

Warm up

Software Testing

Software Testing and Quality Assurance

      @iungo_solutions

<https://iungo.solutions/>

Learning Outcome

A decorative graphic consisting of a purple rectangular frame. To the top-left of the frame are three circles: a small magenta one, a medium cyan one, and a large blue one. To the bottom-right of the frame are four circles: a large magenta one, a medium purple one, a small cyan one, and a medium blue one. The text "Undersand principles of software testing" is centered within the purple frame.

Undersand principles of software testing

Define testing and software testing

      @iungo_solutions

<https://iungo.solutions/>

Testing

What is testing?

Testing is the process of evaluating something to ensure it functions as expected, meets requirements, or satisfies a set of criteria.

Testing can be applied in various contexts, such as verifying the quality, performance, or correctness of an item or system.

How do you test something in real life?

You go and check an object or process to confirm it behaves as intended.

Define testing

Software Testing

Why do we test software?

- **Identify bugs:** detect and fix errors before release.
- **Verify requirements:** ensure the software meets user and business requirements.
- **Improve quality:** deliver robust, secure, and reliable software.
- **Save cost:** fixing bugs early is less expensive than after deployment.

**Define software
testing**

Software Testing

What is software testing?

In software, testing refers to the process of evaluating a program, application, or system to ensure it behaves as expected under various conditions.

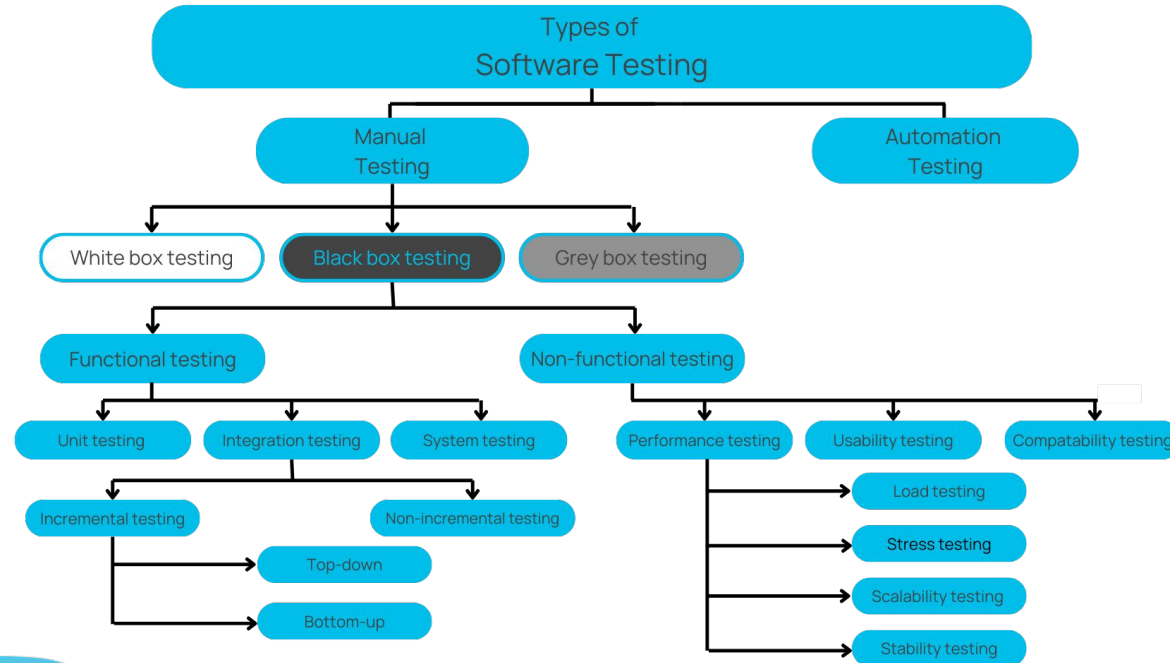
**Define software
testing**

Explain different types of software tests

      @iungo_solutions

<https://iungo.solutions/>

Types of Software Testing

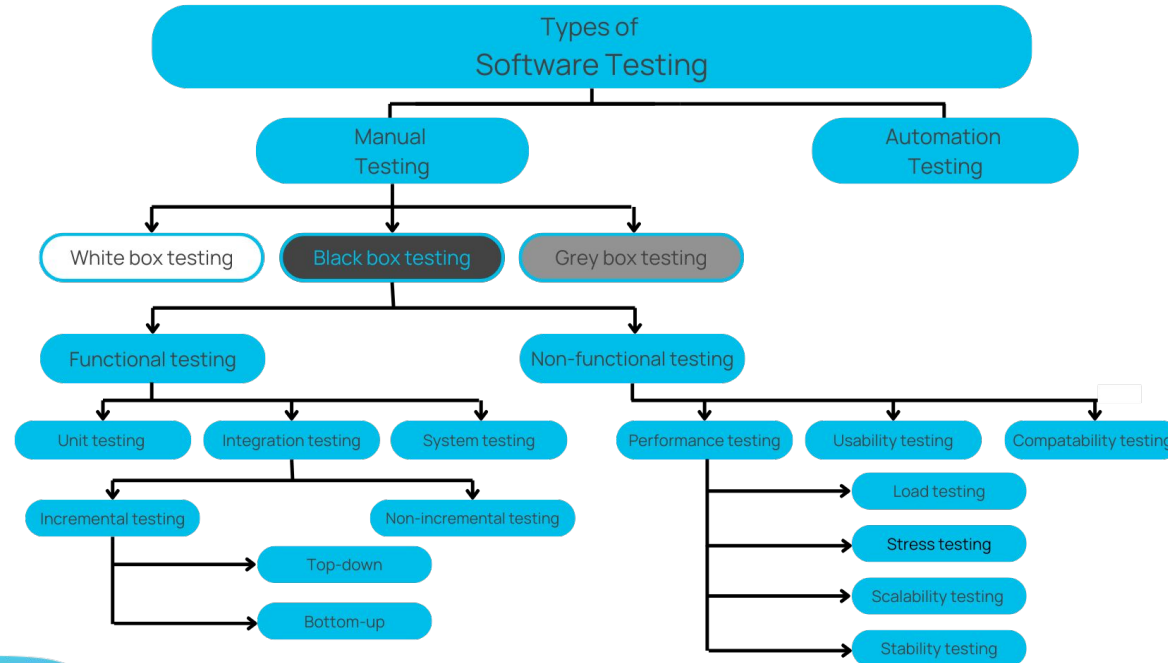


Manual vs Automated Testing

Manual testing is completed by a person by clicking through the app or interacting with the software. It can be prone to human error, but provides immediate visual feedback and can be adopted easily.

Automated testing is carried out by a machine which has a pre written test script. It is much more reliable and interacts automatically, but the quality of the tests depends on the complexity of the test script.

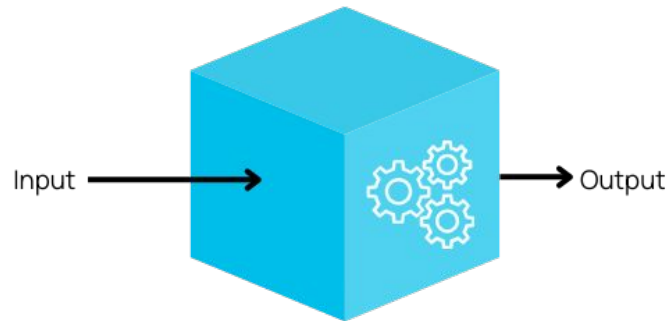
Types of Software Testing



White-Box Testing

Tests internal coding and infrastructure, it checks predefined inputs against expected and desired outputs.

- Looks at the inner workings and structure of the application.
- Main goal is to work on the flow of input to output and strength of the software.

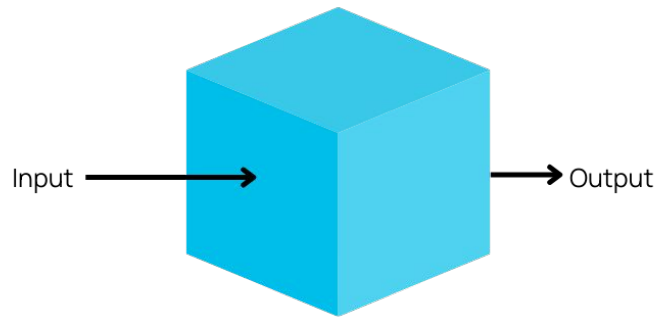


Manual testing

Black-box Testing

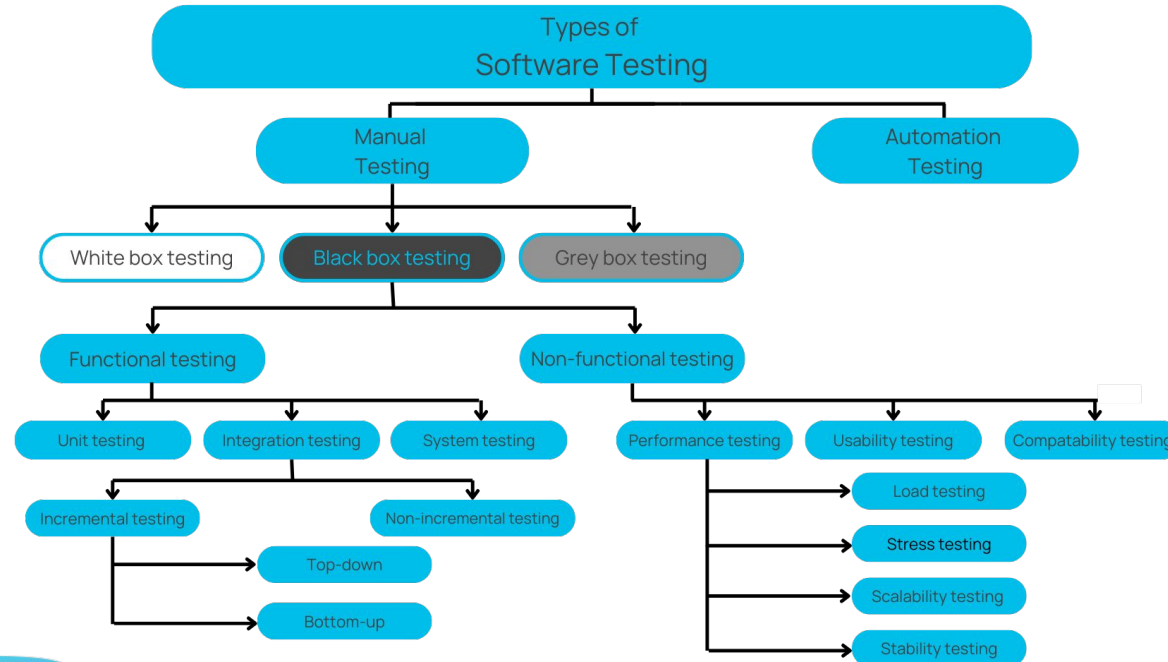
Testing the functionality of the software in line with the requirements and specifications provided. This type of testing is concerned only with functionality, not internal structures.

- The tests are executed from the client/end user's point of view, so no knowledge of the inner working of the software is needed.
- Each test is easily reproducible.



Manual testing

Types of Software Testing



Functional vs Non-Functional

Functional testing: Verifies the application works as intended.

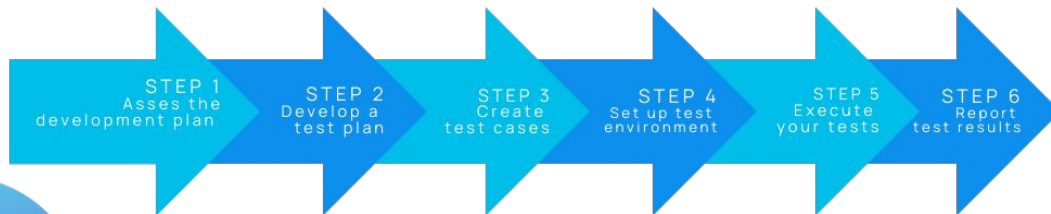
Non-functional testing: Evaluates performance, scalability, and security.

Black box testing

Unit Testing

Isolates certain parts of the code to ensure each component works correctly.

- Helps to catch bugs or issues early before the issues can multiply.
- Skipping unit testing can lead to larger issues further down the testing process.



**Functional
testing**

Integration Testing

Integration testing refers to testing more than one unit of a programme or section of code simultaneously, to see how well they work together.

There are two types of integration testing:

- **Incremental Integration Testing** starts by testing two modules of the programme together, if these are working then another module will be added. This process continues until all the required modules have been tested.
- **Non-incremental Integration Testing** combines all modules and tests them at the same time without breaking them down into smaller groups or components.

Regression Testing

Regression testing checks whether features work correctly after they have been changed or updated. It is the process of repeating tests that have already been run, to check if the expected results are achieved again following an update.

It is called regression testing because if a programme stops functioning correctly following an update, this is referred to as a regression.

Acceptance Testing

This is the final stage of testing before a product is released, and checks that it is suitable for the end users or client it is designed for.

Acceptance testing falls under the category of a **Black-Box Test**, meaning that it is concerned only with how the programme functions for a user.

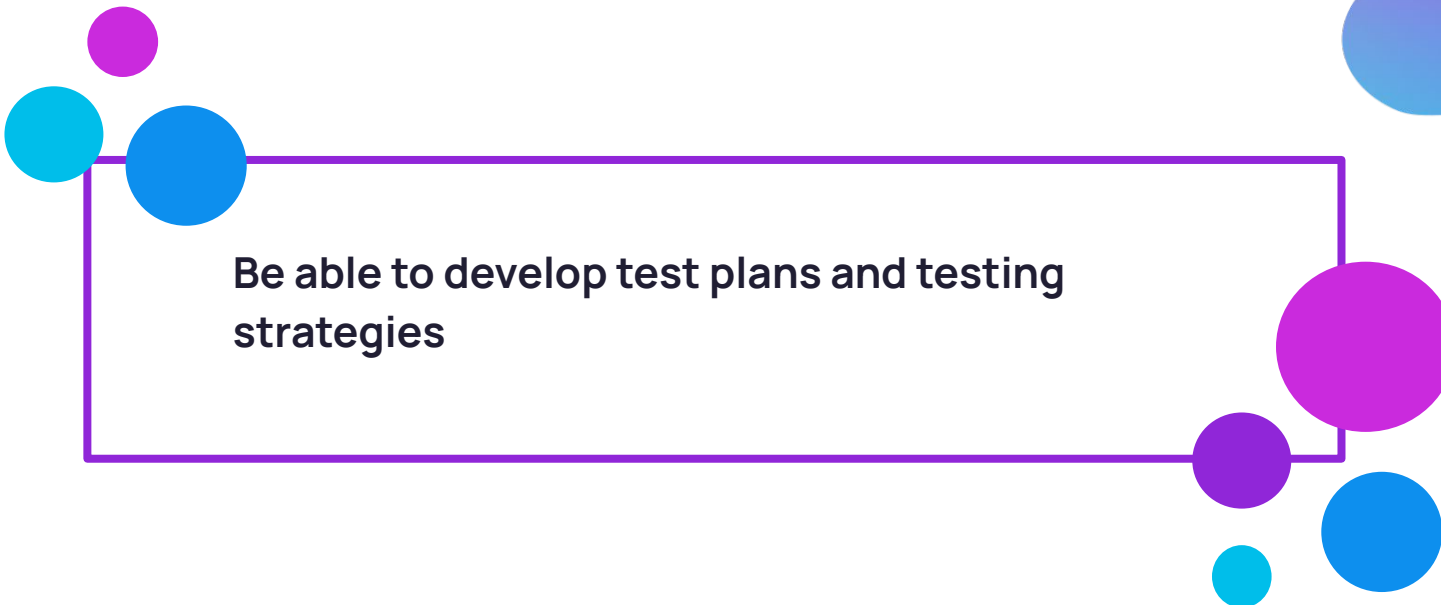
It is carried out by the intended users of the software, and checks the functionality and suitability of the programme when carrying out real-world tasks such as user registration and login, completing transactions without errors, and aspects such as how easy to navigate it is and whether it complies with any relevant regulations.

Load and Stress Testing

Load testing is testing the stability and response time of an application by applying load which would be equal or less than the designed number of users for the software.

Stress testing is testing the stability and response time of an application by applying load which would be more than the designed number of users for the software.

Learning Outcome

A decorative graphic consisting of a purple rectangular frame. To the top-left of the frame are three circles: a small magenta one, a medium cyan one, and a large blue one. To the bottom-right of the frame are four circles: a large magenta one, a medium purple one, a small cyan one, and a medium blue one. The background of the slide features abstract wavy shapes in shades of purple, blue, and cyan.

Be able to develop test plans and testing strategies

Define a test plan and outline their benefits

      @iungo_solutions

<https://iungo.solutions/>

Test plan

A test plan in software development:

- Is a document that outlines the testing project.
- Serves as a roadmap for the testing process.
- Defines what will be tested and the criteria for success. This includes specific features, functionalities, or modules that are within the testing boundaries.
- Ensures alignment with the project goals.

Benefits of a test plan

Some of the benefits of a test plan include:

- Early risk identification, preventing potential risks from becoming major problems later.
- Ensuring comprehensive testing. It helps in defining what will be in and out of scope, and ensuring that all critical functionalities and features are thoroughly tested.

Explain the key components of a test plan

      @iungo_solutions

<https://iungo.solutions/>

Key Components of a Test Plan

- **Objective:** Describes the main goals and purpose of the testing. It answers why the testing is necessary for the project.
- **Scope:** Defines what will and will not be covered in the testing process - what specific features, functionalities, or modules are within the testing boundaries.
- **Testing Approach and Strategy:** Specifies the type of testing to be conducted.
- **Exit Criteria:** Defines when the tests can be considered complete.
- **Resources and Responsibilities:** Lists the people, tools, and resources required for testing, along with their assigned roles.

Create a test plan for a software project

      @iungo_solutions

<https://iungo.solutions/>

Create a Test Plan for a Project

Step 1: Define the test plan objectives

- Clearly state the **objectives** of the testing process. What do you aim to achieve with the testing? This could include ensuring functionality, performance, security, or user acceptance.
- Define what will and will not be tested. Specify the features and functionalities of the application that are in **scope** and out of scope.

Create a Test Plan for a Project

Step 2: Define the Testing Strategy

- Specify the types of testing to be performed, such as:
 - Unit testing
 - Integration testing
 - System testing
 - Acceptance testing
 - Performance testing
- Identify the levels at which testing will occur, such as component, integration, and system levels.

Create a Test Plan for a Project

Step 3: Define Exit Criteria

- Specify the conditions under which testing will be considered complete. This may include:
 - A certain percentage of test cases passing.
 - Critical defects being resolved.
 - Completion of all planned testing activities.

Create a Test Plan for a Project

Step 4: Determine Test Resources

- Outline the roles and skills required for the testing team. Identify who will be responsible for writing, executing, and reviewing tests.
- Specify the testing tools, frameworks, and environments that will be used for testing.

Create a Test Plan for a Project

Step 5: Create Test Cases

- Write detailed **test cases** based on the requirements and specifications of the application. Each test case should include:
 - Description of the test
 - Preconditions
 - Test steps
 - Expected results
- Create **test data**
- Organize test cases into **test suites** as necessary.

Create a Test Plan for a Project

Step 6: Identify Risks and Create a Schedule

- Identify potential risks that could impact the testing process (e.g., resource unavailability, delays in development, unstable test environment) and plan how to mitigate these risks.
- Provide a timeline for all testing activities, including key milestones and deadlines.

Learning Outcome

A decorative graphic consisting of a purple rectangular frame. To the top-left of the frame are three circles: a small magenta one, a medium cyan one, and a large blue one. To the bottom-right of the frame are four circles: a large magenta one, a medium purple one, a small cyan one, and a medium blue one. The background features abstract wavy shapes in shades of purple, blue, and cyan.

Enact a test plan for a software project

Perform manual tests in a programming language



      @iungo_solutions

<https://iungo.solutions/>

Manual test

- **Understand Requirements:** Review the specifications or requirements to understand what needs to be tested.
- **Define Test Cases:** Identify the scenarios to test, including inputs, expected outputs, and the conditions to validate.
- **Define Test Data:** Identify and prepare the data needed for testing.

Manual test

- **Run the Tests:** Execute the test cases using the defined test data.
- **Compare Outputs:** Verify actual outputs against the expected results defined in the test cases.
- **Fix and Retest:** Work with the development team to address reported issues, then retest to confirm fixes.


Test data

Test data can be manually created by the test team, or re-use data pre-loaded in a database from a previous iteration. It should include:

- Examples of both valid, expected input from users, and unexpected or invalid input (for example invalid email addresses or required input fields left blank). This ensures that you can check that the software handles expected input as it should, and also that it handles errors appropriately.
- Edge cases, like minimum and maximum values (e.g. setting the age of a user to 0).
- Real-world data samples that simulate a user in the system.

Test data

When recording the results of your testing, it is important to be specific about what is being tested, the data used to test, and how any issues were resolved, and, most importantly, the comparison of expected results against actual results

Function being tested	Test Data	Expected Outcome	Actual Outcome	Resolution
AddNumbers	7, 5	12	2	Corrected operator from -= to +=
AddNumbers	7,5	12	12	Test Passed 

Recording results

Perform automated tests in a programming language

      @iungo_solutions

<https://iungo.solutions/>

Automated tests

Automated tests are written in code, and can be rerun anytime we need.

For example we can write the tests **before** the actual code: this is called **Test Driven Development**. Then focus on writing the code until all the tests pass.

The tests are normally run one last time right before deploying: this is called **Continuous Integration**.

Automated tests

To write your tests, start by creating a **test suite** to describe the feature that's being tested.

In the test suite you can add one or more tests.

Give human readable names to your tests, like:

should_throw_exception_when_email_is_null()

Inside your tests, you'll execute the code to be tested, then make some **assertions** about it, like comparing a returned value with the expected value. If the assertion is true, the test **passes**, and it **fails** otherwise.

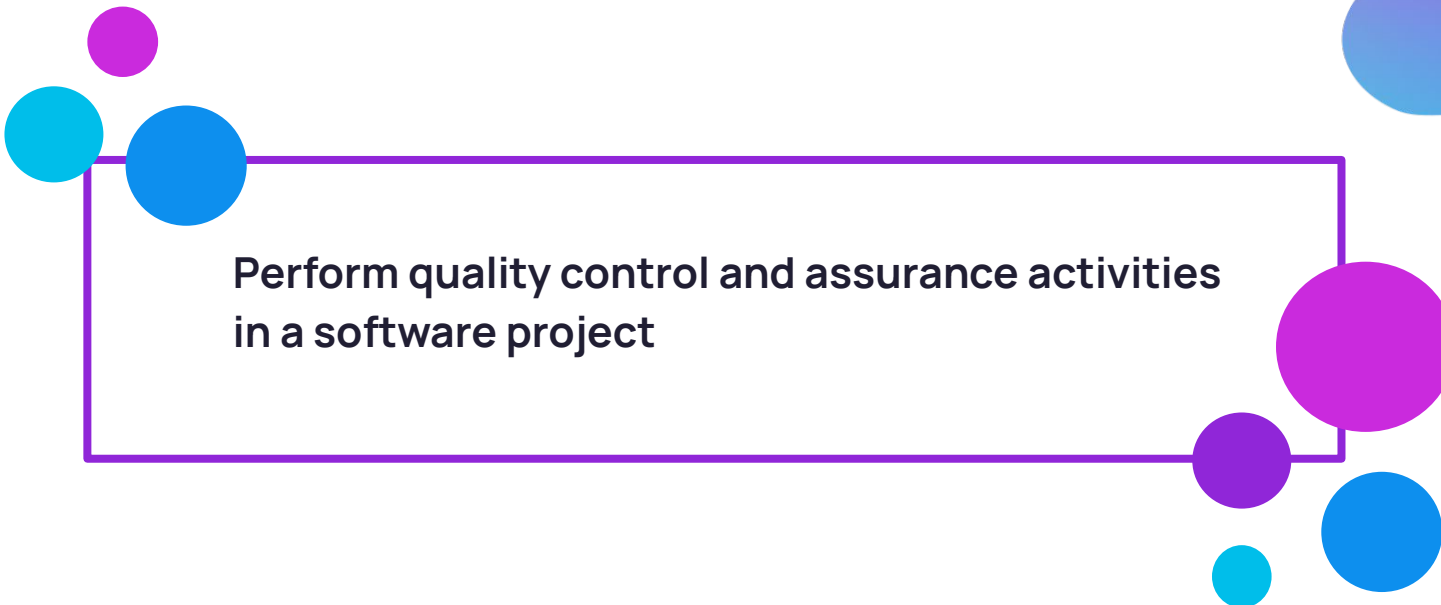
Automated tests

```
function void divide_two_positive_integers() {  
    int expected = 4  
    int actual = division(12, 3)  
    assert(actual == expected)  
}  
  
function void divide_by_0_exception() {  
    try:  
        division(12, 0)  
    catch division_by_0_exception:  
        assert(True)  
    catch exception:  
        assert(False)  
    assert(False)  
}
```

An exception is a runtime error that interrupts the flow of execution if it is not caught in a try-catch block.

Assertions

Learning Outcome

A decorative graphic consisting of a purple rectangular frame. To the top-left of the frame are three circles: a small magenta one, a medium cyan one, and a large blue one. To the bottom-right of the frame are four circles: a large magenta one, a medium purple one, a small cyan one, and a medium blue one. The text "Perform quality control and assurance activities in a software project" is centered within the purple frame.

**Perform quality control and assurance activities
in a software project**

Contrast quality control and quality assurance

      @iungo_solutions

<https://iungo.solutions/>

What is Quality Control?

Quality control is a reactive process, in which people will inspect a product, identify defects and correct them.

It is used to ensure that defects in the process are corrected when they appear.

What examples of this can you think of in real life?

What is Quality Assurance?

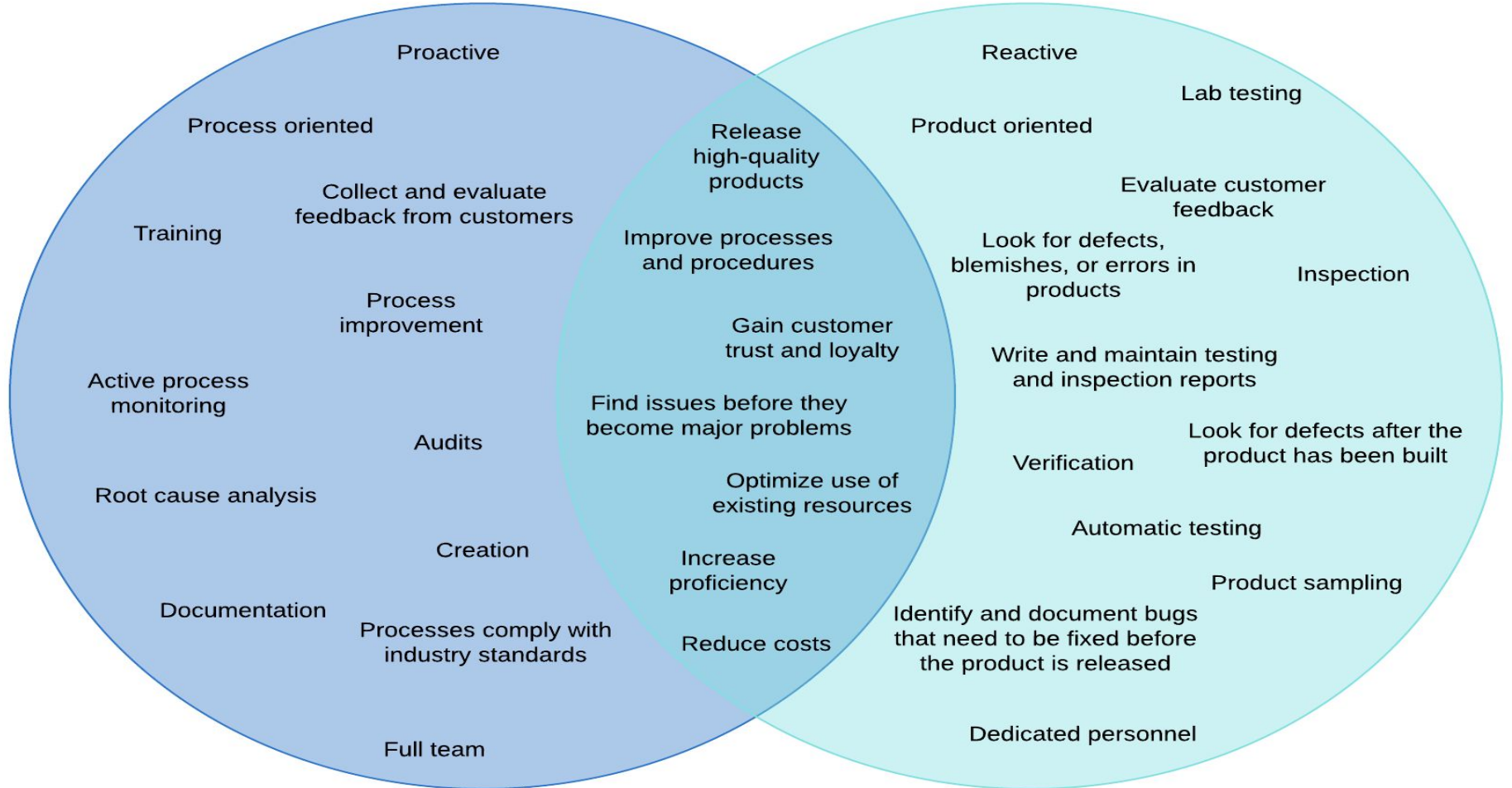
QA stands for **quality assurance**, and, as you may have guessed, it helps us to maintain the quality of our applications and assure that minimal errors occur.

Can you think of any examples of quality assurance in everyday life?

How does this differ from quality control?

QA

QC



QA vs QC

QA is like planning the recipe so the dish turns out well every time.

QC is like tasting the food to make sure it actually turned out right.



**Quality control
and assurance**

QA mitigates QC

The role of QA is to mitigate issues before they become a problem. This can be done through a number of methods:

- Performing integration and regression tests when a piece of software is updated.
- Ensuring all members of the team have the correct training and coding practices.

Correct Coding Practices

One of the main aims of QA is to mitigate issues before they arise, one of the best ways to do so is to ensure the whole team follows the correct coding practices:

- Standardised naming conventions.
- Well documented code (comments and doc strings).
- Well spaced and readable code.

Explain error handling techniques in programming

      @iungo_solutions

<https://iungo.solutions/>

Error handling techniques

When something goes wrong, the program may throw an **exception**.

It could happen for any reason: trying to divide a number by 0, trying to perform an operation on a data type that doesn't support it, trying to access the element of a list using an index that is out of bounds, ...

We can also manually throw exceptions with the keyword **throw**.

Exception should have meaningful names and a message to make it easy to understand what happened just by reading the name.

Exceptions

Error handling techniques

In the real world we have:

functions that call

functions that call

functions that call ...



When an exception is thrown, if it is not dealt with, it propagates to the function that called the current one. It bubbles up until it is caught in a **try-catch** block, or it crashes the program.

Exceptions

Error handling techniques

Some languages don't support exceptions, and functions signal errors with **return codes**: usually non-zero integers indicate an error and zero represents success.

It's crucial to have documentation to know what each code means.

Another technique is **logging**: it involves recording information about the execution of a program, including errors, warnings, and informational messages.

Then the logging data can be stores in a file and kept as a historical record to diagnose old issues.

Perform error handling techniques in iungo a programming language

      @iungo_solutions

<https://iungo.solutions/>

Try, Catch and Finally

The code that could throw the exception that you want to handle goes inside the **try** block.

If the program throws an exception, it jumps to the first **catch** block for the same type of exception. If there's no error, the catch blocks are skipped.

The **finally** block of code will always run, regardless of what happened before, even after the return line. It is used to clean up resources like an opened file.

Error handling techniques

```
string file_path = "~/file_name.txt"

try:
    FileReader file = open(file_path)
    print("File opened successfully.")
    print(read(file))
catch FileNotFoundError:
    print("Error: File not found at path: " + file_path)
catch Error as e:
    print("Error: An error occurred: " + e)
finally:
    close(file)
```

Try, catch and
finally

Error Battleship

Phase 1 - Groups will be given some functions to implement.

Phase 2 - Groups will come up with funny arguments to break other groups' functions. Be meticulous.

Phase 3 - Implement error handling to fix the vulnerabilities found by the other groups.



Try, catch and finally

Identify debugging tools in Integrated Development Environments

      @iungo_solutions

<https://iungo.solutions/>

Debugging Tools in IDEs

Most IDEs come with debugging tools to help developers identify and bugs in the code efficiently, like:

- **Line breakpoints**: put a breakpoint on a line of code and the execution will pause there, to allow you to examine the state of the program at that point.
 - Conditional Breakpoints: the execution pauses only if a condition is true.
 - Watchpoints: the execution pauses when the value of a variable changes.
- **Expression evaluation**: after pausing the execution you can check the value of a variable or an expression at that point.
- **Step execution**: after pausing the execution you can proceed slowly, one line at a time, step into a function call, or return to return to where the function was called.



Utilise debugging tools in an Integrated Development Environment

      @iungo_solutions

<https://iungo.solutions/>

Debugging Tools in IDEs

We'll now look at some real code executed and debugged.

You'll probably notice that the syntax is different from what we're used to. So far we haven't used any specific language, but we'll use Python just for this example.

```
def triangular_number(n: int) -> int:
    sum: int = 0
    for i in range(n):
        sum += i
    return sum

print(triangular_number(5))
```

Debugging Tools in IDEs

We'll now look at some real code executed and debugged.

You'll probably notice that the syntax is different from what we're used to. So far we haven't used any specific language, but we'll use Python just for this example.

```
def triangular_number(n: int) -> int:
    sum: int = 0
    for i in range(n):
        sum += i
    return sum

print(triangular_number(5))
```