

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

Why write automated tests?

1. Finding bugs earlier:

To find defects earlier in the development lifecycle process, so fewer bugs making it through to production, which implies fewer defects reaching to end users

2. Reduced Business Cost:

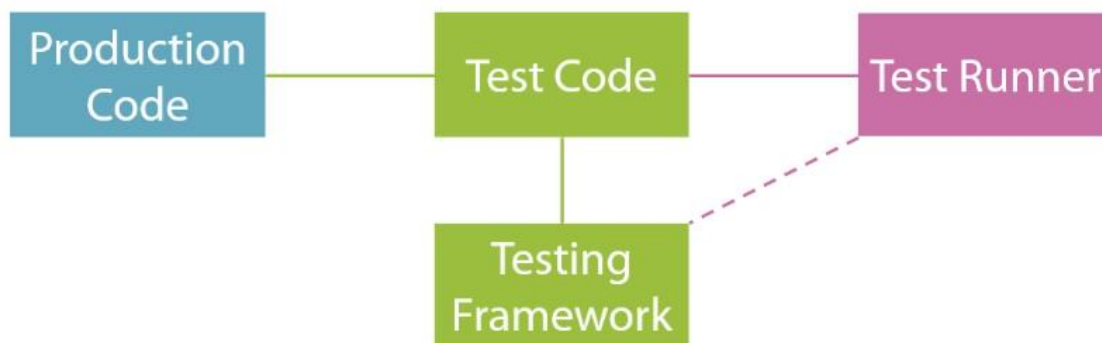
Automated tests can also benefit the business by reducing costs. As by automated tests we can find defects earlier, they are cheaper to fix. If a bug is reported by customer, the developers have to dig into log files, and find the problems. This is an expensive and time-consuming process, as compared to find problems through unit testing.

3. Faster Execution:

Automated tests also give us faster execution of test runs, a suite of tests can be executed more quickly when compared to human manual tests.

Testing Frameworks and Test Runners

Testing Frameworks and Test Runners

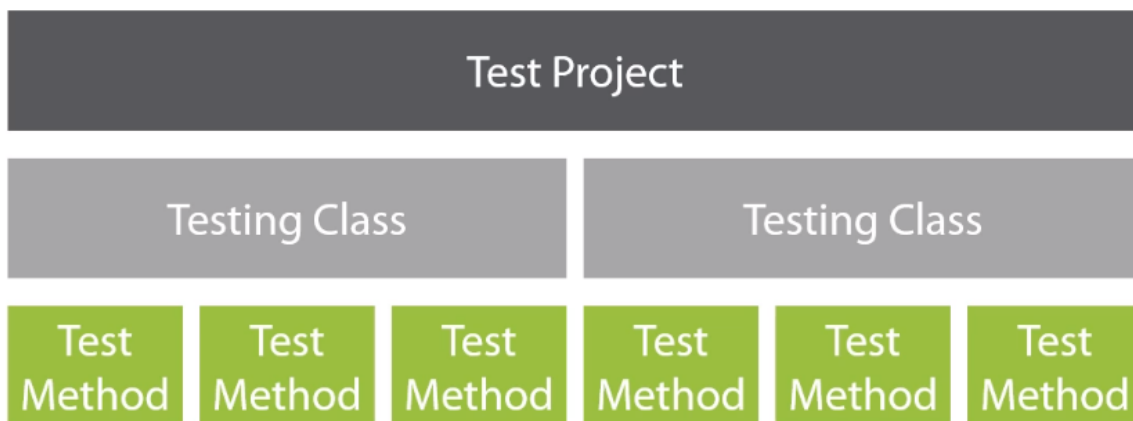


We add a separate test project to our product project. We write test codes in that project. Instead of our manual test code, our test code can make use of prebuilt existing testing framework, and NUnit is our preferred testing

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

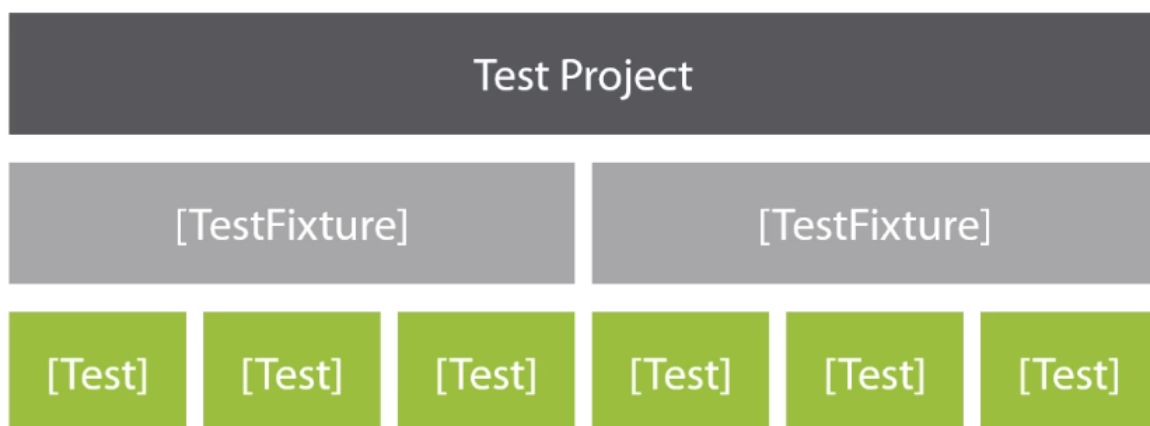
framework. We can execute our test code by using test runners, and this test runner here is provided by Testing Framework, in our case, NUnit.

NUnit Test Suite Organizational Structure



The attributes we are using to in our test project to mark our test class and test methods.

Designating Test Code



UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

Skeleton test project codes are presented here

```
[TestFixture]
public class CalculatorTests
{
    [Test]
    public void ShouldAddTwoNumbers()
    {
        // Test code omitted
    }
}
```

Obviously, this [TestFixture] and [Test] annotations come from our NUnit framework.

All About Asserts

Assets tell the test runners whether a test has passed or failed.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

Example of a brief Assert Function

```
[Test]
public void ShouldAddTwoNumbers()
{
    Assert.That(Calculator.Add(1,2), Is.EqualTo(3));
}
```

The image shows a code snippet for a unit test. The word **[Test]** is in orange. The word **actual** is in a green box with a line pointing to the `Calculator.Add(1,2)` expression. The word **expected** is in a green box with a line pointing to the `Is.EqualTo(3)` expression. The `Assert.That` and `Is.EqualTo` methods are in orange.

Here we are using the NUnit Test Runner to tell the test runner if the test is passed or failed. First parameter of the Assert Method is the actual result from running our production code, and second parameter is our expected value.

The following things ensure a good test.

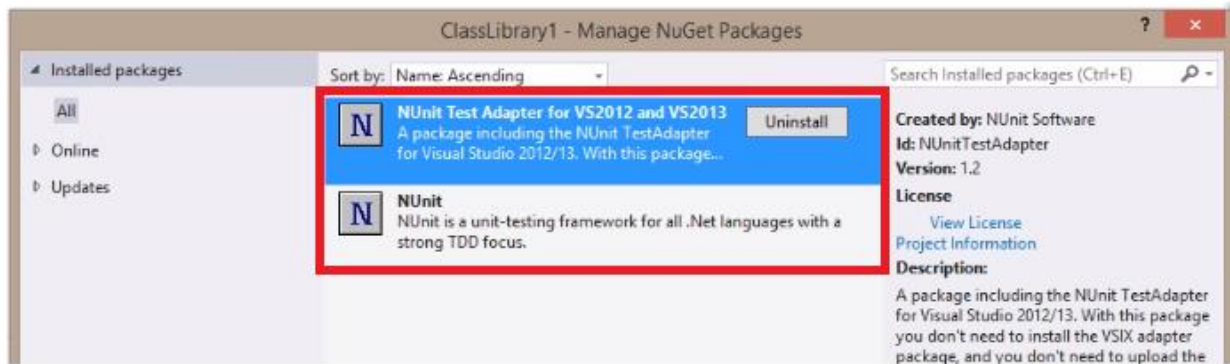
What Makes a Good Test?

- Independent & isolated
- Test single behaviour / logical thing
- Clear intent / readable
- Don't test the compiler
- Reliable & repeatable
- Production quality code
- Valuable



NuGet packages we need to install

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017



To avail the functionality of the test framework functionality, we need the NUnit Nuget package. And to get the NUnit test runner automatically, we need to install the NUnit Test Adapter package.

Different Kind of Asserts

1. Asserting String Equality:

```
var fullName = sut.Join("Sarah", "Smith");  
  
Assert.That(fullName, Is.EqualTo("Sarah Smith"));
```

2. Asserting String Equality with Ignore Case:

```
var fullName = sut.Join("sarah", "smith");  
  
Assert.That(fullName, Is.EqualTo("SARAH SMITH").IgnoreCase);
```

3. Asserting String Inequality:

```
var fullName = sut.Join("Sarah", "Smith");  
  
Assert.That(fullName, Is.Not.EqualTo("Gentry Smith"));
```

4. Assert Float Inequality:

```
var result = sut.AddDoubles(1.1, 2.2);  
  
Assert.That(result, Is.EqualTo(3.3).Within(0.01));
```

Here as the floats will not really fully match, we can add a tolerance of 0.01 with the Within function.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

5. Assert within Range:

In this example, we are testing whether our production result is greater than certain expected result.

```
var sut = new PlayerCharacter {Health = 100};  
  
sut.Sleep();
```

```
Assert.That(sut.Health, Is.GreaterThan(100));
```

In this example, we are testing whether our production result is between a certain expected ranges.

```
var sut = new PlayerCharacter { Health = 100 };  
  
sut.Sleep();
```

```
Assert.That(sut.Health, Is.InRange(101, 200));
```

6. Assert with Null and Booleans:

Here we are checking null, not null, not empty and Boolean True.

```
var sut = new PlayerCharacter();  
  
Assert.That(sut.Name, Is.Not.Empty);
```

```
var sut = new PlayerCharacter();  
  
Assert.That(sut.NickName, Is.Null);
```

```
var sut = new PlayerCharacter();  
  
Assert.That(sut.NickName, Is.Not.Null);
```

```
var sut = new PlayerCharacter();  
  
Assert.That(sut.IsNoob, Is.True);
```

7. Assert With Collections:

This example tests whether any item in a list collection is not empty.

```
var sut = new PlayerCharacter();  
  
Assert.That(sut.Weapons, Is.All.Not.Empty);
```

This example tests whether a production collection or list contains a specific item.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

```
var sut = new PlayerCharacter();  
  
Assert.That(sut.Weapons, Contains.Item("Long Bow"));
```

This example tests whether items in a collection are not repeated.

```
var sut = new PlayerCharacter();  
  
Assert.That(sut.Weapons, Is.Unique);
```

This example checks whether any item in a collection is not equal to a certain item.

```
var sut = new PlayerCharacter();  
  
Assert.That(sut.Weapons, Has.None.EqualTo("Staff Of Wonder"));
```

This example checks whether any collection is equivalent to an expected collection. That means both collections have the same items, but not necessarily in the same order.

```
var sut = new PlayerCharacter();  
  
var expectedWeapons = new[]  
{  
    "Long Bow",  
    "Short Sword",  
    "Short Bow"  
};  
  
Assert.That(sut.Weapons, Is.EquivalentTo(expectedWeapons));
```

8. Asserting Referential Equality:

This example checks if 2 objects have the same reference. Obviously, the following tests will fail.

```
var player1 = new PlayerCharacter();  
var player2 = new PlayerCharacter();  
  
Assert.That(player1, Is.SameAs(player2));
```

9. Asserting that Specific Exceptions are thrown without creating exceptions:

This example checks whether a certain method actually throws an exception for a given set of parameters.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

```
var sut = new Calculator();  
  
Assert.That(() => sut.Divide(200, 0), Throws.Exception);
```

Here we are checking that Divide Method is throwing an exception for a division by zero case, which it should throw.

If we want to check if a function throws a particular kind of exception, we can use the following example.

```
var sut = new Calculator();  
  
Assert.That(() => sut.Divide(200, 0),  
    Throws.TypeOf<DivideByZeroException>());
```

NUnit Deployment in TFS

We have to nothing extra to do this automated testing in TFS. If there is a test project in a TFS project, TFS will automatically detect this, and run all these tests, if all the tests are succeeded, the whole build process will become successful. After a successful build, it will generate a report in the Tests tab of that particular build.


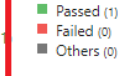
TestQA-.NET Desktop-CI / Build 20180611.1 ☐ Build not retained

[Edit build definition](#) [Queue new build...](#) [Download all logs as zip](#) [Release](#)

Build partially succeeded

Build 20180611.1
Ran for 45 seconds (Default), completed 15 days ago

Summary Timeline Artifacts **Tests**

Total tests	Failed tests	Pass percentage	Run duration	Test failures	Test duration
 1 (+0)	 0 (+0)	100% (+100%)	3s 880ms (+3s 880ms)		

Group by Test run Outcome Failed

This build doesn't have test results for the outcome you've selected.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

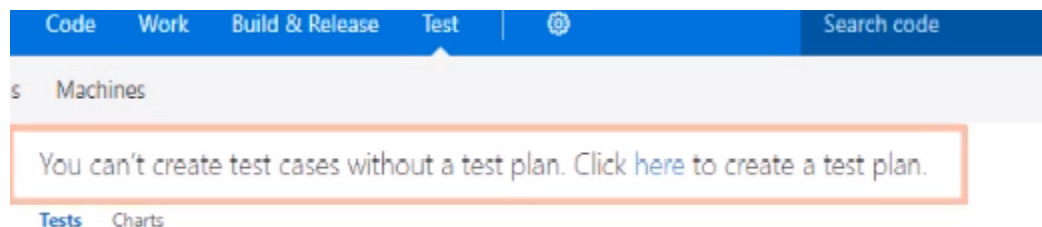
Exploratory Testing

Not all tests can be automated. Even after all kinds of automation, the application needs some manual testing that can be only be done by a human tester. Because the applications will ultimately be used by humans, not some cyborgs.

TFS provides an awesome platform to cover this manual/exploratory testing too. Here the QA/test team can create test plan, build test suits, build test cases on a Product Backlog item and create and save test screen recordings for any product backlog test. We can explore the steps here.

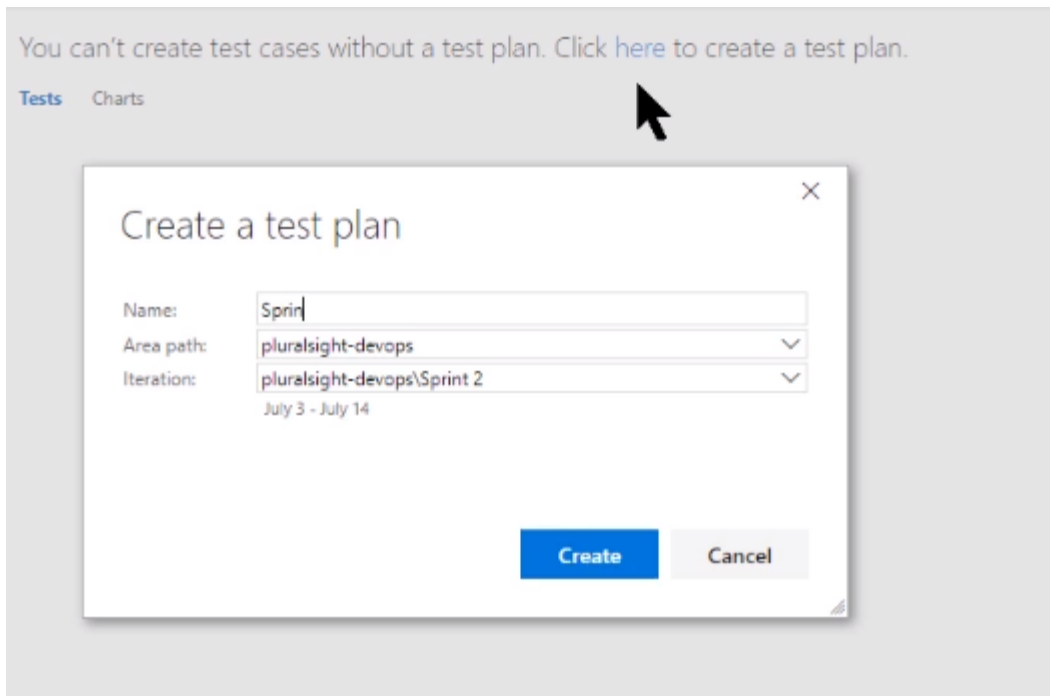
Creating a Test Plan

Every test plan is related to a sprint. So if we don't have a test plan related to a sprint, the following message show when we click Test in the upper menu.



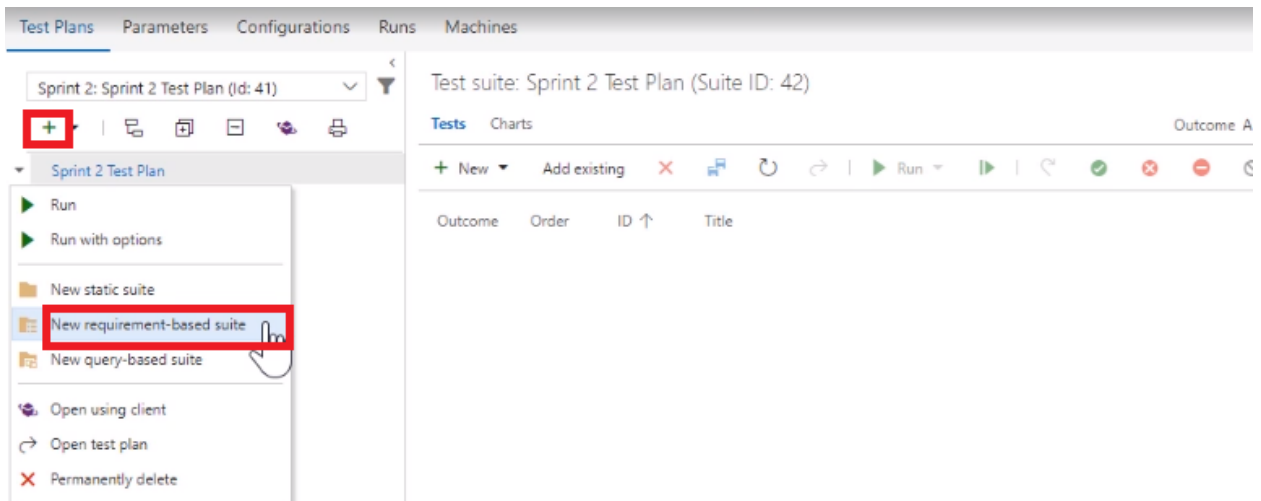
So for a spring, first, we have to create a test plan. By clicking the link here, we shall find a modal like this, we input a test plan name and we get a test plan.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017



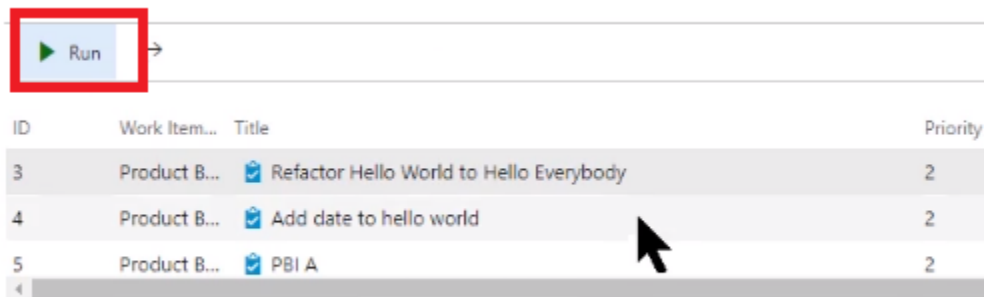
Creating a Test Suite

There is no test suite yet, so we have to create it. We click at the "+" sign in the top left, get a drop down menu, click "New Requirement-based suite".



We get a modal window, we click "Run" and we shall see all the Product Backlog Items in currently working Sprint.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

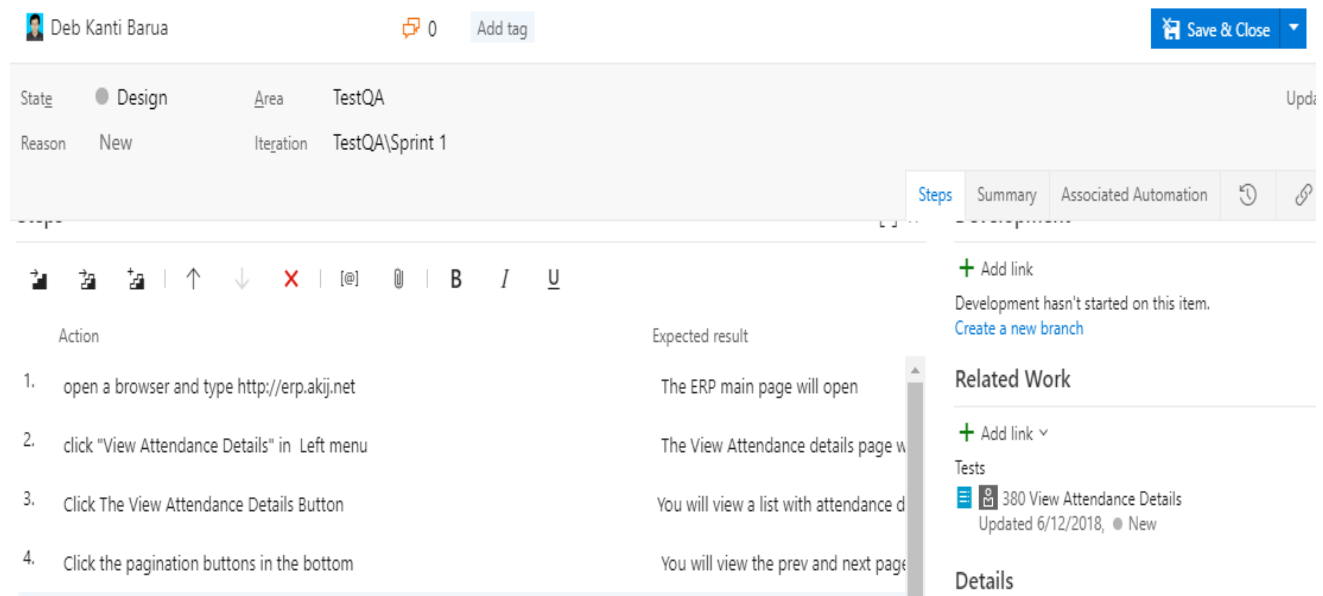


Creating Test Cases

For current test suite selected, we click the “+New” button in the test window, a drop down will appear, click “New Test Case”.



You will get a modal window. Here we can enter the title “View Attendance Details”.

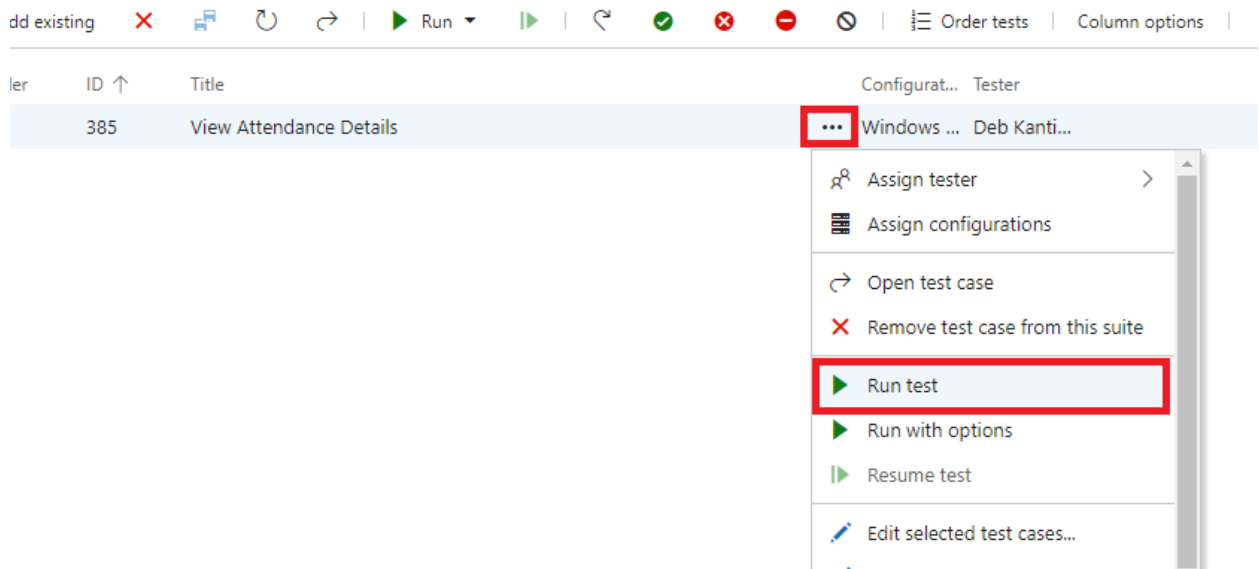


UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

Here in the action and expected result field, we enter all the test cases and expected result and then click “Save & Close” in the upper right corner. The test cases will be saved against the test suite.

Running the test suite

For a test suite, in our example, “View Attendance Details”, we click the corresponding “...” button, get a drop down menu, and click “Run Test”.



A modal window like this image will appear, where all the test cases corresponding to the test suite will appear with the expected results.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

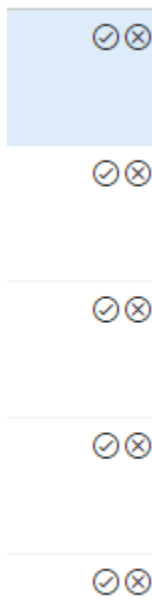
① tfs2017:8080/tfs/DefaultCollection/TestQA/_testExecution/Index

Save and close | Create bug | | +

385: View Attendance Details

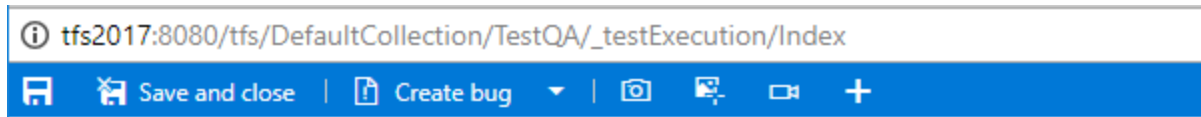
1.	open a browser and type http://erp.akij.net
	EXPECTED RESULT The ERP main page will open
2.	click "View Attendance Details" in Left menu
	EXPECTED RESULT The View Attendance details page will open
3.	Click The View Attendance Details Button
	EXPECTED RESULT You will view a list with attendance details
4.	Click the pagination buttons in the bottom
	EXPECTED RESULT You will view the prev and next page

There will be a passed and failed button like this on the right corresponding to each use case.



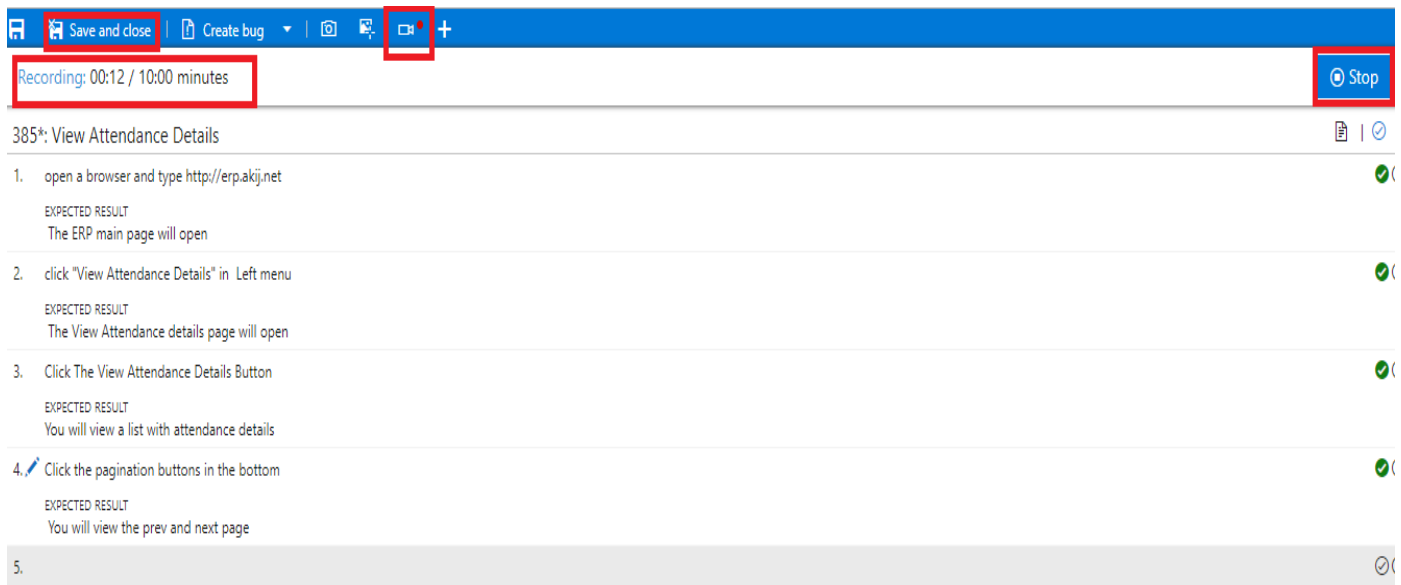
Another thing to note, if we use Google Chrome for TFS, we shall see an upper panel like this right above the test plan,

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017



Here we get the options to get screen recording and screenshot capture for TFS. Remember that, we shall not get this panel for other browsers. But for chrome, we have to do another work. We have to install “Test & Feedback” chrome extension.

Now if we click screen recording, we shall start the recording of the screen. Here we can test all the use case manually, they will be automatically captured in screen recording, and after each manual testing, we can pass or fail each test case by clicking the pass or fail button on the right.

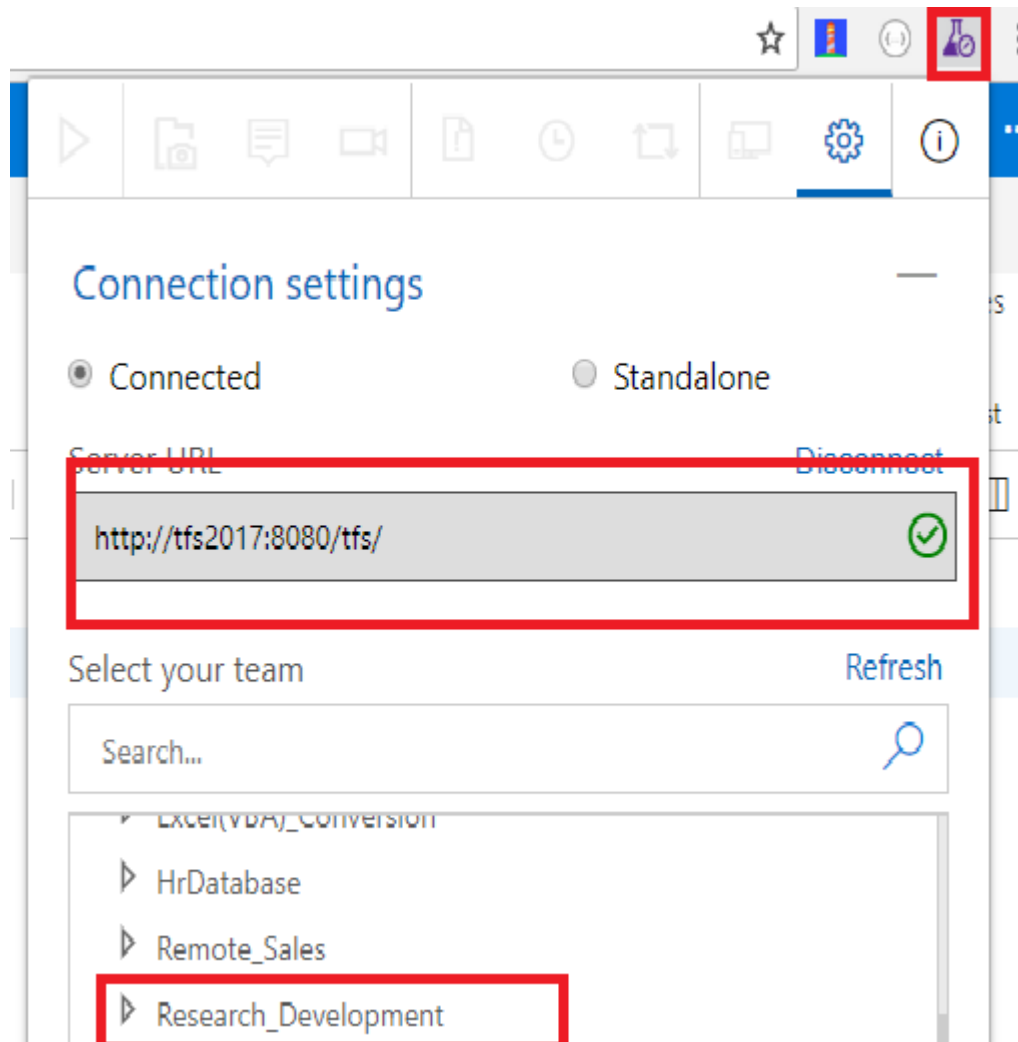


Now if we click save, the screen recording will be saved against the test suite and we can watch it later for manual test verification.

Connecting the “Test and Feedback” extension to the TFS Server

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

Before doing anything about this extension, we have connect it to the TFS server. So we have to click the extension button in the upper right corner, give the TFS server url in proper field, select the TFS project and click “Ok”.



View the data of a test execution

To do that, we have to click the “Runs” menu in the upper panel, and click the last test runs.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

ations **Runs** Machines

Recent test runs

[Test runs](#) [Filter](#)

↻

State	Run Id	Title	Completed Date
✓ Completed	19	380 : View Attendance Details (Manual)	6/26/2018 4:03:32 PM
✓ Completed	16	380 : View Attendance Details (Manual)	6/24/2018 2:55:45 PM
ⓘ Needs investigat...	15	380 : View Attendance Details (Manual)	6/24/2018 2:53:32 PM

We shall double click the corresponding completed test run, we shall find the run summary.

[Run summary](#) [Test results](#) [Filter](#)

↻

Summary

✓ Completed 2 days ago, Ran for 02 minutes 05 seconds

Run type	Manual
Owner	Deb Kanti Barua
Tested build	not available
Release	not available
Release Environment	not available
Build platform	not available
Build flavor	not available
Test settings	Run time settings
MTM lab environment	not available

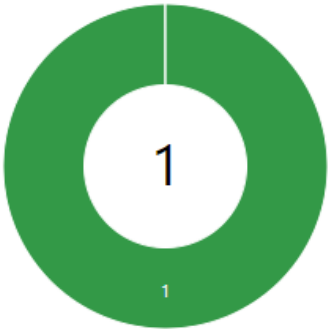
Comments

No comments

Error message

No error message

Outcome



1

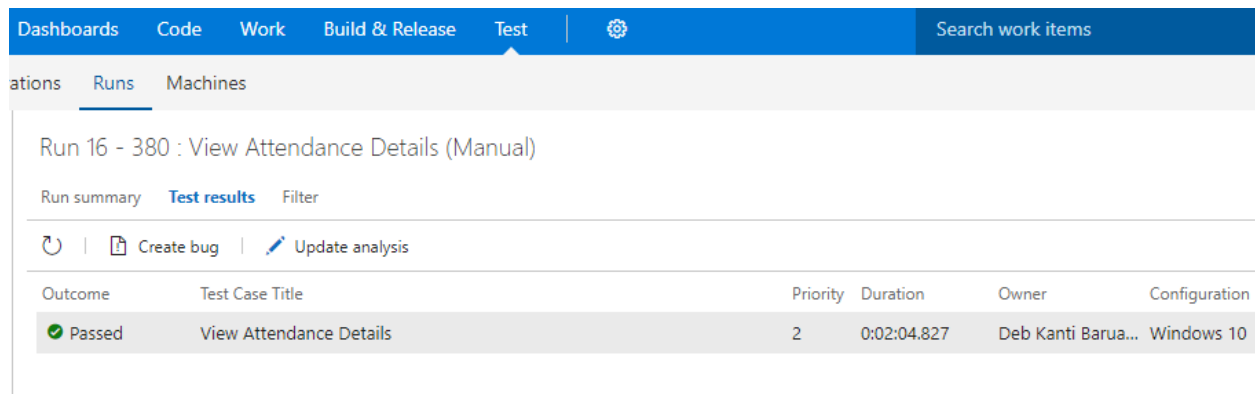
1

■ Passed

Outcome by priority

Now if we click “Test Results” Menu, and we click the corresponding passed test

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

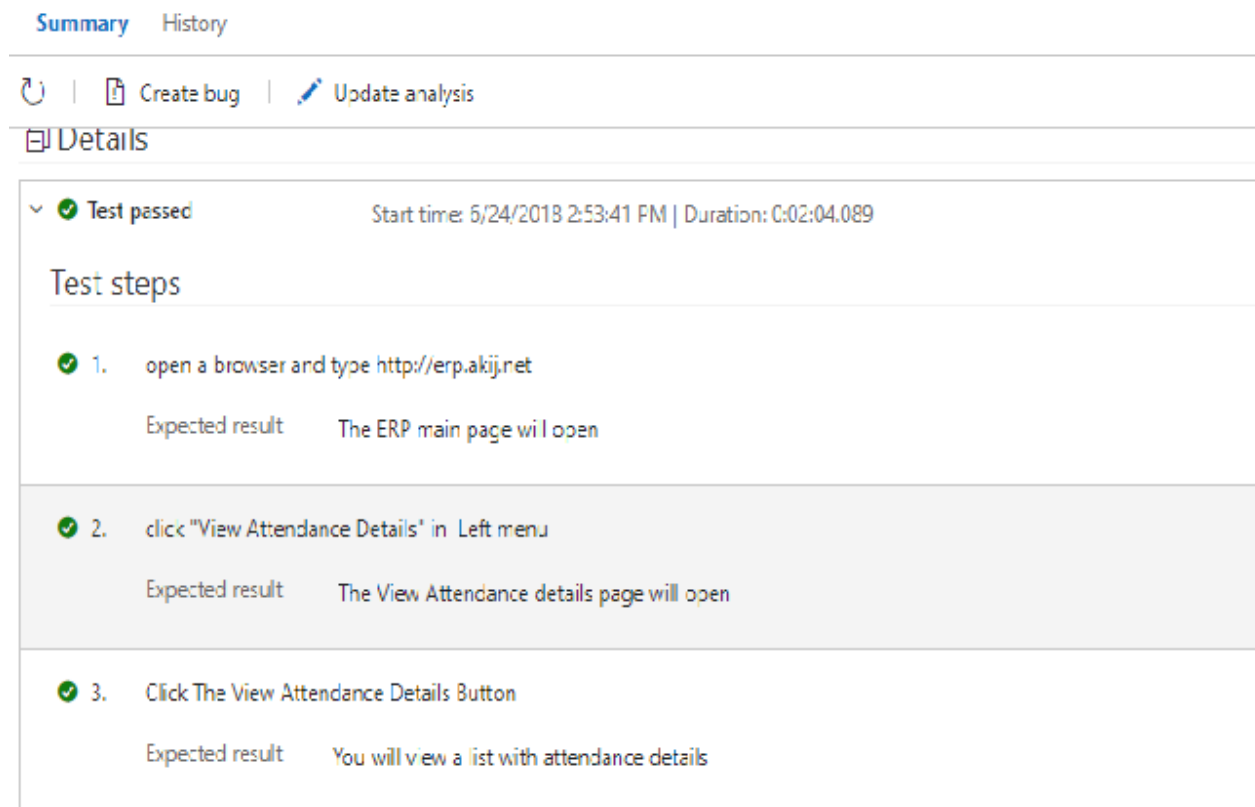


The screenshot shows the TFS 2017 interface with the 'Test' tab selected. Under the 'Runs' sub-tab, a specific test run is displayed. A summary table provides an overview of the test results.

Outcome	Test Case Title	Priority	Duration	Owner	Configuration
Passed	View Attendance Details	2	0:02:04.827	Deb Kanti Barua...	Windows 10

run, we shall get more details on the test run like this.

[Run 16 - 380 : View Attendance Details \(Manual\)](#) / [View Attendance Details](#)



This screenshot provides a detailed view of the test run, including the test steps and their expected results.

Summary History

Details

Test passed Start time: 5/24/2018 2:53:41 PM | Duration: 0:02:04.089

Test steps

1. open a browser and type <http://erp.akij.net>
Expected result The ERP main page will open
2. click "View Attendance Details" in Left menu
Expected result The View Attendance details page will open
3. Click The View Attendance Details Button
Expected result You will view a list with attendance details

And in the bottom portion, we shall get the test screen recording.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

- ✓ 3. Click The View Attendance Details Button

Expected result You will view a list with attendance details

- ✓ 4. Click the pagination buttons in the bottom

Expected result You will view the prev and next page

- ✓ 5.

Result attachments

Name	Size
ScreenRecording-2018-06-24T08-53-53.796Z.webm	1364K

Assign Tester

As like previous task assignment of PBI to tfs project members, we can assign test suite to them also. Just click the ... of the corresponding test suite, a drop down window will appear, we shall pick "Assign Tester", and select our member who is going to be test this test suit, in our example, View Attendance details.

UNIT TESTING AND EXPLORATORY TESTING IN TFS 2017

