Testing and Tools Basics

Functional Testing:

Functional testing is a type of software testing that focuses on verifying that the product's functionality is working correctly according to the specifications and requirements. To test a product in functional testing, you can follow the below steps:

1. **Identify the functional requirements:** The first step is to identify the functional requirements of the product. You can refer to the product specification documents or user stories to determine the requirements.
2. **Create test cases:** Once you have identified the requirements, you can create test cases to verify the functionality of the product. Test cases should cover all the functional requirements of the product.
3. **Prioritize the test cases:** You can prioritize the test cases based on the criticality of the functional requirement and the risk associated with it.
4. **Execute the test cases:** After prioritizing the test cases, you can execute the test cases manually or using automation tools. During test case execution, you should record the results and any defects found.
5. **Report defects:** If any defects are found during test case execution, you should report them to the development team. Defect reports should include a detailed description of the defect, the steps to reproduce it, and the expected results.
6. **Retest defects:** Once the development team fixes the defects, you should retest them to ensure they have been resolved.
7. **Verify the functionality:** Finally, you should verify that all the functional requirements of the product have been implemented correctly.

By following the above steps, you can perform functional testing of your product and ensure that it meets all the functional requirements.

Integration Testing:

Integration testing is a type of software testing that focuses on verifying that the different components or modules of the product work together as expected. To test a product for integration testing, you can follow the below steps:

1. **Identify the modules:** The first step is to identify the modules or components of the product that need to be tested for integration.
2. **Determine the integration points:** Once you have identified the modules, you can determine the integration points where the modules interact with each other.
3. **Create integration test cases:** You can create integration test cases that cover the integration points and verify that the modules work together as expected.
4. **Prioritize the integration test cases:** You can prioritize the integration test cases based on the criticality of the integration point and the risk associated with it.
5. **Execute the integration test cases:** After prioritizing the integration test cases, you can execute them manually or using automation tools. During test case execution, you should record the results and any defects found.
6. **Report defects:** If any defects are found during integration testing, you should report them to the development team. Defect reports should include a detailed description of the defect, the steps to reproduce it, and the expected results.
7. **Retest defects:** Once the development team fixes the defects, you should retest them to ensure they have been resolved.
8. **Verify the integration:** Finally, you should verify that all the modules of the product work together as expected and that the integration points are functioning correctly.

By following the above steps, you can perform integration testing of your product and ensure that the different modules or components work together seamlessly.

Some of the popular tools for Functional testing and integration testing are:

1. **Selenium:** Selenium is an open-source testing framework that supports various programming languages and operating systems. It can be used for functional and integration testing of web applications.
2. **SoapUI:** SoapUI is a popular tool for testing web services and APIs. It supports multiple protocols and standards, including SOAP, REST, HTTP, and JMS.
3. **Postman:** Postman is another popular tool for testing APIs. It provides a user-friendly interface for creating and executing API requests and supports multiple protocols and data formats.
4. **JMeter:** JMeter is an open-source load testing tool that can be used for integration testing of web applications and APIs. It supports various protocols and can simulate thousands of users and transactions.

System Testing:

System testing is a type of software testing that focuses on verifying the entire system or product's behavior as a whole. It is performed to ensure that the product meets the specified requirements and is ready for release. To test a product for system testing, you can follow the below steps:

1. **Define the scope of system testing:** The first step is to define the scope of system testing. This includes identifying the functionalities and features of the product that need to be tested.
2. **Create system test cases:** Once you have defined the scope, you can create system test cases that cover all the functionalities and features. The test cases should be designed to verify the product's behavior as a whole, including its user interface, performance, security, and usability.
3. **Prioritize the system test cases:** You can prioritize the system test cases based on the criticality of the functionality and feature and the risk associated with it.
4. **Execute the system test cases:** After prioritizing the system test cases, you can execute them manually or using automation tools. During test case execution, you should record the results and any defects found.
5. **Report defects:** If any defects are found during system testing, you should report them to the development team. Defect reports should include a detailed description of the defect, the steps to reproduce it, and the expected results.
6. **Retest defects:** Once the development team fixes the defects, you should retest them to ensure they have been resolved.
7. **Verify the system behavior:** Finally, you should verify that the product's behavior as a whole meets the specified requirements. This includes verifying the product's user interface, performance, security, and usability.

By following the above steps, you can perform system testing of your product and ensure that it is ready for release. System testing helps to verify that the product meets the specified requirements and behaves as expected as a whole.

Some of the popular tools for system testing are:

1. **Selenium:** Selenium is an open-source testing framework that supports various programming languages and operating systems. It can be used for system testing of web applications.

2. **JMeter:** JMeter is an open-source load testing tool that can be used for system testing of web applications. It can simulate thousands of users and transactions and test the performance and scalability of the system.
3. **Test Complete:** Test Complete is another commercial testing tool that can be used for system testing of desktop, web, and mobile applications. It provides a user-friendly interface for creating and executing test cases and supports various programming languages and frameworks.

These tools are just a few examples of the many available tools for system testing. You should evaluate your specific project requirements, team skills, and budget to choose the best tool for your system testing needs.

Functional and Integration Testing Tools:

Test the product using SoapUI for functional testing and Integration testing:

To test a product using **SoapUI** for functional testing, you can follow the below steps:

1. Install and launch the **SoapUI** tool on your system.
2. Create a new project in **SoapUI** by clicking on the "File" menu, then selecting "New SoapUI Project."
3. Enter the name of your project, select the appropriate SOAP version, and enter the WSDL URL for the product you want to test.
4. Once you have created the project, you can create a new test suite by clicking on the "Add" button in the test suite window.
5. In the test suite window, you can create a new test case and name it according to your requirements.
6. Add the test steps to your test case by clicking on the "Add" button in the test case window.
7. In the test steps, you can specify the SOAP request by entering the endpoint URL, selecting the appropriate HTTP method, and entering the request body.
8. After specifying the SOAP request, you can execute the test case by clicking on the "Run" button in the test case window.
9. Once the test case execution is complete, you can view the test results in the "Test Case Execution" window.
10. Analyze the test results and fix any errors or issues in the SOAP requests.

By following the above steps, you can use **SoapUI** for functional testing of your product. Additionally, you can also use **SoapUI** for performance testing, security testing, and load testing of your product.

Test a RESTful API using SoapUI for functional testing:

1. **Create a new SoapUI project:** Launch SoapUI and create a new project by clicking on the "New SoapUI Project" button. Enter a name for the project and select "REST" as the project type.
2. **Add a REST API definition:** Add a REST API definition to the project by clicking on the "Add REST Service" button. Enter the API endpoint and any required parameters or headers.
3. **Add a REST request:** Add a REST request to the API definition by clicking on the "Add REST Resource" button. Enter the resource path and any required parameters or headers.
4. **Test the REST request:** After adding the REST request, you can test it to ensure it is working as expected. Click on the "Test Request" button to send a request to the API and receive a response. SoapUI provides a response section that displays the response from the API, including the status code, headers, and body.
5. **Create a test case:** Once the REST request is tested, you can create a test case to validate the API response. Click on the "Create Test Case" button to create a new test case. Enter a name for the test case and select the REST request as the test step.
6. **Add assertions:** After creating the test case, you can add assertions to validate the API response. Click on the "Add Assertion" button to add an assertion to the test step. Enter the expected value and select the assertion type.
7. **Execute the test case:** After creating the test case and adding assertions, you can execute the test case to validate the API response. Click on the "Run Test Case" button to run the test case. SoapUI will automatically run the test steps and validate the API response.
8. **Analyze the results:** Once the test case is executed, you can analyze the results to identify defects and areas for improvement. SoapUI provides detailed reports on the test results, including the number of tests passed and failed, the duration of each test, and any defects found.

By following the above steps, you can test your RESTful API using **SoapUI for functional testing**.

Add SoapUi Assertions scripts:

In SoapUI, you can add assertions scripts in the "Assertions" panel of a test step. Here's how you can add assertions scripts in SoapUI:

1. Open your SoapUI project and navigate to the test case you want to add assertions to.
2. Click on the test step you want to add assertions to.
3. In the test step editor, click on the "Assertions" tab.

4. Click on the "+" icon to add a new assertion.
5. Select the type of assertion you want to add (e.g. "Script Assertion").
6. In the assertion editor, you can write your assertion script in the "Script" tab.
7. Once you've written your assertion script, click on the "Save" button to save the assertion.
8. You can add multiple assertions to a test step by repeating steps 4-7.
9. Once you've added all your assertions, you can run your test case to see if your assertions pass or fail.

Here are some example test scripts you can use in SoapUI to test your product:

1. Test if the response status code is 200:

```
assert context.response.httpResponse.statusCode == 200
```

2. Test if the response contains a certain string:

```
assert context.response.responseContent.contains("expected string")
```

3. Test if the response time is less than a certain value:

```
assert context.timeTaken < 200
```

4. Test if the response JSON is valid:

```
import groovy.json.JsonSlurper
def responseJson = new JsonSlurper().parseText(context.response.responseContent)
assert responseJson.property1 == "expected value"
```

5. Test if the response header contains a certain value:

```
assert context.response.httpResponse.getHeaderField("header-name") == "expected value"
```

6. Test if the response body matches a regular expression:

```
assert context.response.responseContent ==~ /expected regular expression/
```

7. Test if the response is not empty:

```
assert context.response.responseContent != null && context.response.responseContent != ""
```

You can use these assertion scripts in your SoapUI test cases to test your product's functionality.

REST Project in SoapUI

1.Create a REST Project
2.Add a REST request
3.Add request parameters
4.Create a Test Case

5.Add assertions
6.Run and Validate

SOAP Project in SoapUI

1.Create a SOAP API Project
2.Add WSDL
3.Create Test Suite - Test Cases
4.Add Assertions
5.Run Test Step - Test Case - Test Suite
6.Run in sequence and in parallel
7.Create API Documentation
8.Add Load Test and Security Test
9.Run Load Test and Security Test
10.Launch Test Runner
11.Launch Load Test Runner
12.Launch Security Test Runner
13.Deploy as war

Test the product using Postman for functional testing and Integration testing:
Postman is a popular tool used for testing RESTful APIs. To test a product using Postman, you can follow the below steps:

1. **Create a collection:** The first step is to create a collection in Postman to group together the API requests. You can create a new collection by clicking on the "New" button in Postman and selecting "Collection".
2. **Add requests to the collection:** Once the collection is created, you can add API requests to the collection. You can add requests manually or import them from a file or URL. To add a request manually, click on the "New" button in Postman and select "Request". Enter the API endpoint and any required parameters or headers.
3. **Test the requests:** After adding the requests to the collection, you can test them to ensure they are working as expected. You can run the requests individually or in batches to save time. Postman provides a response section that displays the response from the API, including the status code, headers, and body.
4. **Create test cases:** Once the requests are tested, you can create test cases to validate the API responses. You can create test cases using Postman's built-in testing framework, which supports various scripting languages such as JavaScript.

5. **Execute the test cases:** After creating the test cases, you can execute them to validate the API responses. Postman provides a test runner that allows you to run the test cases automatically.
6. **Analyze the results:** Once the test cases are executed, you can analyze the results to identify defects and areas for improvement. Postman provides detailed reports on the test results, including the number of tests passed and failed, the duration of each test, and any defects found.

By following the above steps, you can test your product using Postman. Postman provides a user-friendly interface for creating and executing API requests and supports various scripting languages, making it a popular choice for API testing.

Test a RESTful API using Postman:

1. **Create a new collection:** Launch Postman and create a new collection by clicking on the "New" button. Enter a name for the collection and click on "Create."
2. **Add a request to the collection:** Add a request to the collection by clicking on the "New Request" button. Enter the API endpoint and any required parameters or headers.
3. **Test the request:** After adding the request, you can test it to ensure it is working as expected. Click on the "Send" button to send a request to the API and receive a response. Postman provides a response section that displays the response from the API, including the status code, headers, and body.
4. **Create a test case:** Once the request is tested, you can create a test case to validate the API response. Click on the "Tests" tab and write a test case in the scripting language of your choice. For example, if you want to validate that the response status code is 200, you can write a test case as follows:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});
```

5. **Execute the test case:** After creating the test case, you can execute it to validate the API response. Click on the "Send" button to send the request again and execute the test case. Postman will automatically run the test case and validate the API response.
6. **Analyze the results:** Once the test case is executed, you can analyze the results to identify defects and areas for improvement. Postman provides detailed reports on the test results, including the number of tests passed and failed, the duration of each test, and any defects found.

7. **Add more requests and test cases:** After testing the first request, you can add more requests to the collection and create more test cases to validate the API responses.

By following the above steps, you can test your RESTful API using Postman for functional testing. Postman provides a user-friendly interface for creating and executing API requests, as well as a robust testing framework for validating API responses.

Here are some example test scripts you can use in Postman to test your product:

1. Test if the response status code is 200:

```
pm.test("Response status code is 200", function () {
    pm.response.to.have.status(200);
});
```

2. Test if the response contains a certain string:

```
pm.test("Response body contains string", function () {
    pm.expect(pm.response.text()).to.include("expected string");
});
```

3. Test if the response time is less than a certain value:

```
pm.test("Response time is less than 200ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(200);
});
```

4. Test if the response JSON schema is valid:

```
pm.test("Response JSON schema is valid", function () {
    pm.response.to.have.jsonSchema({
        "type": "object",
        "properties": {
            "property1": { "type": "string" },
            "property2": { "type": "number" }
        },
        "required": ["property1", "property2"]
    });
});
```

5. Test if the response header contains a certain value:

```
pm.test("Response header contains value", function () {
    pm.expect(pm.response.headers.get("header-name")).to.include("expected value");
});
```

6. Test if the response body matches a regular expression:

```
pm.test("Response body matches regular expression", function () {
    pm.expect(pm.response.text()).to.match(/expected regular expression/);
```

```
});
```

7. Test if the response is not empty:

```
pm.test("Response is not empty", function () {
    pm.response.to.not.be.empty;
});
```

You can use these test scripts in your Postman requests to test your product's functionality and ensure that it meets your requirements.

System Testing Tools

Test the product using Test Complete for System testing:

Test Complete is a commercial testing tool that can be used for system testing of desktop, web, and mobile applications. To test a product using Test Complete for system testing, you can follow the below steps:

1. **Create a project:** The first step is to create a Test Complete project and add the system under test to the project. You can do this by clicking on the "New Project" button in Test Complete and selecting the appropriate project type based on the system under test.
2. **Create test cases:** Once the project is created, you can create test cases to test the system's functionality. You can create test cases manually or record them using Test Complete's record and playback feature.
3. **Execute test cases:** After creating the test cases, you can execute them to test the system's functionality. You can run the test cases individually or in batches to save time.
4. **Analyze results:** Once the test cases are executed, you can analyze the results to identify defects and areas for improvement. Test Complete provides detailed reports on the test results, including the number of tests passed and failed, the duration of each test, and any defects found.
5. **Debug defects:** If any defects are found during testing, you can use Test Complete's debugging tools to identify the root cause of the defects. Test Complete provides a debugger that allows you to step through the code and analyze variables and objects.
6. **Retest defects:** After fixing the defects, you should retest them to ensure they have been resolved. You can use Test Complete's regression testing feature to retest the defects automatically.

By following the above steps, you can test your product using Test Complete for system testing. Test Complete provides a user-friendly interface for creating and

executing test cases and supports various programming languages and frameworks, making it a popular choice for system testing.

Ad-hoc Testing:

Ad-hoc testing is an informal testing approach that involves exploring the product's functionality and features without a specific plan or test cases. It is performed to identify defects that may not be found during formal testing approaches. To test a product for ad-hoc testing, you can follow the below steps:

1. **Explore the product:** The first step in ad-hoc testing is to explore the product's functionality and features without a specific plan or test cases. This can be done by clicking on various buttons, links, and menus to see how they behave.
2. **Try different scenarios:** Once you have explored the product, you can try different scenarios to see how the product responds. This can include entering invalid data, performing actions out of order, and using the product in unexpected ways.
3. **Record defects:** If any defects are found during ad-hoc testing, you should record them. Defect reports should include a detailed description of the defect, the steps to reproduce it, and the expected results.
4. **Prioritize defects:** After recording defects, you should prioritize them based on the severity of the defect and the impact it has on the product's functionality.
5. **Report defects:** If any defects are found during ad-hoc testing, you should report them to the development team. Defect reports should include a detailed description of the defect, the steps to reproduce it, and the expected results.
6. **Retest defects:** Once the development team fixes the defects, you should retest them to ensure they have been resolved.

By following the above steps, you can perform ad-hoc testing of your product and identify defects that may not be found during formal testing approaches. Ad-hoc testing helps to identify defects that may be missed during formal testing approaches and can improve the overall quality of the product.