaluates system performance
 - when stressed for short periods of time.
- Stress testing
 - also known as **endurance testing**.

Stress testing



- Stress tests are black box tests:
 - designed to impose a range of abnormal and even illegal input conditions
 - so as to stress the capabilities of the software.

Stress Testing



- If the requirements is to handle a specified number of users, or devices:
 - stress testing evaluates system performance when all users or devices are busy simultaneously.

Stress Testing



- If an operating system is supposed to support 15 multiprogrammed jobs,
 - the system is stressed by attempting to run 15 or more jobs simultaneously.
- A real-time system might be tested
 - to determine the effect of simultaneous arrival of several high-priority interrupts.

Volume Testing



- Addresses handling large amounts of data in the system:
 - whether data structures (e.g. queues, stacks, arrays, etc.) are large enough to handle all possible situations
 - Fields, records, and files are stressed to check if their size can accommodate all possible data volumes.

Configuration Testing

- Analyze system behavior:
 - in various hardware and software configurations specified in the requirements
 - sometimes systems are built in various configurations for different users
 - for instance, a minimal system may serve a single user,
 - other configurations for additional users.

Recovery Testing



- These tests check response to:
 - presence of faults or to the loss of data, power, devices, or services
 - subject system to loss of resources
 - check if the system recovers properly.

Maintenance Testing



- Verify that:
 - all required artifacts for maintenance exist
 - they function properly

Documentation tests



- Check that required documents exist and are consistent:
 - user guides,
 - maintenance guides,
 - technical documents

Documentation tests



- Sometimes requirements specify:
 - format and audience of specific documents
 - documents are evaluated for compliance

Usability testing



- Usability Testing also known as User Experience(UX) Testing, is a testing method for measuring how easy and user-friendly a software application is
- All aspects of user interfaces are tested:
 - Is the system is easy to learn?
 - Is the system useful and adds value to the target audience?
 - Are Content, Color, Icons, Images used are aesthetically pleasing?

Regression Testing



- Does not belong to either unit test, integration test, or system test.
- In stead, it is a separate dimension to these three forms of testing.

Regression testing



- Regression testing is the running of test suite:
 - after each change to the system or after each bug fix
 - ensures that no new bug has been introduced due to the change or the bug fix.

Regression testing



- Regression tests assure:
 - the new system's performance is at least as good as the old system
 - always used during phased system development.

Module Testing



- Module testing is defined as a software testing type, which checks individual subprograms, subroutines, classes, or procedures in a program. Instead of testing whole software program at once, module testing recommends testing the smaller building blocks of the program.

Module Testing



- Module testing is largely a white box oriented. The objective of doing Module, testing is not to demonstrate proper functioning of the module but to demonstrate the presence of an error in the module.
- Module level testing allows to implement parallelism into the testing process by giving the opportunity to test multiple modules simultaneously.

Incremental testing



- Incremental testing is one of the testing approach which is commonly used in software field during the testing phase of integration testing which is performed after unit testing. Several stubs and drivers are used to test the modules one after one which helps in discovering errors and defects in the specific modules.

Incremental testing



- after the completion of unit testing, integration testing is performed accordingly which is the simple process of detecting the interface and interaction between different modules.
- So, while the ongoing process of integration takes place there is a lot of methods and technology used one of that is incremental testing.
- It's a kind of approach where developers sums up the modules one after one to unfold the defects.

System Testing



- System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.
- The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system.

Security Testing



- Security Testing is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss.
- Security testing of any system is focuses on finding all possible loopholes and weaknesses of the system which might result into the loss of information or reputation of the organization.

Security Testing



The goal of security testing is to:

- To identify the threats in the system.
- To measure the potential vulnerabilities of the system.
- To help in detecting every possible security risks in the system.
- To help developers in fixing the security problems through coding.

Storage testing



- Storage testing is a type of software testing that is performed to verify whether the software stores the relevant data in the appropriate directories or not and there is sufficient space to prevent unexpected termination due to insufficient disk space, i.e., stack overflow.

Storage testing



Objective of Storage Testing:

- ❑ To determine practical storage limitation before deployment.
- ❑ To determine behavior of system when new hardware device is replaced or any existing device is upgraded.
- ❑ To minimize the response time.
- ❑ To fasten the processing.

Installation testing



- Testing the procedures to achieve an installed software system that can be used are known as installation testing. In this installation testing checking full or partial upgrades and other features install/uninstall processes are included.
- The installation testing ensures that the software application has been successfully installed with all its inherent features or not. It is also named as implementation testing, mainly it's done in the end phase.

What are the features of installation testing ?

- Activity based testing
- Executed during operational Acceptance testing
- Performed by software testing engineers along configuration manager
- Helps in deliver optimum user experience
- Helps in identification and detection of bugs during the installation

Procedure testing



- Procedure testing models the procedural requirements of the software system as a complete and delivered unit. Procedure Requirements define what is expected of any procedural documentation and shall be written in the form of Procedural instructions.
- These procedural instructions will normally come in the form of one of the following documents: A user Guide, An Instruction Manual, A User Reference Manual