Software Quality Attributes:

Software quality attributes are the characteristics that define the overall quality and performance of a software product. These attributes are used to evaluate and measure the effectiveness and efficiency of a software system. Several quality attributes are commonly used in software testing methodologies. Here are some of the most important ones:

1. Functionality: This refers to the ability of the software to perform the intended tasks and meet the user's needs.
2. Reliability: This refers to the ability of the software to function without failure over a period of time, under different conditions and usage patterns.
3. Usability: This refers to the ease of use of the software, and how easily the user can interact with it to accomplish their goals.
4. Performance: This refers to the speed and responsiveness of the software, including how quickly it can process data and perform tasks.
5. Security: This refers to the ability of the software to protect against unauthorized access, attacks, and data breaches.
6. Maintainability: This refers to the ease with which the software can be modified, updated, and repaired over time, without affecting its overall functionality or reliability.
7. Scalability: This refers to the ability of the software to handle increasing amounts of data or users, without suffering from degraded performance or reliability.

Overall, these software quality attributes are critical for ensuring that a software product is effective, efficient, reliable, and user-friendly. By evaluating these attributes during the testing process, developers can identify and fix any issues that may affect the quality and performance of the software, and ensure that it meets the needs of its intended users.

Bulding Blocks of Software testing

The building blocks of software testing are the key components that make up the testing process. These building blocks are used to plan, design, execute, and report on software tests. Here are some of the most important building blocks of software testing:

1. Test Plan: This is a comprehensive document that outlines the scope, objectives, and strategies for testing a software product. It includes details on the testing approach, resources required, test cases, and timelines.
2. Test Cases: These are specific scenarios and conditions that are designed to test the functionality, performance, and other quality attributes of the software product. They are often created based on user requirements, specifications, and use cases.
3. Test Suites: These are collections of related test cases that are grouped together for more efficient execution and reporting. Test suites may be designed based on functional areas, use cases, or other criteria.

4. Test Scripts: These are automated scripts that are used to execute test cases and generate test results. Test scripts can be created using various testing tools and frameworks, and are often used for regression testing.
5. Test Environment: This is the setup and configuration of the hardware, software, and other components required to execute the software tests. The test environment should replicate the production environment as closely as possible.
6. Test Data: This is the input and output data used in the test cases to simulate real-world scenarios and conditions. Test data may include sample data, boundary values, and edge cases.
7. Test Reporting: This involves documenting and communicating the results of the software tests to stakeholders. Test reports may include details on test coverage, pass/fail rates, defects found, and other relevant metrics.

By using these building blocks in a systematic and structured way, software testing teams can ensure that the software product meets the desired quality attributes and requirements, and is ready for release to end-users.

People, process, product, and technology (also known as the "Four P's"):

People, process, product, and technology (also known as the "Four P's") are the key components that make up a software development and testing ecosystem. These components are interdependent and must work together to ensure the successful delivery of high-quality software products.

1. People: This refers to the team of professionals involved in the software development and testing process, including developers, testers, project managers, and other stakeholders. People bring unique skills, perspectives, and experience to the project and are critical to its success.
2. Process: This refers to the set of procedures, methods, and guidelines that are followed to design, develop, test, and deliver the software product. The software development and testing process includes multiple stages, such as requirements gathering, design, coding, testing, and deployment. The process ensures that the software is developed and tested in a consistent and repeatable way.
3. Product: This refers to the software product itself, including its features, functionality, and quality attributes. The product must meet the needs of the end-users and stakeholders, as well as comply with industry standards and regulations.
4. Technology: This refers to the tools, platforms, and infrastructure used to develop and test the software product. Technology includes programming languages, development frameworks, testing tools, and other software components. The technology must be reliable, scalable, and secure to support the development and testing process.

By considering the interplay of people, process, product, and technology, software development and testing teams can ensure that the software product is delivered on time, within budget, and to

the desired quality standards. Effective collaboration and communication among the stakeholders is crucial to ensure the success of the project.

Types of Test Techniques:

There are various types of test techniques that are used in software testing methodologies. Each technique is designed to test a specific aspect of the software product and provides valuable insights into its functionality, performance, and other quality attributes. Here are some of the most common types of test techniques:

1. Black Box Testing: This is a testing technique where the tester examines the behavior of the software without knowledge of the internal workings of the system. It is based on testing the inputs and outputs of the software and analyzing how the system responds.
2. White Box Testing: This is a testing technique where the tester has knowledge of the internal workings of the software system. It involves examining the source code and internal data structures to test the system's logic and performance.
3. Grey Box Testing: This is a testing technique that combines both black box and white box testing. It involves testing the system from the perspective of an end-user while also having access to some knowledge of the system's internal workings.
4. Manual Testing: This is a testing technique that involves a human tester manually executing test cases and verifying the results. Manual testing can be time-consuming but provides valuable feedback on the usability and user experience of the software product.
5. Automated Testing: This is a testing technique that uses testing tools and frameworks to automate the execution of test cases and generate test reports. Automated testing is faster and more efficient than manual testing and is often used for regression testing.
6. Exploratory Testing: This is a testing technique that involves the tester exploring the software system without a specific test plan or script. It is based on the tester's intuition and experience and is often used to identify unexpected bugs or issues.
7. Acceptance Testing: This is a testing technique that involves verifying whether the software product meets the requirements and expectations of the end-users and stakeholders. It is typically the final stage of testing before the software product is released to production.

Overall, the choice of test techniques depends on the specific goals, requirements, and constraints of the software testing project. A combination of different test techniques can provide a comprehensive evaluation of the software product and help ensure that it meets the desired quality standards.

Testing Types:

1. Functional Testing: This is a type of testing that verifies whether the software product meets its intended functional requirements. It involves testing the software's behavior under various inputs and conditions.

2. Non-functional Testing: This is a type of testing that verifies whether the software product meets non-functional requirements, such as performance, security, usability, and compatibility.
3. Regression Testing: This is a type of testing that verifies whether the changes made to the software product during development have introduced any new defects or issues.
4. User Acceptance Testing (UAT): This is a type of testing that involves end-users or stakeholders testing the software product to ensure that it meets their requirements and expectations.
5. Integration Testing: This is a type of testing that verifies the interactions between different software modules or components.
6. System Testing: This is a type of testing that verifies whether the software product meets its intended requirements and works as a complete system.
7. Smoke Testing: This is a type of testing that verifies whether the basic functionality of the software product is working as expected.

Testing Levels:

1. Unit Testing: This is the testing of individual units or components of the software product.
2. Integration Testing: This is the testing of the interactions between different software modules or components.
3. System Testing: This is the testing of the software product as a complete system.
4. Acceptance Testing: This is the testing of the software product to ensure that it meets the end-users' requirements and expectations.

The testing levels progress from lower levels of testing (unit testing) to higher levels of testing (acceptance testing). Each level of testing verifies different aspects of the software product, and defects and issues identified at a lower level of testing are resolved before moving to the next level.

Overall, the choice of testing types and levels depends on the specific goals, requirements, and constraints of the software testing project. A combination of different testing types and levels can provide a comprehensive evaluation of the software product and help ensure that it meets the desired quality standards.

error vs fault vs failure vs incident vs test vs testcase

1. Error: A mistake made by a human that results in an incorrect or unexpected result in the software.
2. Fault: A defect or bug in the software that causes it to behave incorrectly or produce incorrect results. Faults are typically caused by errors in the design, coding, or testing of the software.
3. Failure: An event that occurs when the software does not perform as expected or produces incorrect results due to a fault.

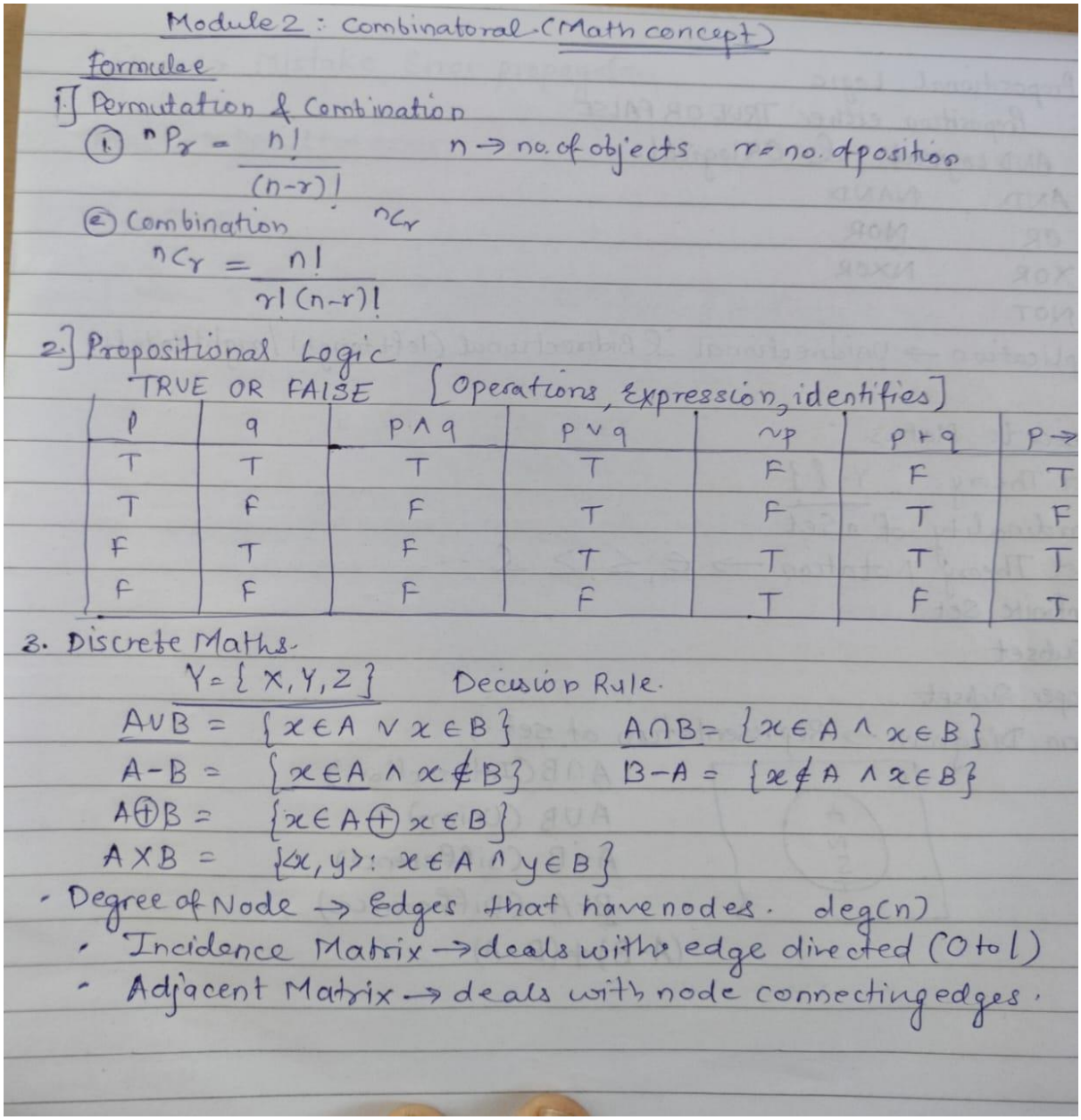
4. Incident: An event that occurs during testing or production that requires investigation, such as a failure, error, or unexpected behavior.
5. Test: The process of evaluating the software product to identify faults, errors, or other defects.
6. Test Case: A set of instructions or procedures that are used to evaluate a specific aspect of the software product. Test cases are designed to identify defects or other issues in the software.

Importance Of Software Testing

Software testing is a crucial process in the software development life cycle, as it helps to identify defects or bugs in the software before it is released to the end-users. The importance of software testing can be summarized as follows:

1. Improved quality: Software testing ensures that the software meets the required quality standards and is free from defects. This results in a better quality product that is more reliable and meets the needs of the end-users.
2. Reduced costs: Identifying and fixing defects early in the development process can save significant costs in terms of time and money. It is much cheaper to fix a defect during the development phase than after the product has been released to the market.
3. Increased customer satisfaction: Software testing helps to ensure that the software meets the needs of the end-users, and is reliable and user-friendly. This results in increased customer satisfaction, which is essential for the success of any software product.
4. Compliance with regulations: Many software products are subject to regulations and standards, such as safety standards, data privacy regulations, and accessibility standards. Software testing helps to ensure that the software complies with these regulations and standards.
5. Increased productivity: By identifying and fixing defects early in the development process, software testing helps to ensure that developers can focus on developing new features and functionality, rather than fixing defects.

In summary, software testing is essential to ensure that software products are of high quality, meet customer needs, comply with regulations, and are reliable and user-friendly.



Module 2 : Combinatoral (Math concept)

Formulae

1.] Permutation & Combination

(1) $^nP_r = \frac{n!}{(n-r)!}$ $n \rightarrow$ no. of objects $r =$ no. of position

(2) Combination ^nC_r

$^nC_r = \frac{n!}{r!(n-r)!}$

2.] Propositional Logic

TRUE OR FAISE [Operations, Expression, identifies]

p	q	p∧q	p∨q	~p	p→q	p→
T	T	T	T	F	F	T
T	F	F	T	F	T	F
F	T	F	T	T	T	T
F	F	F	F	T	F	T

3. Discrete Maths.

$Y = \{X, Y, Z\}$ Decision Rule.

$A \cup B = \{x \in A \vee x \in B\}$ $A \cap B = \{x \in A \wedge x \in B\}$

$A - B = \{x \in A \wedge x \notin B\}$ $B - A = \{x \notin A \wedge x \in B\}$

$A \oplus B = \{x \in A \oplus x \in B\}$

$A \times B = \{\langle x, y \rangle : x \in A \wedge y \in B\}$

- Degree of Node → Edges that have nodes. deg(n)
- Incidence Matrix → deals with edge directed (0 to 1)
- Adjacent Matrix → deals with node connecting edges.

Equivalance Classes:

Equivalence class testing is a software testing technique that involves dividing the input data into groups, or "equivalence classes," and then selecting a representative value from each class to test. The goal of this technique is to reduce the number of test cases required to adequately test the software, while still ensuring that all possible inputs are covered.

In equivalence class testing, the input data is classified into groups based on the assumption that if one value in the group is valid or invalid, then all other values in the same group are also valid or invalid. For example, if the input data is a positive integer, then the equivalence classes could be:

- Valid positive integers (e.g., 1, 2, 3)
- Invalid negative integers (e.g., -1, -2, -3)

- Invalid non-integer values (e.g., 1.5, 2.5, 3.5)

By selecting a representative value from each equivalence class, we can ensure that we test both valid and invalid inputs without having to test every possible value individually.

Equivalence class testing is particularly useful when there are a large number of possible input values, as it can reduce the number of test cases required while still providing good test coverage. However, it should be used in conjunction with other testing techniques to ensure that all possible inputs are adequately covered.

Boundary Value Analysis:

Boundary Value Analysis (BVA) is a testing technique that is commonly used in software testing methodology. BVA is a black-box testing technique that focuses on testing the boundary conditions of input parameters. The goal of BVA is to identify errors or defects that occur at or near the boundaries of the input domain.

BVA is based on the idea that errors are more likely to occur at the boundaries of input parameters. Therefore, the testing process should focus on testing the values that are at the edge of the input domain, rather than focusing on values that are in the middle of the domain.

BVA is particularly useful in situations where input values are limited by a range of values, such as input fields that accept only numeric values. In such cases, BVA can help identify errors that occur when values fall outside the acceptable range.

BVA can also be used in combination with other testing techniques, such as Equivalence Partitioning (EP). EP is a technique that divides the input domain into a set of equivalence classes, where each class represents a set of input values that produce the same output behavior. BVA can be used to test the boundary conditions of each equivalence class.

Overall, BVA is a valuable testing technique that helps identify errors that occur at or near the boundaries of input parameters. It is particularly useful in situations where input values are limited by a range of values, and it can be used in combination with other testing techniques to improve the overall effectiveness of the testing process.

Edge Testing:

Edge testing, also known as boundary testing, is a software testing technique that focuses on testing the extreme or boundary values of input data. The goal of edge testing is to identify and expose defects that may occur at the boundaries of the input domain.

In edge testing, testers identify the maximum and minimum values of input parameters and test the behavior of the software at these values. This is important because many errors can occur at the edges of the input domain, such as input validation errors, buffer overflows, and arithmetic errors.

For example, if a software application has an input field that accepts only numeric values between 1 and 100, edge testing would involve testing the behavior of the application when the input is exactly 1 or 100. Testers would check to see if the application accepts the input, validates it correctly, and produces the expected output.

Edge testing can be used in combination with other testing techniques, such as equivalence partitioning and decision table testing, to increase the effectiveness of the testing process. By focusing on the extreme values of input data, testers can identify and eliminate defects that may not be found using other testing techniques.

In summary, edge testing is a valuable testing technique that helps identify and eliminate defects that occur at the boundaries of the input domain. It is an important part of a comprehensive software testing strategy and can be used in combination with other testing techniques to ensure high-quality software.

Domain Tetsing:

Domain testing is a software testing technique that involves selecting input values from various domains or ranges of values and testing the system's response to those inputs. It aims to ensure that the system is capable of handling inputs from different domains and produces the expected output.

In domain testing, the input values are selected from a set of valid and invalid values that fall within the boundaries or ranges of the input domain. The input domain is defined as the set of all possible input values that the system can accept.

The input domain can be divided into various subdomains, such as:

- Valid inputs: inputs that the system should accept and produce the expected output.
- Invalid inputs: inputs that the system should reject and produce an error or warning message.
- Extreme inputs: inputs that are at the boundaries of the input domain, such as the minimum and maximum values or values close to them.
- Erroneous inputs: inputs that are unlikely to be entered by a user, such as a string of letters in a numeric input field.

Domain testing can help identify defects that may occur when the system is presented with inputs from different domains. It ensures that the system is robust enough to handle unexpected or erroneous inputs, and that it produces the correct output for valid inputs.

By selecting test cases from each subdomain, domain testing can provide a comprehensive and effective approach to testing the system's input handling capabilities. However, it is important to combine domain testing with other testing techniques to ensure that all possible scenarios are covered.

Category Partitioning Method

Category Partitioning Method is a software testing technique used to identify test cases based on various categories or partitions of input values. It is a variation of Equivalence Class Testing technique, which divides the input values into a set of equivalence classes.

In Category Partitioning Method, the input values are divided into categories, where each category represents a group of input values that are expected to behave similarly. The categories can be based on different characteristics of the input values, such as their data type, range, length, format, etc.

The testing process involves selecting at least one test case from each category to ensure that all possible combinations of input values are covered. This technique is useful in reducing the number of test cases required to achieve maximum test coverage and provides a systematic approach to identify test cases that exercise different combinations of inputs.

For example, suppose we have an input field that accepts a username, which can be a combination of alphabets, digits, and special characters. In that case, we can divide the input values into the following categories:

- Valid usernames: usernames that meet all the requirements for a valid username.
- Invalid usernames: usernames that fail to meet one or more requirements, such as usernames with spaces or usernames that are too long.
- Edge cases: usernames that are at the boundary of the acceptable range, such as the shortest and longest possible usernames.
- Invalid data types: inputs that are not strings, such as integers or floating-point numbers.

By selecting at least one test case from each category, we can ensure that all possible combinations of input values are covered, and the system is tested thoroughly. Category Partitioning Method is an effective technique for designing test cases for complex systems with multiple inputs and input categories.

Combinatorial

Combinatorial Testing is a software testing technique that aims to identify faults that may occur due to interactions between input parameters or factors in a system. It involves generating a set of test cases that cover all possible combinations of input parameter values.

Combinatorial testing is based on the mathematical concept of combinatorics, which deals with the study of combinations and permutations. The goal of combinatorial testing is to identify the minimum number of test cases that cover all possible combinations of input parameters.

For example, suppose we have a web application that allows users to search for products based on various criteria, such as product category, price range, and brand. In that case, we can use combinatorial testing to identify all possible combinations of input parameter values, such as:

- Category: electronics, clothing, accessories, books
- Price range: $0-$25, $25-$50, $50-$100, $100 and above

- Brand: Samsung, Apple, Sony, HP, Dell, Lenovo

Using combinatorial testing, we can generate a set of test cases that cover all possible combinations of input parameter values, such as:

- Electronics, $0-$25, Samsung
- Clothing, $25-$50, Apple
- Accessories, $50-$100, Sony
- Books, $100 and above, HP
- Electronics, $25-$50, Dell
- Clothing, $50-$100, Lenovo
- Accessories, $0-$25, Samsung
- Books, $50-$100, Apple

By using combinatorial testing, we can ensure that all possible interactions between input parameters are covered, and the system is tested thoroughly. It is an effective technique for identifying faults that may occur due to interactions between input parameters and can help reduce the number of test cases required to achieve maximum test coverage.

Decision Table Based Testing:

Decision table-based testing is a black-box testing technique used to test software systems and applications. It involves creating a decision table that lists all possible combinations of input conditions and their corresponding output actions. Each row in the decision table represents a specific scenario or combination of input conditions, and each column represents a specific input condition or output action.

The decision table is created based on the system's requirements, and it serves as a test case specification for the software testers. The purpose of this technique is to ensure that all possible combinations of input conditions and their corresponding output actions are tested, thus achieving maximum test coverage.

To perform decision table-based testing, the following steps are typically followed:

1. Identify the input conditions and output actions of the system under test.
2. Create a decision table that lists all possible combinations of input conditions and their corresponding output actions.
3. Identify the minimum number of test cases required to cover all the combinations in the decision table.
4. Execute the identified test cases and record the results.
5. Verify that the actual results match the expected results for each test case.

Overall, decision table-based testing is an effective technique for identifying defects and ensuring that all possible combinations of input conditions and their corresponding output actions are tested. It can also help to improve the quality of software systems and applications by reducing the risk of defects in production.

Design Test Cases for a Software Program that takes in an input of up to 1000 numbers, finds the maximum and output is the max number with *equivalence classes*

Equivalence class testing is a technique that involves identifying different groups or classes of inputs that are expected to behave similarly. To design test cases for a software program that takes in an input of up to 1000 numbers, finds the maximum, and outputs the max number using equivalence class testing, we can follow these steps:

1. Identify the different types of inputs that the program can receive and divide them into equivalence classes. For example, we could divide the inputs into the following equivalence classes:
    - Positive integers
    - Negative integers
    - Mixed positive and negative integers
    - Decimal numbers
    - Empty set
    - Single number set
    - Set with 1000 numbers
    - Set with more than 1000 numbers
    - Non-numeric values
2. For each equivalence class, design test cases that cover both valid and invalid inputs. Some examples of test cases are:
    - Equivalence class: Positive integers
        - Test case 1: Input a set of positive integers including zero. The output should be the maximum of the set.
        - Test case 2: Input a set of positive integers greater than zero. The output should be the maximum of the set.
    - Equivalence class: Negative integers
        - Test case 3: Input a set of negative integers. The output should be the maximum of the set.

        - Test case 4: Input a set of negative integers including zero. The output should be zero.
    - Equivalence class: Mixed positive and negative integers
        - Test case 5: Input a set of mixed positive and negative integers. The output should be the maximum of the set.
        - Test case 6: Input a set of mixed positive and negative integers including zero. The output should be zero.
    - Equivalence class: Decimal numbers
        - Test case 7: Input a set of decimal numbers. The output should be the maximum of the set.
        - Test case 8: Input a set of decimal numbers including zero. The output should be zero.
    - Equivalence class: Empty set
        - Test case 9: Input an empty set. The output should be an error message indicating that the set is empty.
    - Equivalence class: Single number set
        - Test case 10: Input a set with only one number. The output should be that number.
    - Equivalence class: Set with 1000 numbers
        - Test case 11: Input a set with 1000 numbers. The output should be the maximum of the set.
    - Equivalence class: Set with more than 1000 numbers
        - Test case 12: Input a set with more than 1000 numbers. The program should handle the input gracefully and return the maximum of the first 1000 numbers.
    - Equivalence class: Non-numeric values
        - Test case 13: Input a set with non-numeric values. The program should handle the input gracefully and return an error message.
3. Execute the test cases and compare the actual output with the expected output to ensure that the program functions correctly in all cases.