**Summary**

Great work! In this module, we defined a *functional test* as a test that verifies that a given function of the software does what it should. We also defined some popular kinds of functional tests, including smoke tests, unit tests, and UI tests.

The Tailspin team chose one type of functional test to automate first: UI tests. Amita had been running these tests manually. The manual tests were tedious and time-consuming. By automating the tests, Amita can now focus more on usability testing, which is usually best done by humans.

When you incorporate functional tests in your pipeline, focus on the kinds of tests that matter most. For example, if your application distributes work among various components, you can write integration tests that verify that all components work together to form a complete system.

In the next module, you'll switch to another type of important testing: *nonfunctional testing*. The team will define nonfunctional testing and focus on one type of performance test.

**Learn more**

Use the following resources to learn more.

**Learn more about NUnit**

In this module, you worked with NUnit tests. If you're interested in unit-testing .NET applications, consider these additional resources:

* NUnit.org
* Unit testing your code
* Unit testing C# by using NUnit and .NET Core

**Learn about parallel jobs**

You have set up UI tests that run on Windows. But you can also run your tests on other platforms, such as macOS and Linux. To speed up the testing process, you can run all of these jobs in parallel. For more information, see Jobs  and Parallel jobs .

**Learn more about Selenium tests**

If you're interested in testing UI by using Selenium, check out these resources to help you go further:

* SeleniumHQ.org
* Selenium IDE
* UI testing by using Selenium
* UI testing considerations
* Use WebDriver (Chromium) for test automation

**Improve code quality by using Azure Test Plans**

Now that you're familiar with automated tests, let's talk about Azure Test Plans , which provides all the tools you need to test your applications. You can create and run manual test plans, generate automated tests, and collect feedback from users.

**Manage test plans, test suites, and test cases**

Azure Test Plans provides three main types of test-management artifacts: test plans, test suites, and test cases.

* Test plans group together test suites and individual test cases. Test plans include static test suites, requirement-based suites, and query-based suites.
* Test suites group test cases into separate testing scenarios within a single test plan. By grouping test cases, you can better see which scenarios are complete.
* Test cases validate individual parts of your code or app deployment. You can ensure your code works correctly, has no errors, and meets business and customer requirements.

**Use the Test & Feedback extension**

The Test & Feedback extension enables exploratory testing techniques in Azure Test Plans. With it, you can:

* Capture your findings along with rich diagnostic data. This data includes comments, screenshots with annotations, and audio or video recordings that describe your findings and highlight issues. In the background, the extension captures additional information such as user actions. The extension uses the image action log, page load data, and system information about the browser, operating system, and more. This information later helps you debug or reproduce the issue.
* Create work items such as bugs, tasks, and test cases from within the extension. The captured information automatically becomes part of the filed work item and helps with end-to-end traceability.
* Collaborate with your team by sharing your findings. Export your session report or connect to Azure Test Plans for a fully integrated experience.