Unit Testing

Dawid Obrocki
Customer Engineer
Microsoft UK

Agenda

Morning Sessions

- Introductions
- What and why of testing
- Testing schools of thought
- Unit testing C# with .NET in Visual Studio

Afternoon Sessions

- Running tests in Azure DevOps
- Testing UI components
- Beyond Unit Testing – Accessibility, Integration and Azure DevTest
- Q&A / AMA / BYOC

What and why of testing

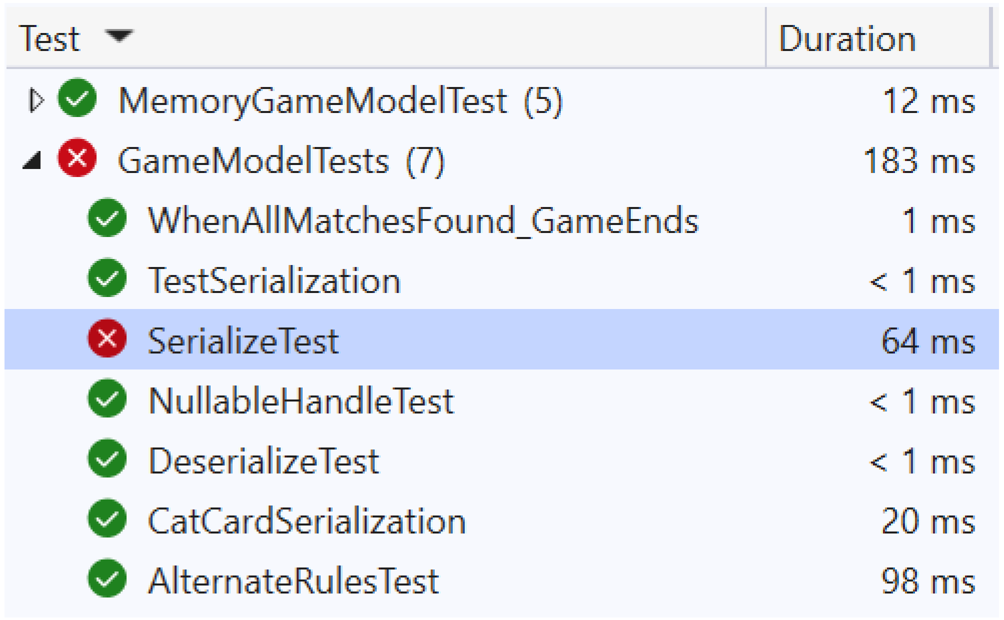
A unit test is a test that exercises individual software components or methods, also known as "unit of work". Unit tests should only test code within the developer's control. They do not test infrastructure concerns. Infrastructure concerns include interacting with databases, file systems, and network resources.

- Unit tests

Smallest testable unit of code
Methods in procedural programming
Classes in OOP
Not multi-layered
No disk and network access

- Zero-box tests

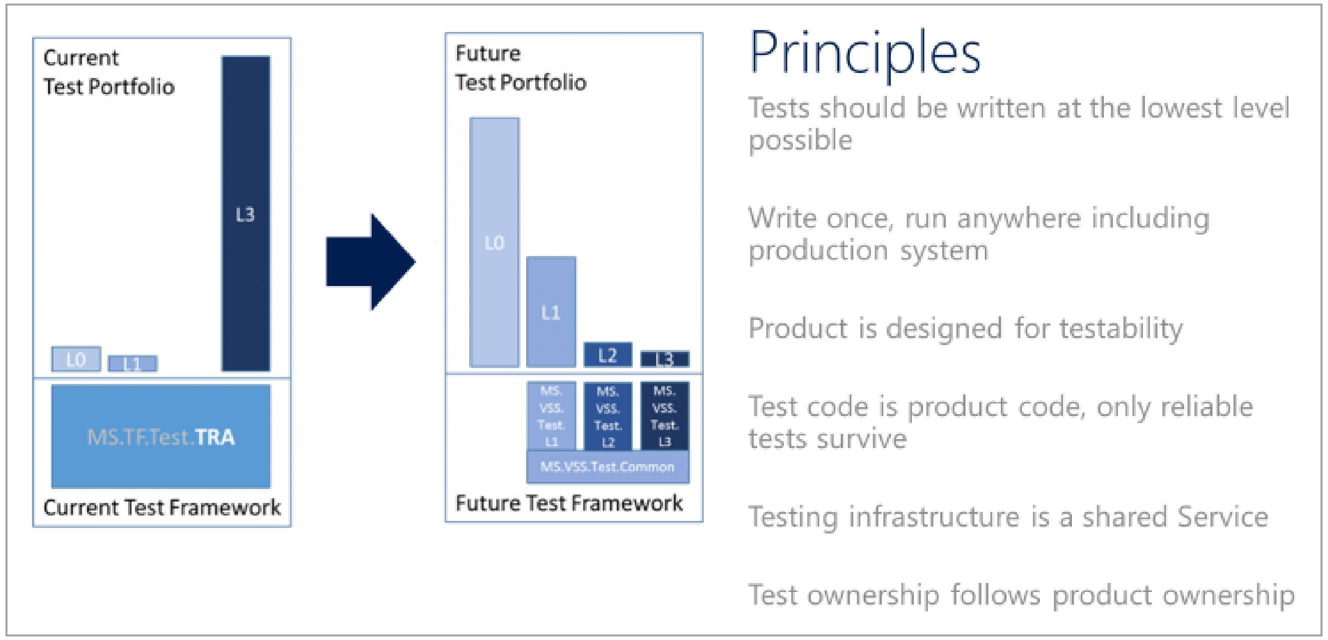
No exchange installation/role required
Fast, stable and deterministic
Can be run on dev. box concurrently with other tests



Test	Duration
▷ ✓ MemoryGameModelTest (5)	12 ms
◢ ✗ GameModelTests (7)	183 ms
✓ WhenAllMatchesFound_GameEnds	1 ms
✓ TestSerialization	< 1 ms
✗ SerializeTest	64 ms
✓ NullableHandleTest	< 1 ms
✓ DeserializeTest	< 1 ms
✓ CatCardSerialization	20 ms
✓ AlternateRulesTest	98 ms

What and why of testing

- Validate code changes and quality
- Industry examples
- Effects on architecture
- Code coverage and code health



Principles

Tests should be written at the lowest level possible

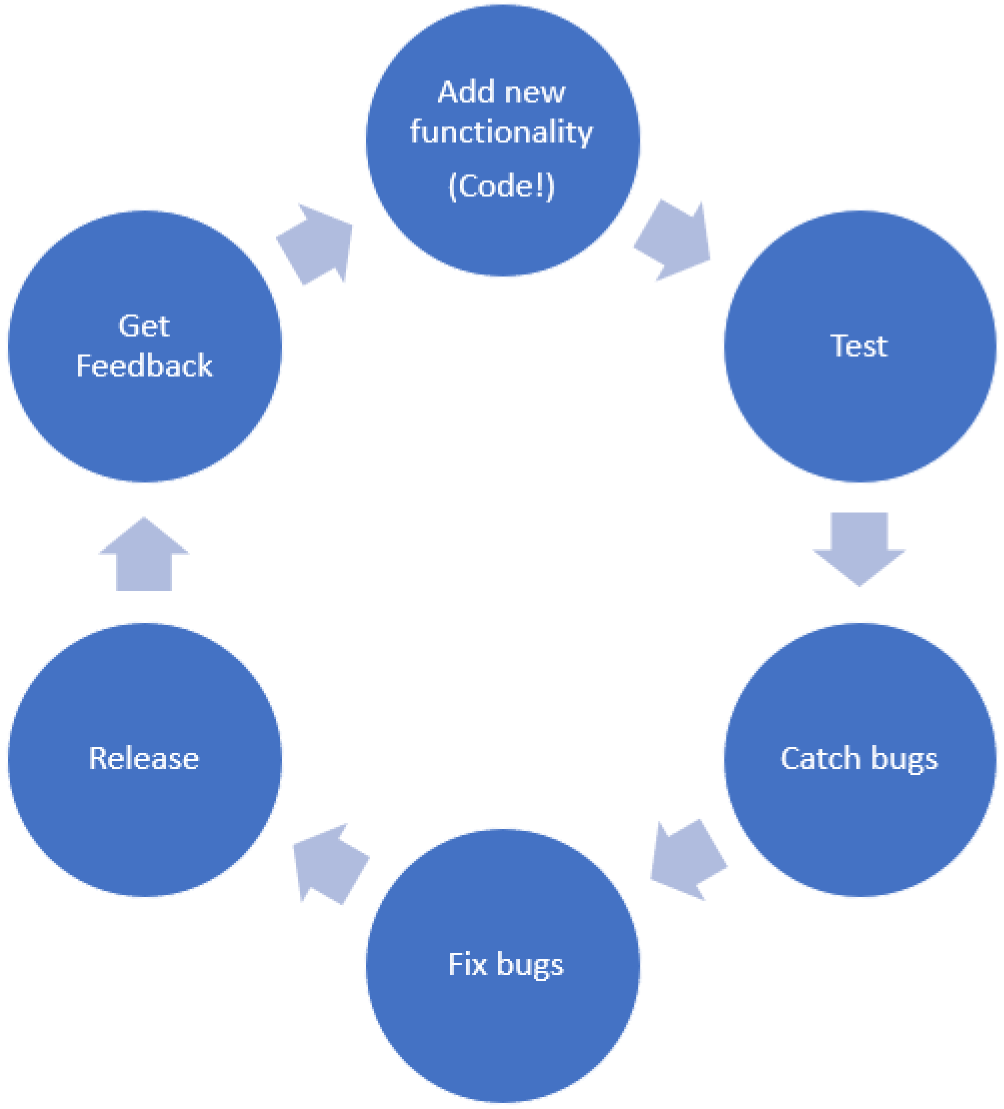
Write once, run anywhere including production system

Product is designed for testability

Test code is product code, only reliable tests survive

Testing infrastructure is a shared Service

Test ownership follows product ownership



Selling the Vision

Unit Tests? Bah!

Some believed in value of unit testing, some didn't
Dredged up experiences of poor unit test practices
Unit tests replace functional tests? That isn't right.

Response

Functional tests tightly coupled to implementation isn't right either… we need both
Lightning fast, rock solid reliability wired into PR
Think of unit tests as a design tool… better code

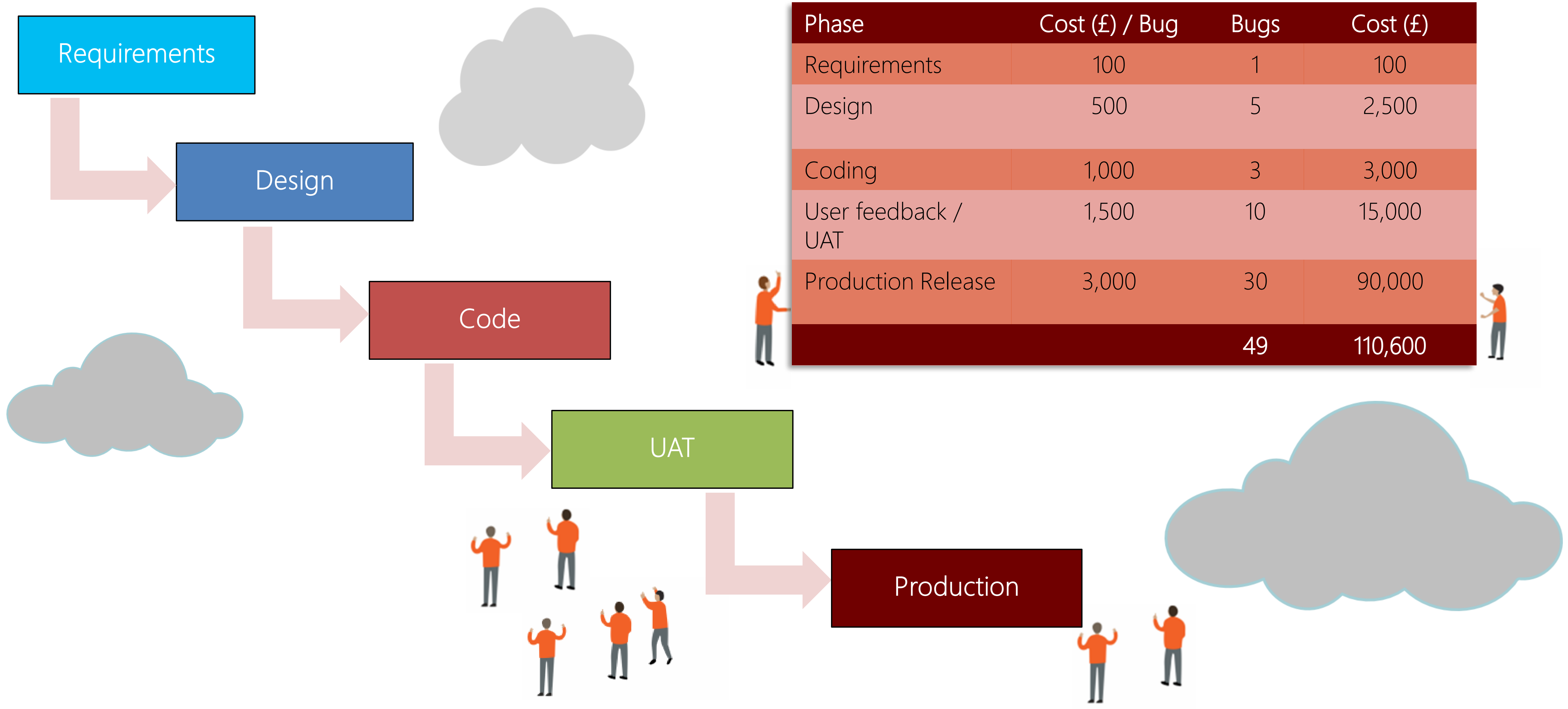
Unit Tests? Finally!

Passionate unit test advocates given a voice
Seen as an opportunity to do it "right"
Philosophical divide: "classical" and "mockist"

Response

Fowler: Mocks aren't stubs frames the debate
Observation: mockist is best for greenfield
Guidance: mockist if you can, classical is fine

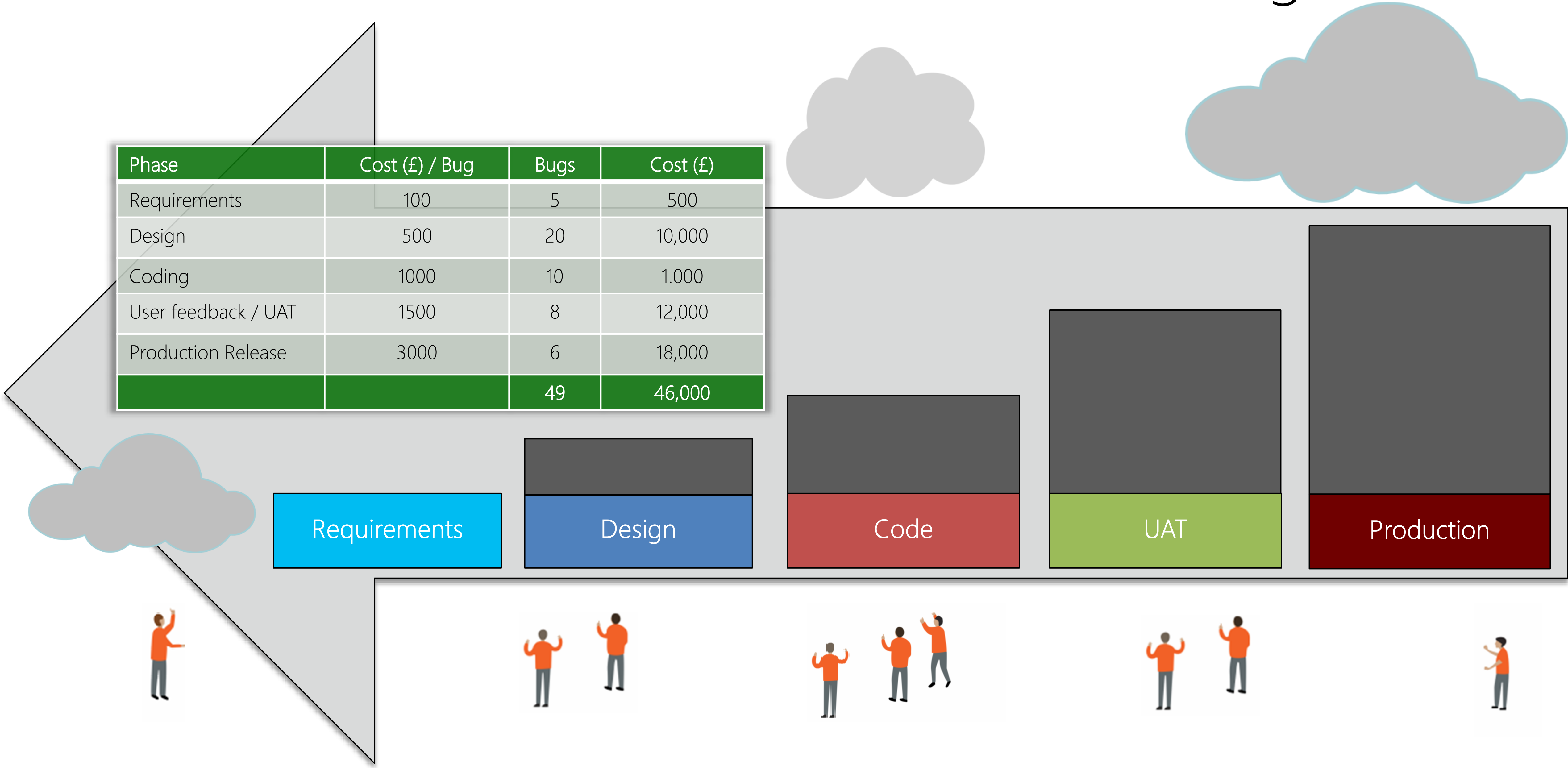
Traditional Testing Approach



Phase	Cost (£) / Bug	Bugs	Cost (£)
Requirements	100	1	100
Design	500	5	2,500
Coding	1,000	3	3,000
User feedback / UAT	1,500	10	15,000
Production Release	3,000	30	90,000
		49	110,600

What Shift Left means in Software Testing?

Phase	Cost (£) / Bug	Bugs	Cost (£)
Requirements	100	5	500
Design	500	20	10,000
Coding	1000	10	1.000
User feedback / UAT	1500	8	12,000
Production Release	3000	6	18,000
		49	46,000



Which of these options is not a benefit of testing?

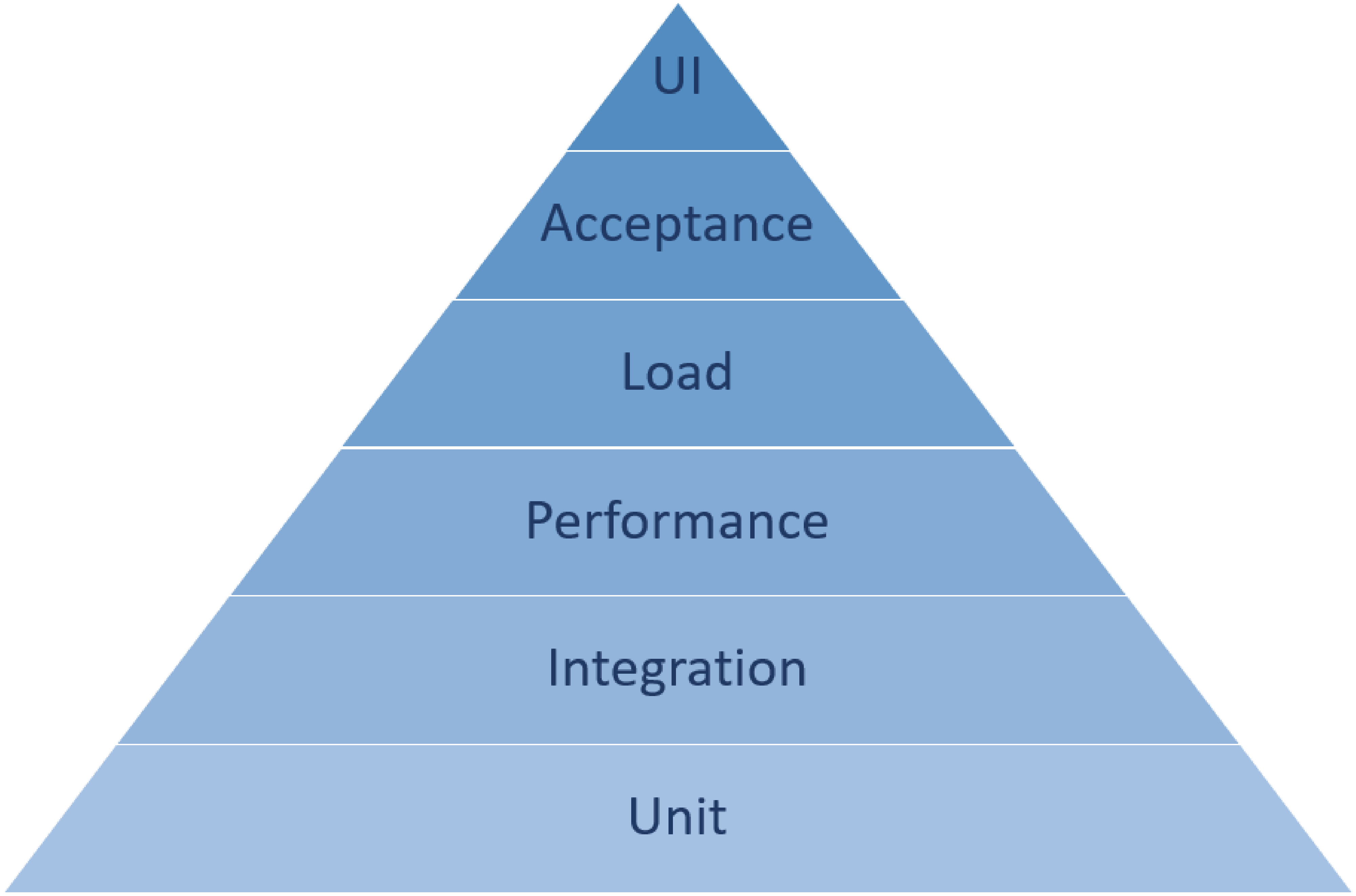
- A) Tests encourage a more modular architecture
- B) The code coverage tests provide is the ultimate indication of repo health.
- C) Tests help keep track of the different capabilities of a program.

The correct code coverage goal in a repo is:

- A) 100 percent
- B) 70 percent
- C) There's no one true answer. Your code coverage goal depends on the repository.

Characteristics of a good Unit Test ✓

- Fast. It is not uncommon for mature projects to have thousands of unit tests. Unit tests should take very little time to run. Milliseconds.
- Isolated. Unit tests are standalone, can be run in isolation, and have no dependencies on any outside factors such as a file system or database.
- Repeatable. Running a unit test should be consistent with its results, that is, it always returns the same result if you do not change anything in between runs.
- Self-Checking. The test should be able to automatically detect if it passed or failed without any human interaction.
- Timely. A unit test should not take a disproportionately long time to write compared to the code being tested. If you find testing the code taking a large amount of time compared to writing the code, consider a design that is more testable.


UI
Acceptance
Load
Performance
Integration
Unit

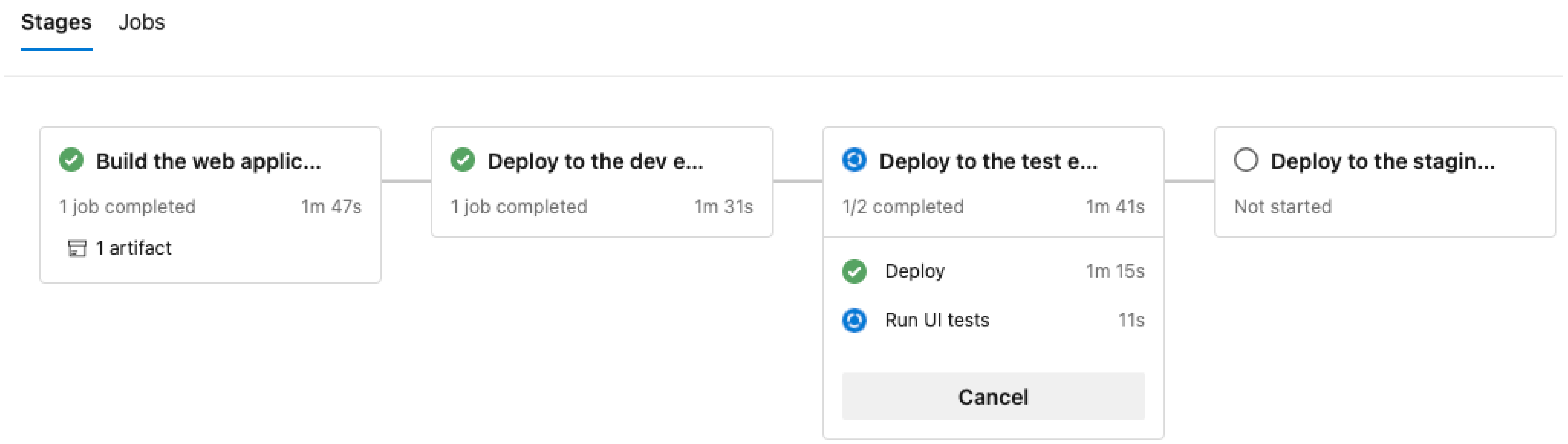
Types of Testing

Black Box	White Box	Gray Box
Input & Output System User Acceptance Performance Testing	Inside is visible Unit & Integration Testing	Combination of Black & White Box

Why do we need integration tests?

UI Tests

- Windows Application Driver
- Selenium
- SpecFlow
- Other



Stages Jobs

✓ Build the web applic...
1 job completed 1m 47s
1 artifact

✓ Deploy to the dev e...
1 job completed 1m 31s

Deploy to the test e...
1/2 completed 1m 41s
✓ Deploy 1m 15s
Run UI tests 11s
Cancel

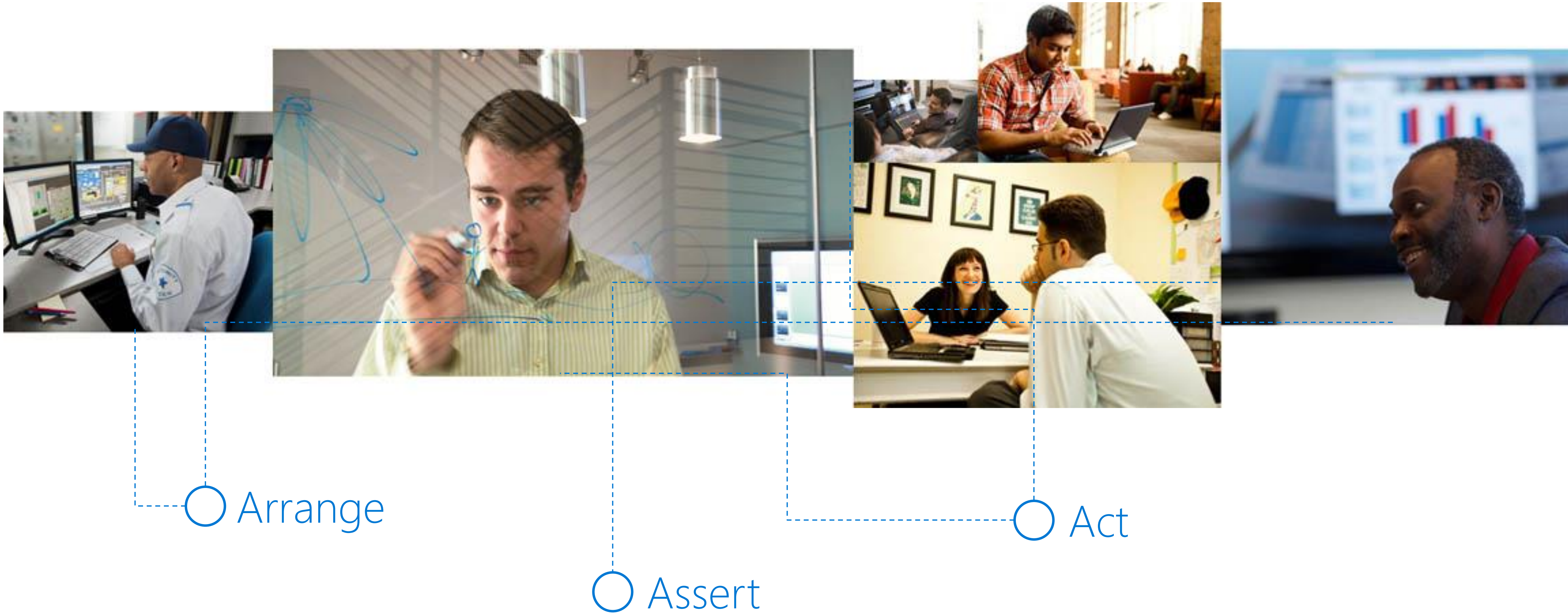
Deploy to the stagin...
Not started

What is Performance Testing?

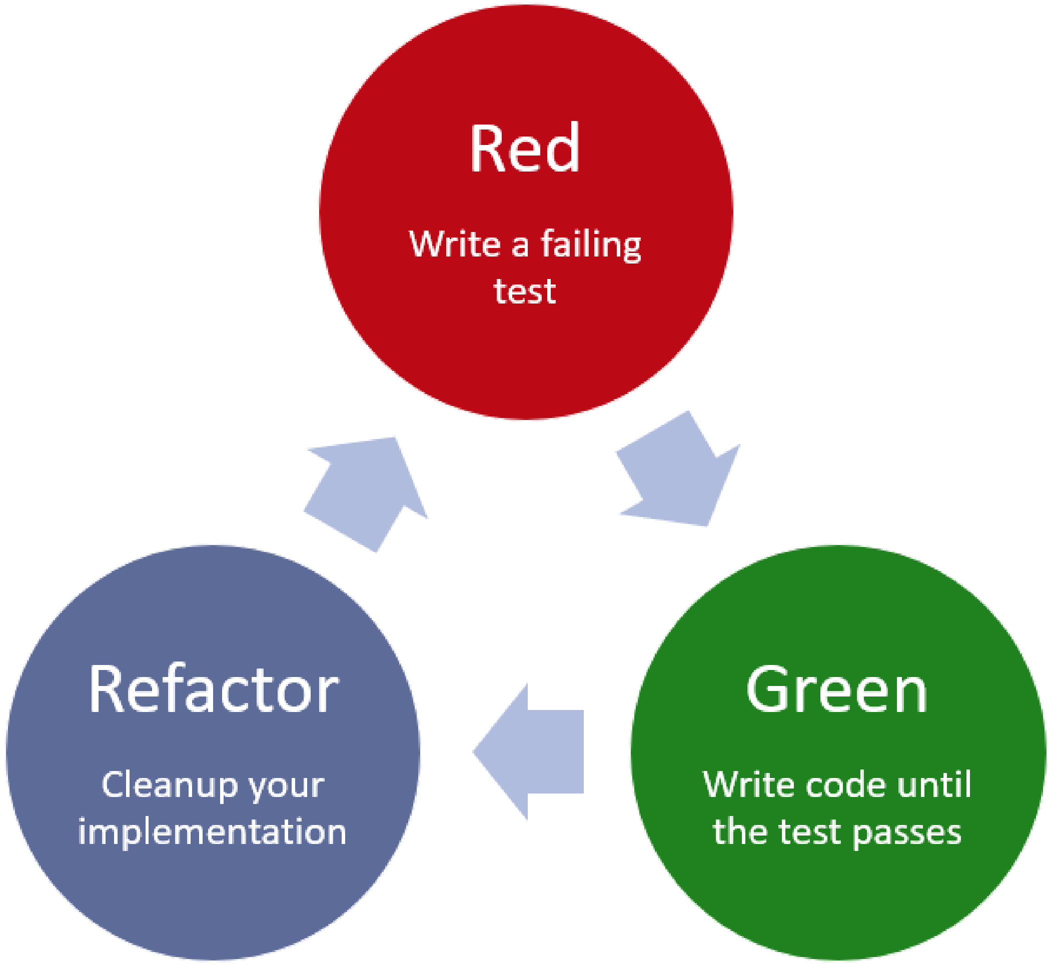
*"Assess performance **impact** of a given load for an application or resource."*

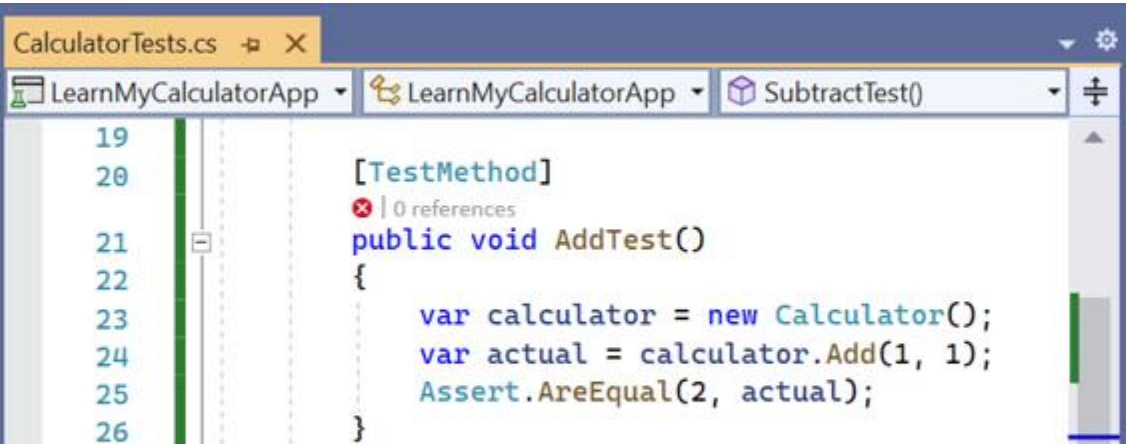
- When Performance Testing should happen?
- What needs to be measured?
- What are your performance requirements?

Testing School of Thought

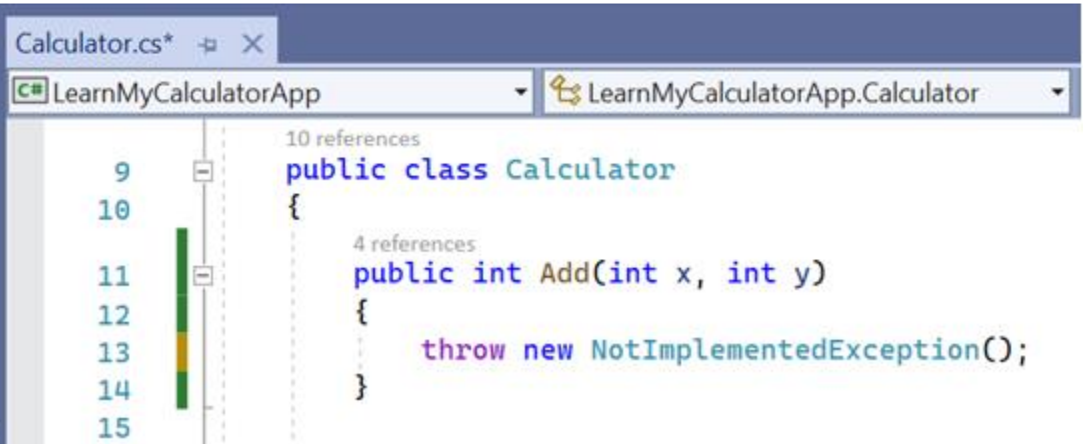


Testing schools of thought - TDD





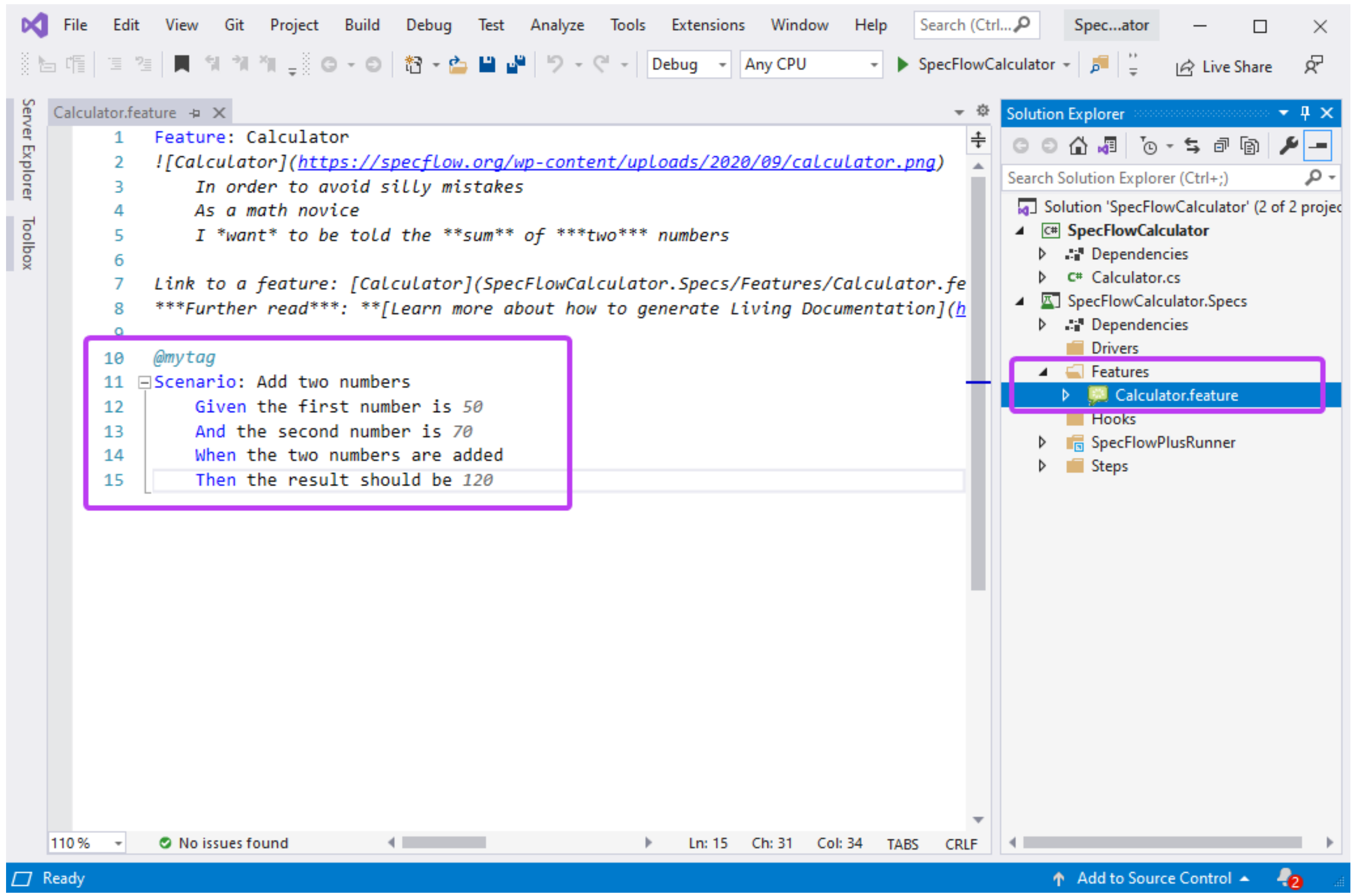
```
[TestMethod]
| 0 references
public void AddTest()
{
    var calculator = new Calculator();
    var actual = calculator.Add(1, 1);
    Assert.AreEqual(2, actual);
}
```



```
10 references
public class Calculator
{
    4 references
    public int Add(int x, int y)
    {
        throw new NotImplementedException();
    }
```

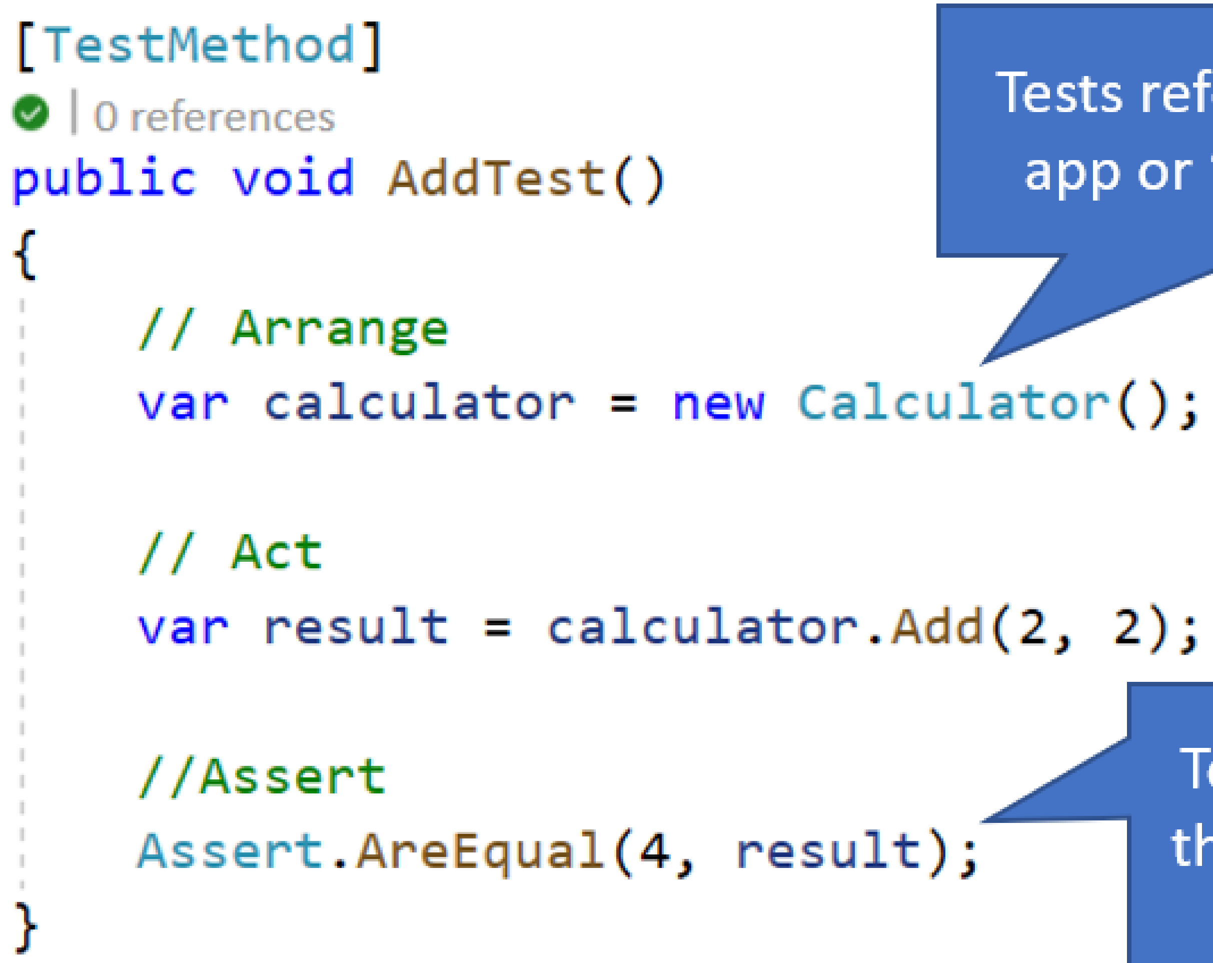
Testing schools of thought - BDD

```
[TestMethod]
[TestCategory("Behavior")]
0 references
public void AddTwoNumbersWithCalculatorTest()
{
    // When the calculator app is active for the user
    // User should be able to input two numbers in the calculator
    // No more than two numbers need to be accepted at this time
    // The user should be able to select `Add`
    // The calculator should then output the result of the two added
      numbers
    // This test should fail in any case where the addition is incorrect or
       the result does not match the expected output
}
```



```
Feature: Calculator
![Calculator](https://specflow.org/wp-content/uploads/2020/09/calculator.png)
    In order to avoid silly mistakes
    As a math novice
    I *want* to be told the **sum** of ***two*** numbers

Link to a feature: [Calculator](SpecFlowCalculator.Specs/Features/Calculator.fe
***Further read***: **[Learn more about how to generate Living Documentation](h

@mytag
Scenario: Add two numbers
    Given the first number is 50
    And the second number is 70
    When the two numbers are added
    Then the result should be 120
```

Arrange, Act, Assert



```
[TestMethod]
✓ | 0 references
public void AddTest()
{
    // Arrange
    var calculator = new Calculator();

    // Act
    var result = calculator.Add(2, 2);

    //Assert
    Assert.AreEqual(4, result);
}
```

Tests reference your real app or "product code"

Tests use asserts to compare the expected results with the actual output.

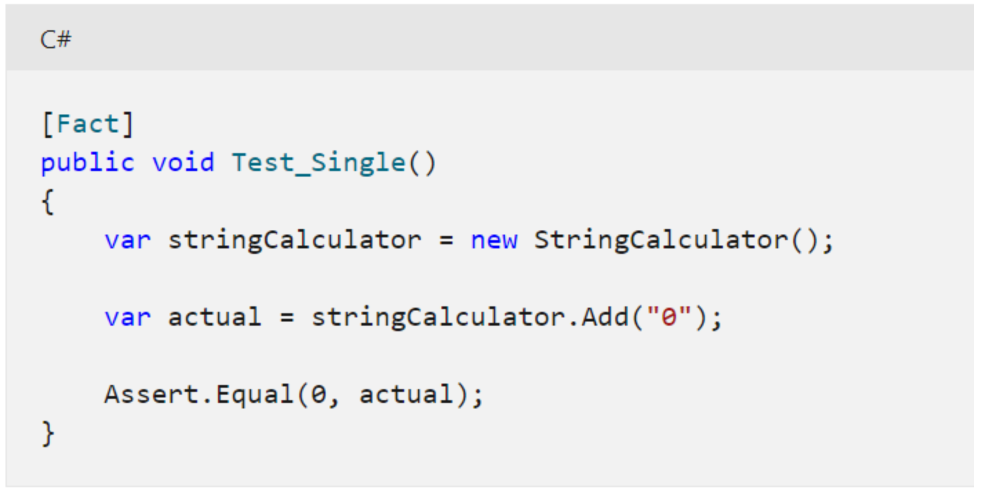
```csharp
[TestMethod]
public void AddTest()
{
    // Arrange
    var calculator = new Calculator();

    // Act
    var actual = calculator.Add(1, 1);
    var subtractActual = calculator.Subtract(actual, 1) == 1;

    // Assert
    Assert.IsNotNull(calculator);
    Assert.AreEqual(2, actual);
    Assert.IsTrue(subtractActual);
    StringAssert.Contains(actual.ToString(), "2");
}
```

Naming

The name of your test should consist of three parts:

- The name of the method being tested.
- The scenario under which it's being tested.
- The expected behavior when the scenario is invoked.

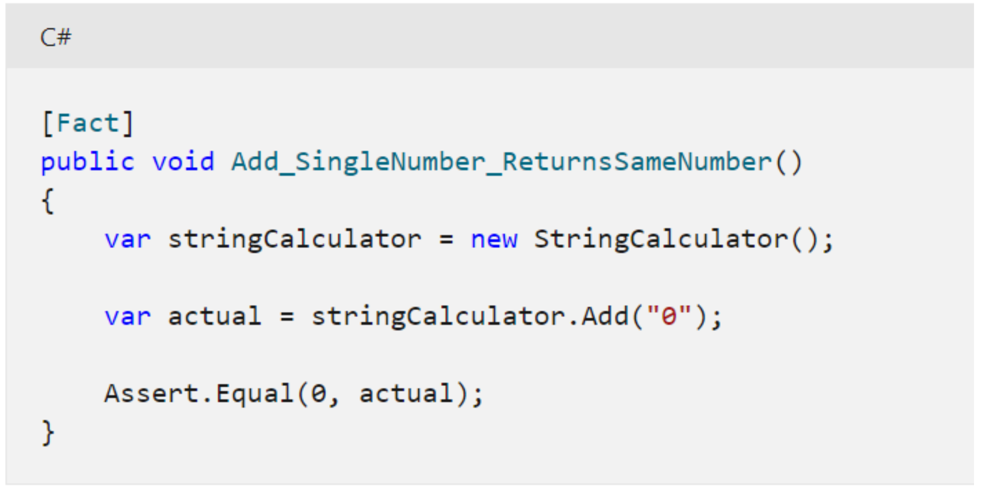
Bad:



C#

```
[Fact]
public void Test_Single()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("0");

    Assert.Equal(0, actual);
}
```

Better:



C#

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("0");

    Assert.Equal(0, actual);
}
```

Bad:

C#

```
[Fact]
public void Add_MultipleNumbers_ReturnsCorrectResults()
{
    var stringCalculator = new StringCalculator();
    var expected = 0;
    var testCases = new[]
    {
        "0,0,0",
        "0,1,2",
        "1,2,3"
    };

    foreach (var test in testCases)
    {
        Assert.Equal(expected, stringCalculator.Add(test));
        expected += 3;
    }
}
```

Better:

C#

```
[Theory]
[InlineData("0,0,0", 0)]
[InlineData("0,1,2", 3)]
[InlineData("1,2,3", 6)]
public void Add_MultipleNumbers_ReturnsSumOfNumbers(string input, int expected)
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add(input);

    Assert.Equal(expected, actual);
}
```

Bad:

C#

```
[Fact]
public void Test_Single()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("0");

    Assert.Equal(0, actual);
}
```

Better:

C#

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("0");

    Assert.Equal(0, actual);
}
```

Bad:

C#

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("42");

    Assert.Equal(42, actual);
}
```

Better:

C#

```
[Fact]
public void Add_SingleNumber_ReturnsSameNumber()
{
    var stringCalculator = new StringCalculator();

    var actual = stringCalculator.Add("0");

    Assert.Equal(0, actual);
}
```

Bad:

C#

```
[Fact]
public void Add_BigNumber_ThrowsException()
{
    var stringCalculator = new StringCalculator();

    Action actual = () => stringCalculator.Add("1001");

    Assert.Throws<OverflowException>(actual);
}
```

Better:

C#

```
[Fact]
void Add_MaximumSumResult_ThrowsOverflowException()
{
    var stringCalculator = new StringCalculator();
    const string MAXIMUM_RESULT = "1001";

    Action actual = () => stringCalculator.Add(MAXIMUM_RESULT);

    Assert.Throws<OverflowException>(actual);
}
```

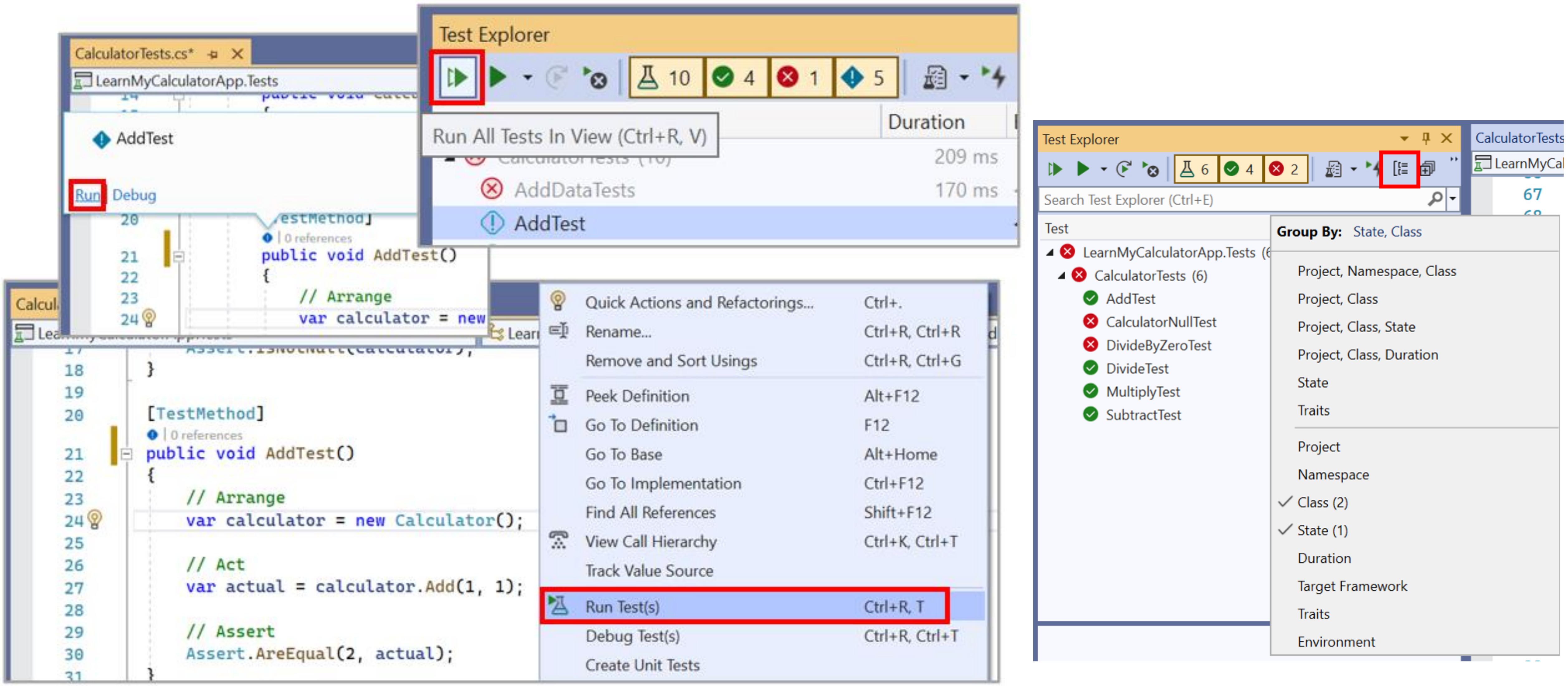
What is the most common type of test?

- A) Integration
- B) UI
- C) Unit

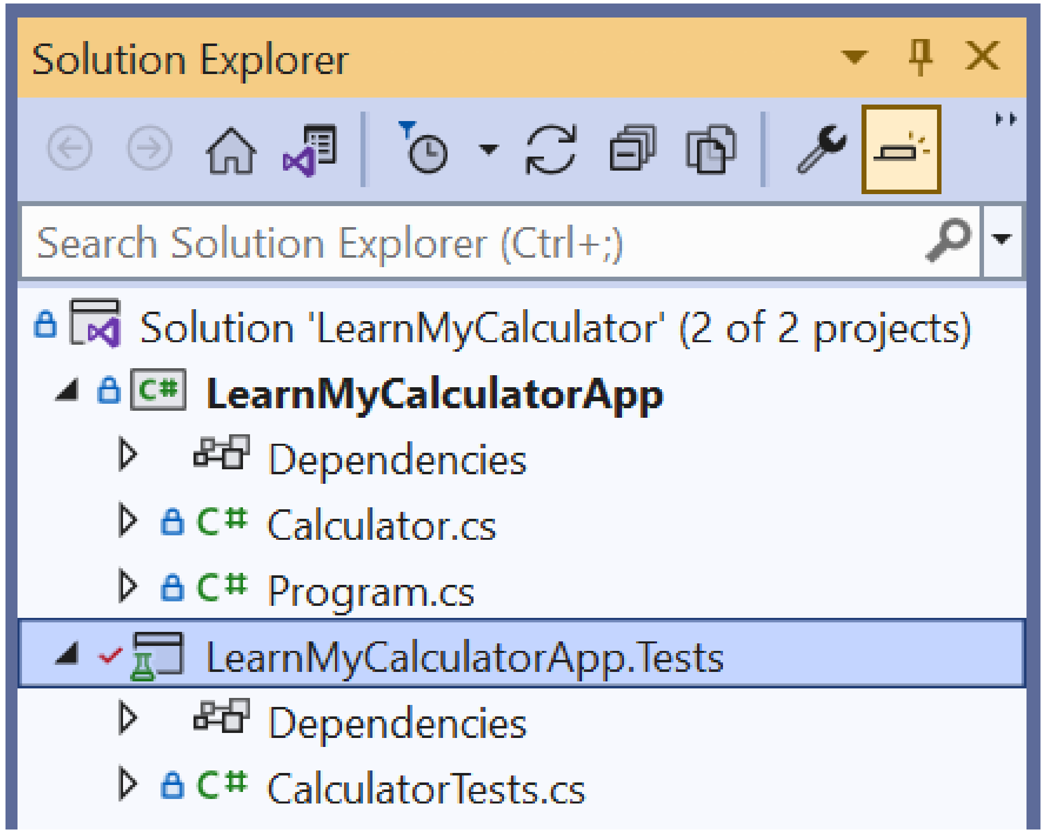
If I want to test how my app scales with multiple users using it, what type of test am I likely to write?

- A) Load
- B) Integration
- C) Performance

Unit testing C# with .NET in Visual Studio


```
    }

    [TestMethod]
    0 references
    public void AddTest()
    {
        // Arrange
        var calculator = new Calculator();

        // Act
        var actual = calculator.Add(1, 1);

        // Assert
        Assert.AreEqual(2, actual);
    }
```

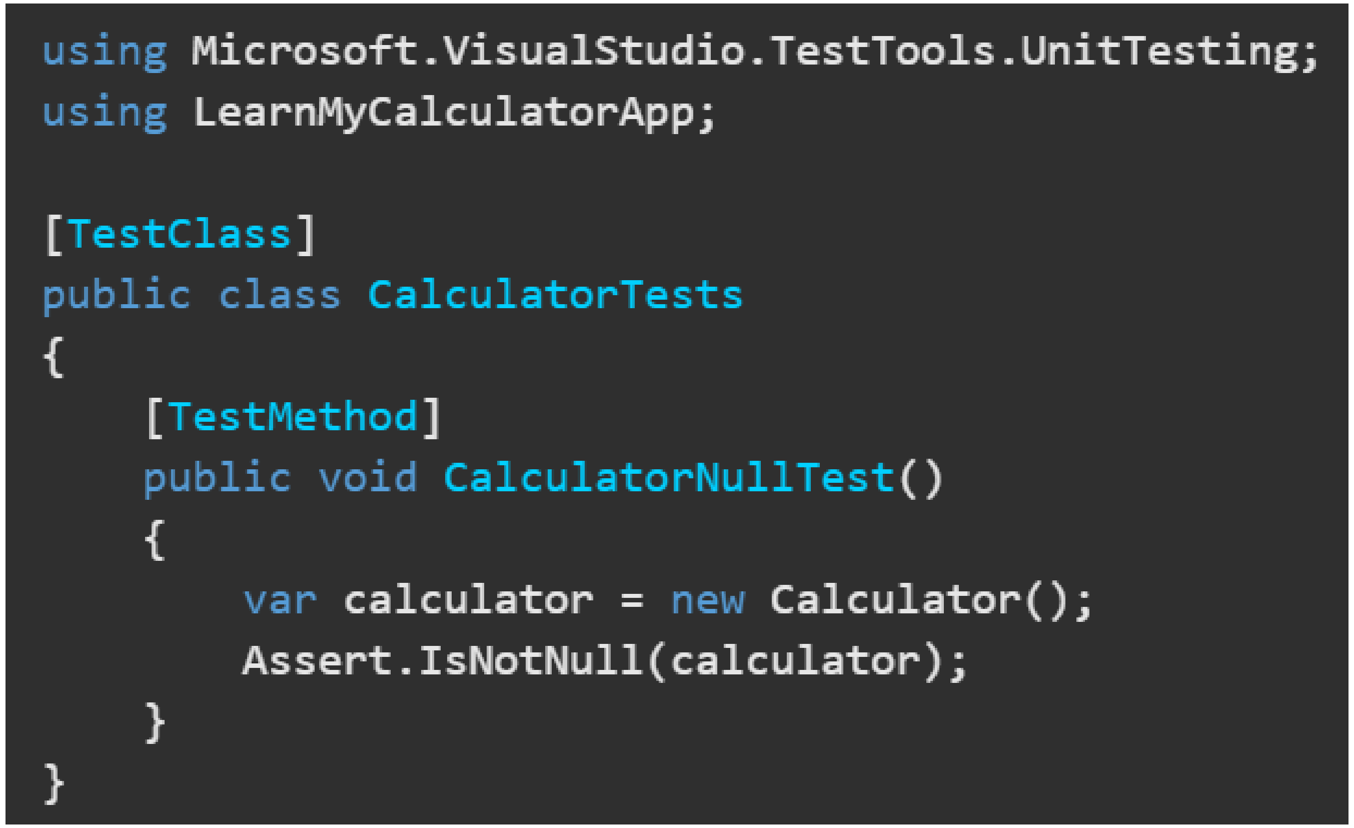
Microsoft.VisualStudio.TestTools.UnitTesting



Solution Explorer

Search Solution Explorer (Ctrl+;)

Solution 'LearnMyCalculator' (2 of 2 projects)
- LearnMyCalculatorApp
  - Dependencies
  - Calculator.cs
  - Program.cs
- LearnMyCalculatorApp.Tests
  - Dependencies
  - CalculatorTests.cs



```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using LearnMyCalculatorApp;

[TestClass]
public class CalculatorTests
{
    [TestMethod]
    public void CalculatorNullTest()
    {
        var calculator = new Calculator();
        Assert.IsNotNull(calculator);
    }
}
```

bUnit?

```
[Fact]
public void CounterShouldIncrementWhenClicked()
{
  // Arrange: render the Counter.razor component
  using var ctx = new TestContext();
  var cut = ctx.RenderComponent<Counter>();

  // Act: find and click the <button> element to increment
  // the counter in the <p> element
  cut.Find("button").Click();

  // Assert: first find the <p> element, then verify its content
  cut.Find("p").MarkupMatches("<p>Current count: 1</p>");
}
```

What do you need to add a reference from your test project to product code?

A) Add a reference to the target project

B) Both import the namespace and add a project reference

C) Import the namespace, add a project reference, and add the @test decorator on the test method

Which of the following causes a test to fail?

A) A failing assert statement is the only reason a test would fail

B) A test fails when *most* assertions in the test are failing

C) Tests can fail for various reasons, including at least one failing assertion, an uncaught exception, or test time-out.

- Dummies
- Stubs
- Spies
- Mocks
- True Mocks

Mocking Frameworks

```
 // Moq
mockWeatherService.Setup(x => x.GetWeatherForCity(It.IsAny<string>())).Returns(new WeatherForecast());

// NSubstitute
mockWeatherService.GetWeatherForCityk(Arg.Any<string>()).Returns(new WeatherForecast());

// FakeItEasy
A.CallTo(() => mockWeatherService.GetWeatherForCity(A<string>.Ignored))).Returns(new WeatherForecast());
```

Basics

- Stubs & Mocks
- Expect
- Register
- Argument filter
- Return
- Throw
- Execute

What improves testability ?

Watch out for problematic patterns

- **Static methods**
- **Singletons**
- **Sealed classes with no interface**
- **Concrete class with no interface/abstract class**

Favor composition over inheritance, interfaces, DI (IoC)

Use factory method/abstract factory, builder/director patters, SOLID, DRY

Code Coverage

- Legacy code = any code without test
- Two ways of changing code (Michael Feathers)

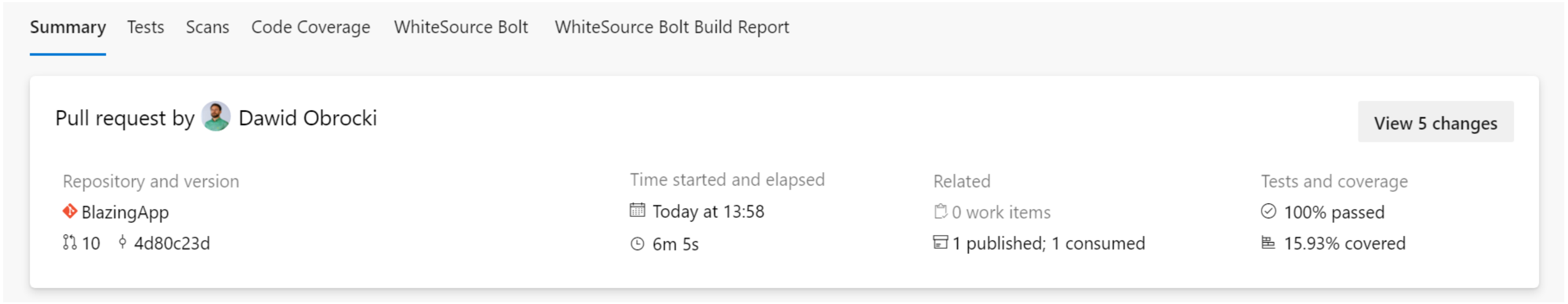
  Edit and pray

  Cover and modify
- Fear of regression

  Lack of confidence in code

  Prevention of problems by making minimal code change doesn’t work

Unit Tests in Azure DevOps – Build Pipeline

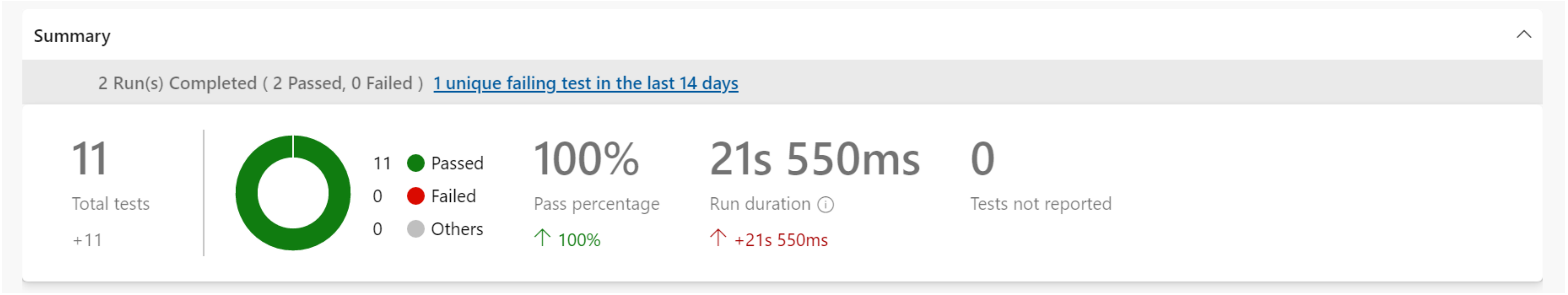


Summary Tests Scans Code Coverage WhiteSource Bolt WhiteSource Bolt Build Report

Pull request by Dawid Obrocki

View 5 changes

Repository and version	Time started and elapsed	Related	Tests and coverage
BlazingApp	Today at 13:58	0 work items	100% passed
10 4d80c23d	6m 5s	1 published; 1 consumed	15.93% covered



Summary

2 Run(s) Completed (2 Passed, 0 Failed) 1 unique failing test in the last 14 days

11
Total tests
+11

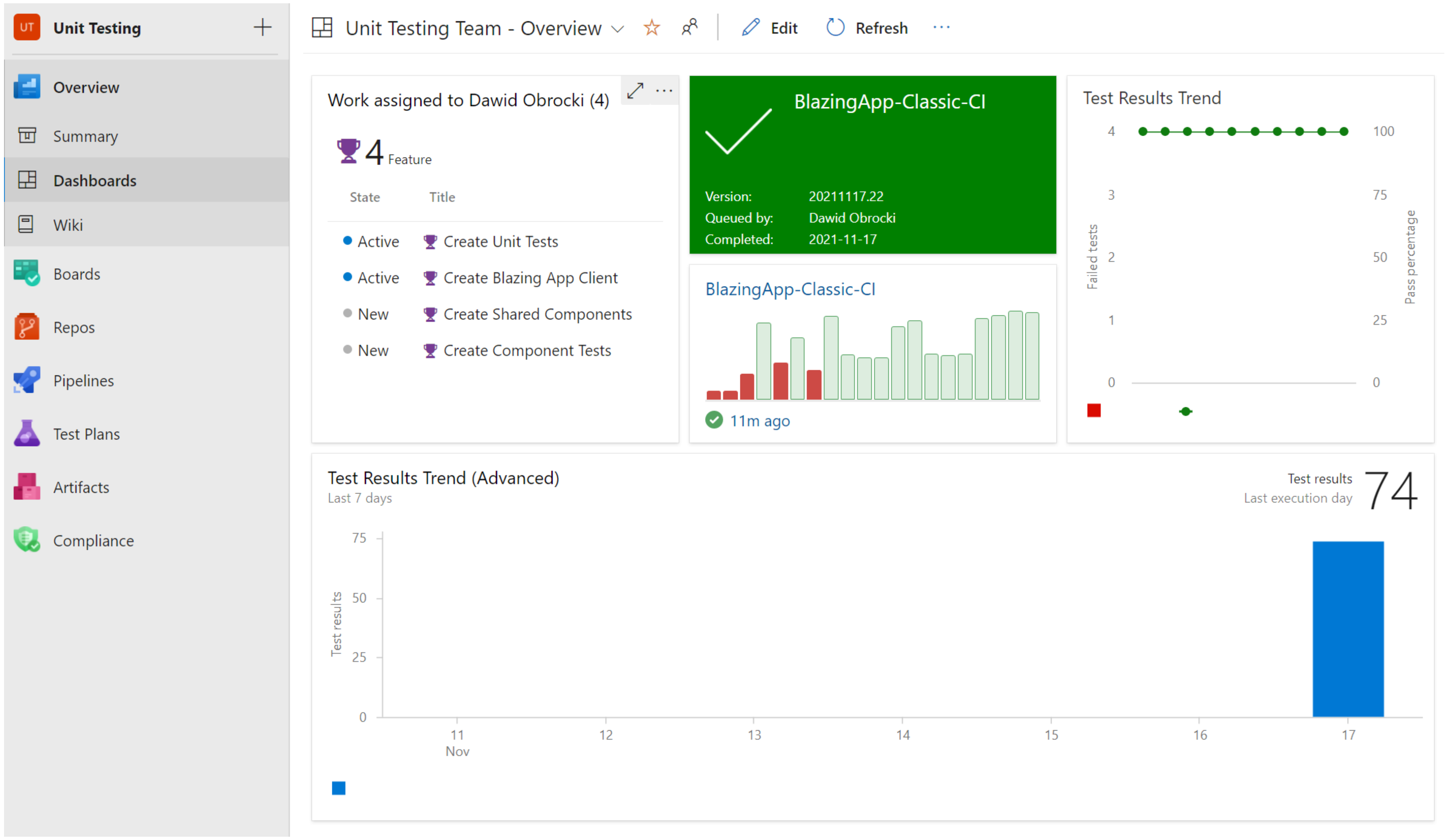
11	🟢	Passed
0	🔴	Failed
0	⚪	Others

100%
Pass percentage
↑ 100%

21s 550ms
Run duration
↑ +21s 550ms

0
Tests not reported

Unit Tests in Azure DevOps - Dashboards



Work assigned to Dawid Obrocki (4)

4 Feature

State	Title
Active	Create Unit Tests
Active	Create Blazing App Client
New	Create Shared Components
New	Create Component Tests

BlazingApp-Classic-CI

Version: 20211117.22
Queued by: Dawid Obrocki
Completed: 2021-11-17

BlazingApp-Classic-CI

11m ago

Test Results Trend

Test Results Trend (Advanced)
Last 7 days

Test results
Last execution day
74

How can you run tests in Visual Studio?

A) You can run tests only from the right-click menu in Visual Studio.

B) You can run tests only from Test Explorer in Visual Studio.

C) You can run tests by using the right-click menu, keyboard shortcuts, or CodeLens icons.

How does the Group By setting in Test Explorer allow you to view test groupings?

A) By project

B) By class

C) By project, by namespace, and then by class

D) All of the above, in any order

DEMO

Testing for Accessibility

Defect Identification - Testing Tools

Automated Testing

- Accessibility Testing Extension for Azure DevOps
- Pa11y and Deque Labs
- Chromium developer tools (Edge, Chrome)
- Visual Studio Web Accessibility checker
- Accessibility Insights & WAVE
- Color Contrast Analyzer
- Mobile Testing tools (Android / iOS)

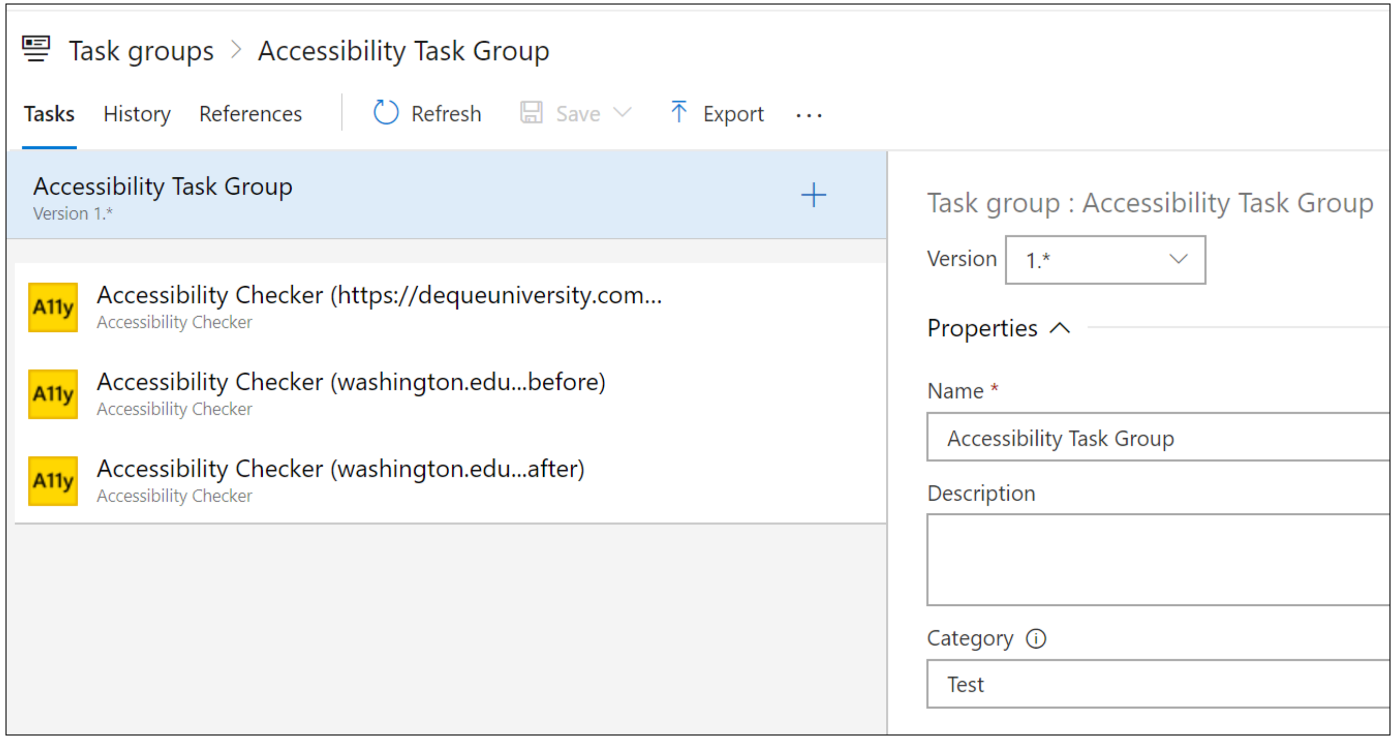
Assistive Technology Based

- Keyboard navigation
- Screen Reader
  - Windows Narrator
  - JAWS
  - NVDA

Accessibility Testing Extension for Azure DevOps

Automated Testing Tools

Accessibility Testing Extension helps integrate Accessibility Testing into your Azure DevOps **Release Pipelines**. Fully customizable and supports all major international accessibility standards.



Accessibility Task Group
Version 1.*

A11y Accessibility Checker (https://dequeuniversity.com...
Accessibility Checker

A11y Accessibility Checker (washington.edu...before)
Accessibility Checker

A11y Accessibility Checker (washington.edu...after)
Accessibility Checker

Task group : Accessibility Task Group

Version: 1.*

Properties

Name *: Accessibility Task Group

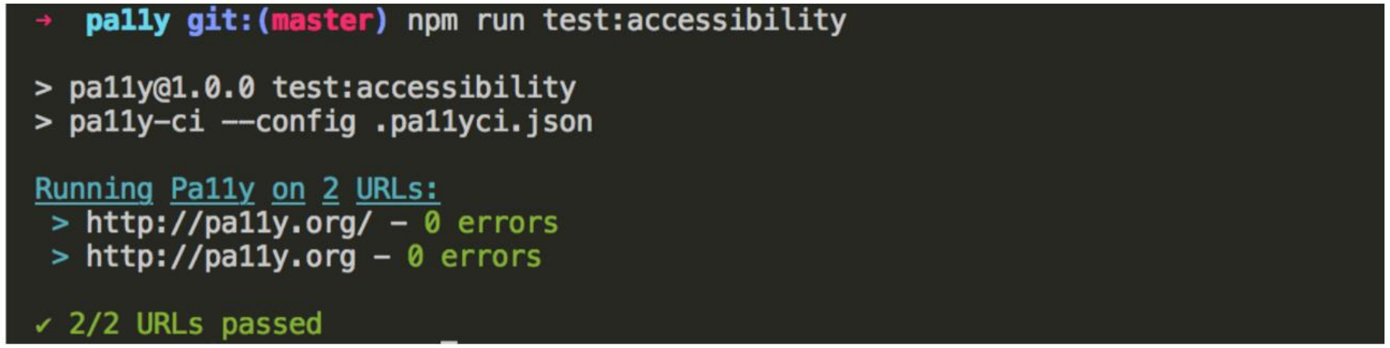
Description:

Category: Test

Pa11y & Deque Labs

Automated Testing Tools

Pa11y works by command line, web service, web dashboard, or console application. It integrates with CI tools like Jenkins or Travis.



```
→  pa11y git:(master) npm run test:accessibility

> pa11y@1.0.0 test:accessibility
> pa11y-ci --config .pa11yci.json

Running Pa11y on 2 URLs:
 > http://pa11y.org/ - 0 errors
 > http://pa11y.org - 0 errors

✓ 2/2 URLs passed
```

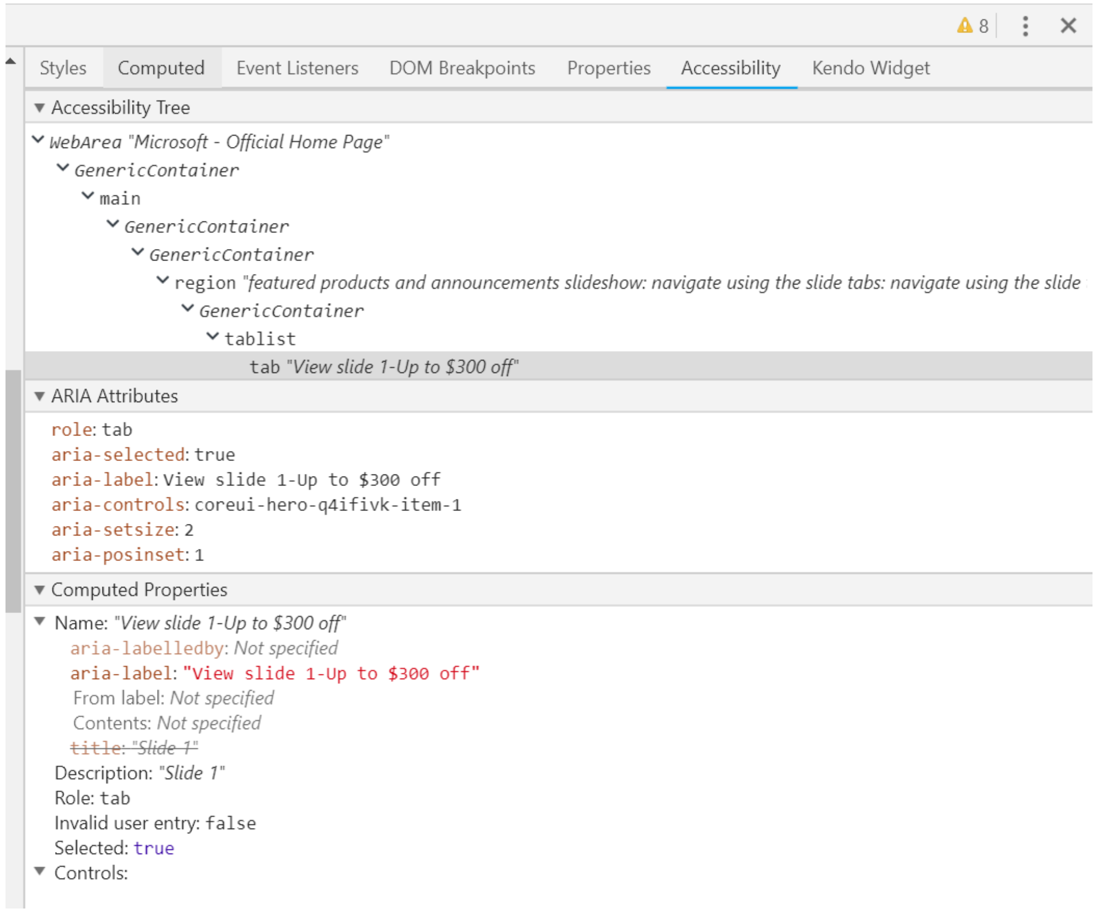
Deque Labs
axe-core: A library for automated Web UI testing
axe-webdriverjs: Provides a chainable aXe API for Selenium's WebDriverJS and automatically injects into all frames
axe-coconut: a devtool for chrome
axe-firefox-devtools: a devtool for Firefox

Chromium Developer Tools

Automated Testing Tools

Accessibility Tab shows all the properties that relate to accessibility on the selected element.

Available on Microsoft Edge and Chrome.

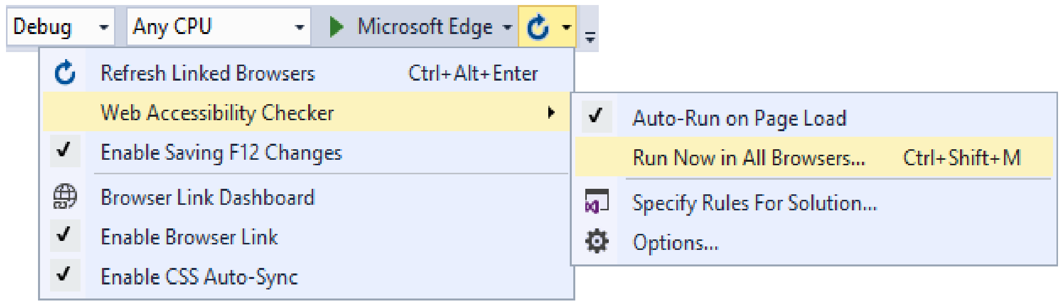


```
▼ Accessibility Tree
⌄ WebArea "Microsoft - Official Home Page"
  ⌄ GenericContainer
    ⌄ main
      ⌄ GenericContainer
        ⌄ GenericContainer
          ⌄ region "featured products and announcements slideshow: navigate using the slide tabs: navigate using the slide
            ⌄ GenericContainer
              ⌄ tablist
                  tab "View slide 1-Up to $300 off"
▼ ARIA Attributes
  role: tab
  aria-selected: true
  aria-label: View slide 1-Up to $300 off
  aria-controls: coreui-hero-q4ifivk-item-1
  aria-setsize: 2
  aria-posinset: 1
▼ Computed Properties
▼ Name: "View slide 1-Up to $300 off"
    aria-labelledby: Not specified
    aria-label: "View slide 1-Up to $300 off"
    From label: Not specified
    Contents: Not specified
    ~~title: "Slide 1"~~
  Description: "Slide 1"
  Role: tab
  Invalid user entry: false
  Selected: true
▼ Controls:
```

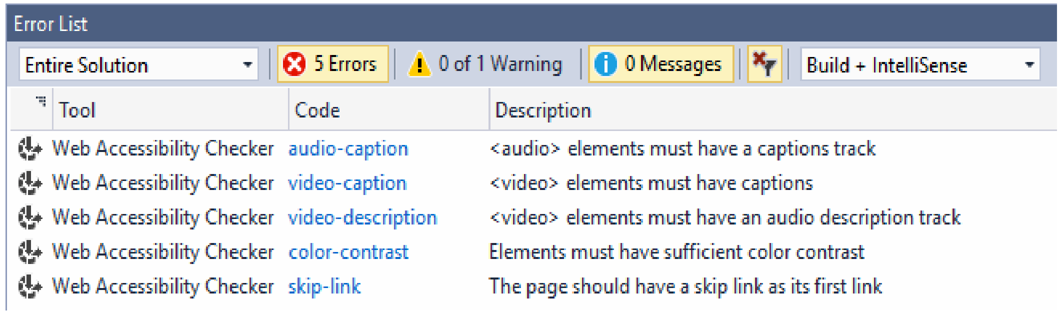
Visual Studio Web Accessibility Checker

Automated Testing Tools

Run it from the Browser Link


Debug
Any CPU
Microsoft Edge
Refresh Linked Browsers Ctrl+Alt+Enter
Web Accessibility Checker
✓ Enable Saving F12 Changes
Browser Link Dashboard
✓ Enable Browser Link
✓ Enable CSS Auto-Sync
✓ Auto-Run on Page Load
Run Now in All Browsers... Ctrl+Shift+M
Specify Rules For Solution...
Options...

Integrates with Visual Studio error list



Error List

Entire Solution | 5 Errors | 0 of 1 Warning | 0 Messages | Build + IntelliSense

Tool	Code	Description
Web Accessibility Checker	audio-caption	<audio> elements must have a captions track
Web Accessibility Checker	video-caption	<video> elements must have captions
Web Accessibility Checker	video-description	<video> elements must have an audio description track
Web Accessibility Checker	color-contrast	Elements must have sufficient color contrast
Web Accessibility Checker	skip-link	The page should have a skip link as its first link

- For Visual Studio 2015, 2017, 2019
- For .NET Applications or static web sites
- Tests WCAG Level A, AA, and Section 508.

WebAIM WAVE Tool (Chrome, Firefox)

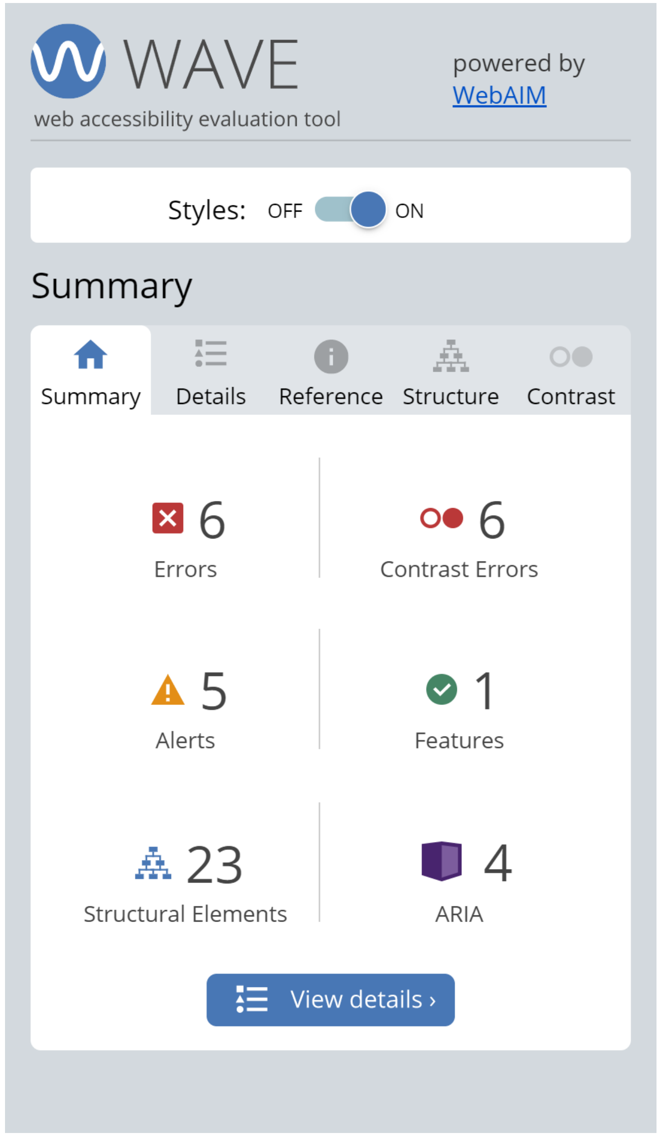
Automated Testing Tools

WebAIM WAVE Extension helps you find:

- WCAG A/AA/AAA Errors
- ARIA Tags
- Section 508 Errors
- Color Contrast Problems

The absence of errors DOES NOT mean the page is accessible.

Only humans can determine what is accessible.


WAVE
web accessibility evaluation tool
powered by WebAIM

Styles: OFF ON

Summary

Summary Details Reference Structure Contrast

6 Errors
6 Contrast Errors
5 Alerts
1 Features
23 Structural Elements
4 ARIA

View details ›

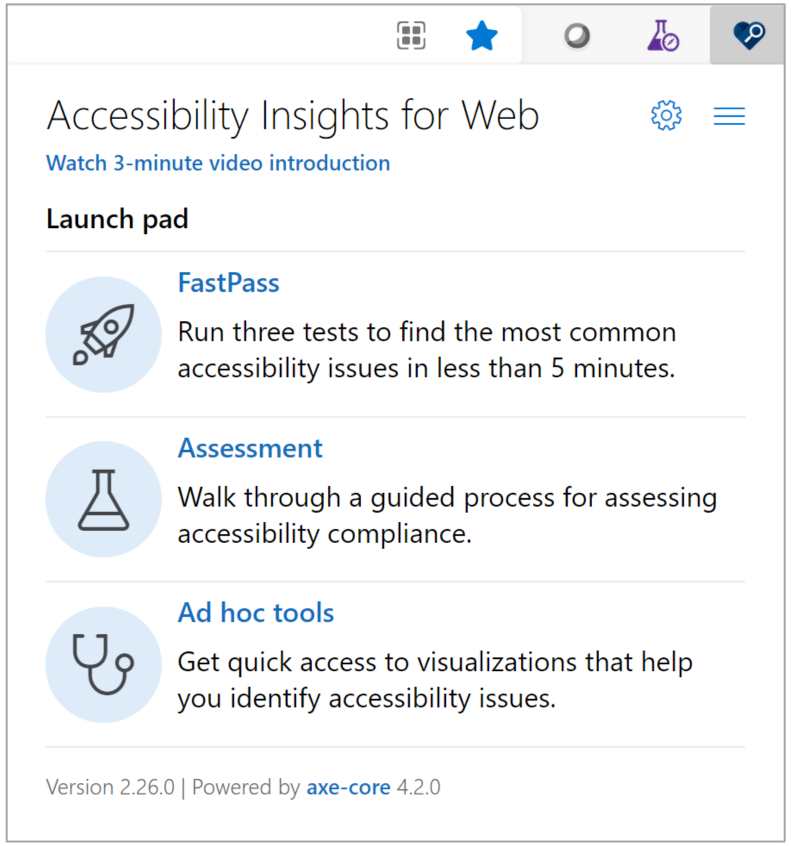
Accessibility Insights

Fast Pass

Automated Testing Tools

Run tests to find the most common accessibility issues in **less than 5 minutes**.

- Automated checks will put the target page through accessibility spec rules.
- **Tab stops** provide a way to visualize tab order on the page.



Accessibility Insights for Web

Watch 3-minute video introduction

Launch pad

FastPass

Run three tests to find the most common accessibility issues in less than 5 minutes.

Assessment

Walk through a guided process for assessing accessibility compliance.

Ad hoc tools

Get quick access to visualizations that help you identify accessibility issues.

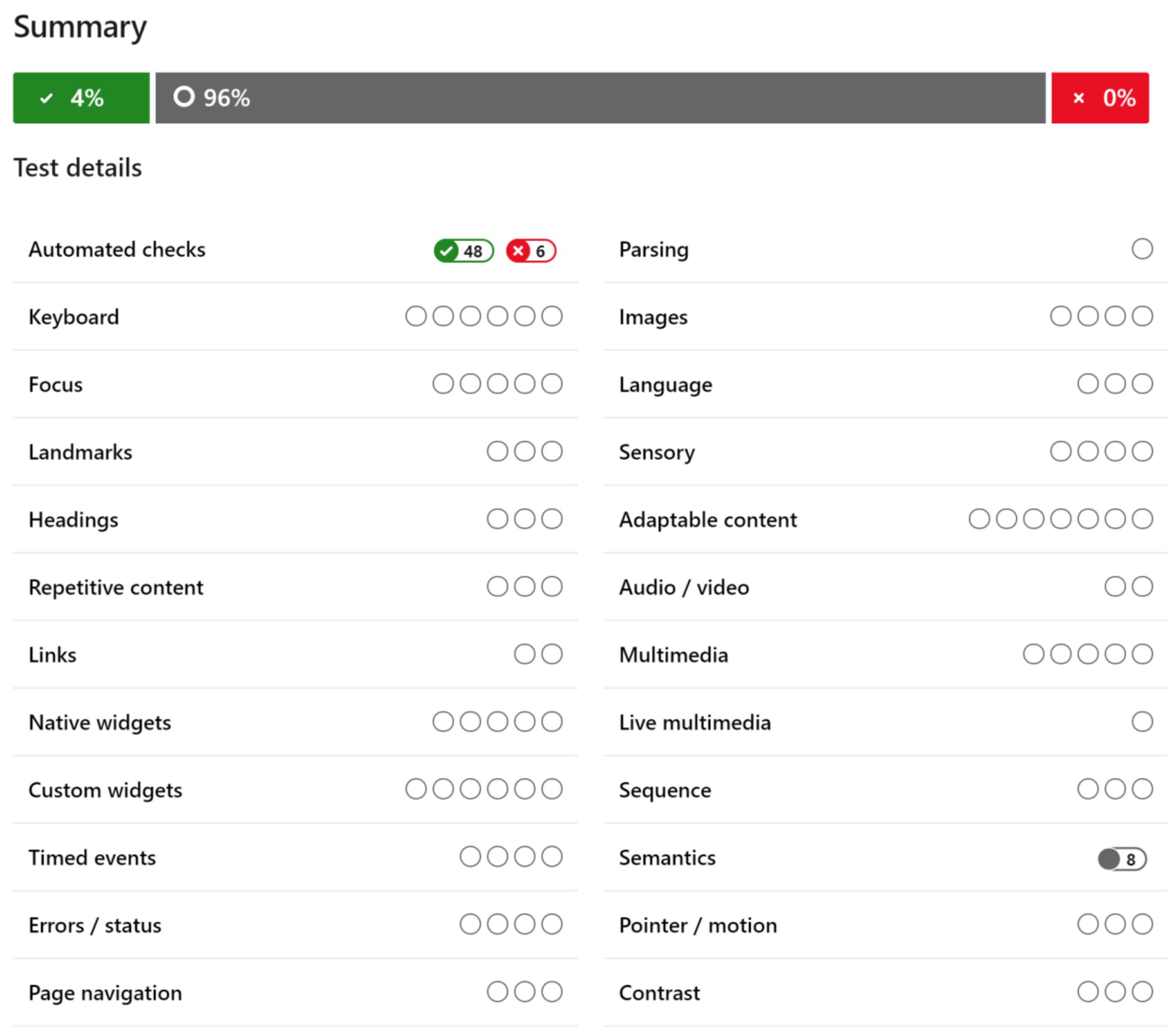
Version 2.26.0 | Powered by axe-core 4.2.0

Accessibility Insights

Assessment

Automated Testing Tools

A guided process for assessing accessibility compliance at Level AA for WCAG 2.1.



Summary

✓ 4% | ○ 96% | × 0%

Test details

Automated checks	✓ 48 ✗ 6	Parsing	○
Keyboard	○○○○○○	Images	○○○○
Focus	○○○○○	Language	○○○
Landmarks	○○○	Sensory	○○○○
Headings	○○○	Adaptable content	○○○○○○○
Repetitive content	○○○	Audio / video	○○
Links	○○	Multimedia	○○○○○
Native widgets	○○○○○	Live multimedia	○
Custom widgets	○○○○○○	Sequence	○○○
Timed events	○○○○	Semantics	● 8
Errors / status	○○○○	Pointer / motion	○○○
Page navigation	○○○	Contrast	○○○

Windows Narrator

Assistive Technology Based Tools

- Press Windows logo key + Ctrl + Enter to start or stop Narrator.
- Ctrl key to Silence Narrator.
- Change speech rate CapsLock + Plus(+) and CapsLock + Minus(-)
- Common navigation TAB and cursor/arrow keys.
- Narrator settings Windows logo key + Ctrl + N
- Change Narrator "views": CapsLock + Up arrow or CapsLock + Down arrow
- Move by item: CapsLock + Left arrow or CapsLock + Right arrow
- Change verbosity: CapsLock + A (cycles through 6 levels)

Microsoft

Azure

Azure DevTest Labs

Why Dev & Test?

Getting Dev/Test Environments

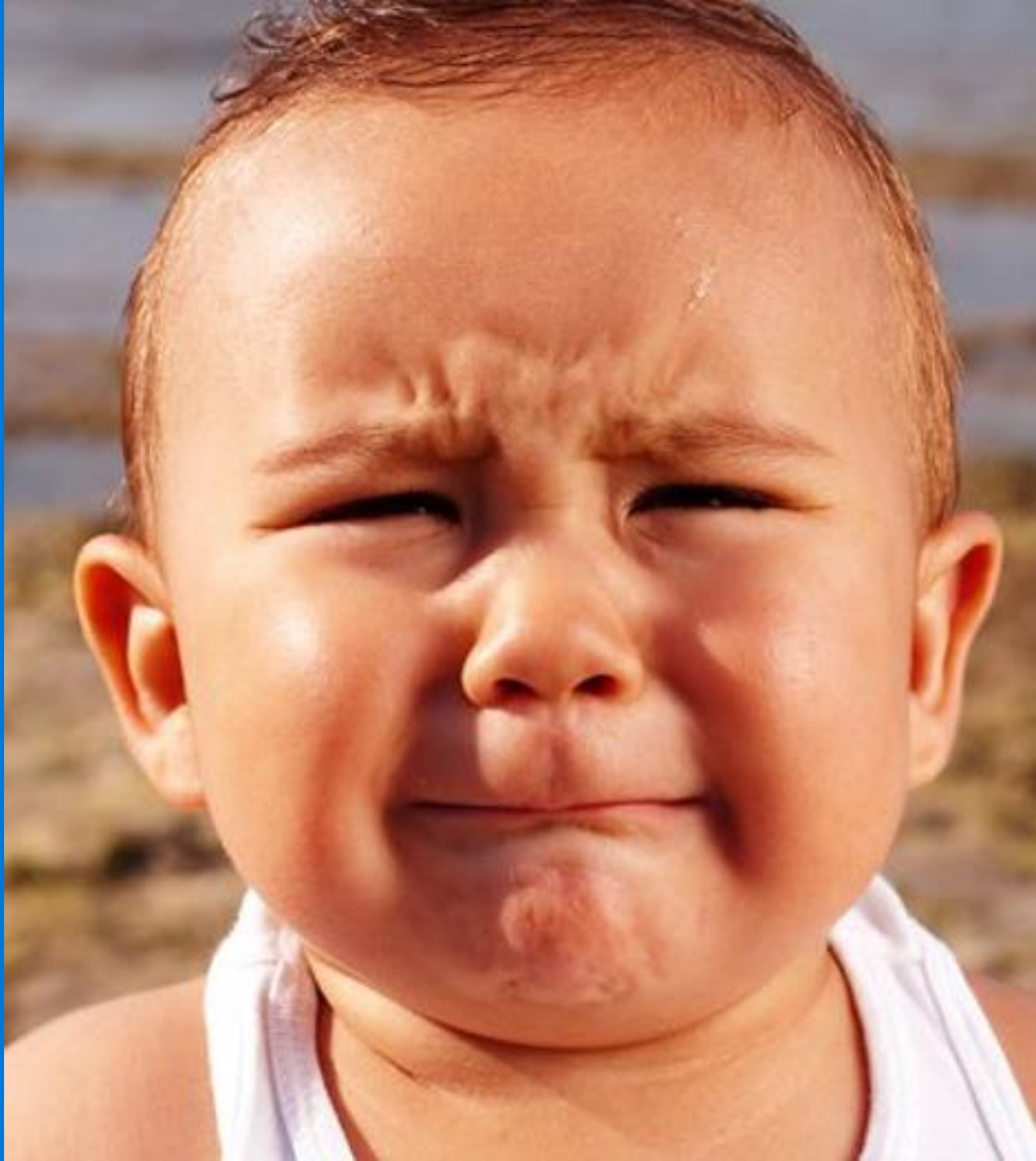
SUCKS!!

Long Infrastructure **Wait Time**

Time-consuming Configurations

Cost Control Issues

Production **fidelity** Issues



65% of developers say it is too complicated and time-consuming to get Dev/Test resources

10% Average utilization of dedicated Dev/Test infrastructure

Source: Business Case for Test Environment Management Whitepaper, Cognizant

DevTest investments are significant

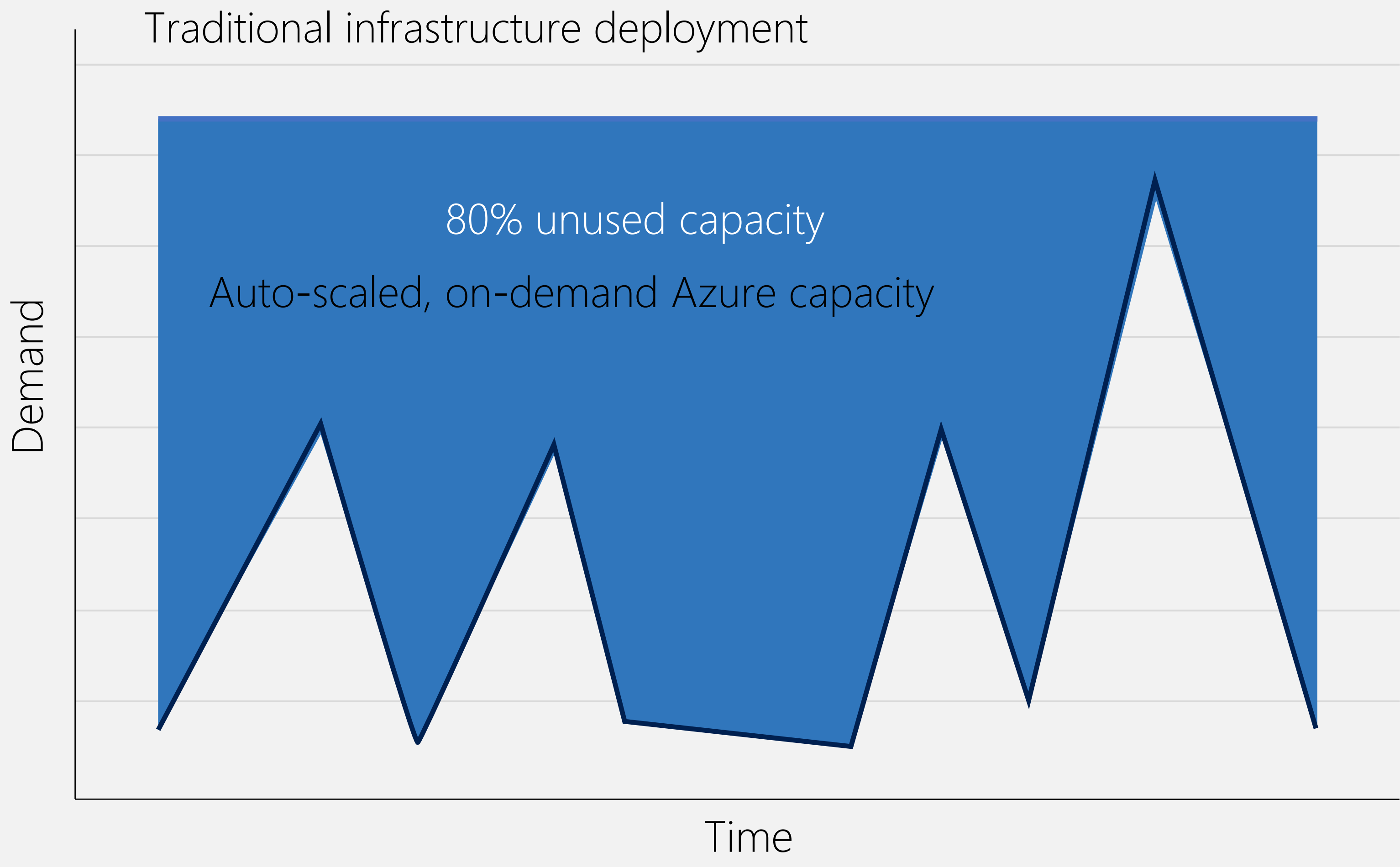
50%

of infrastructure spent on non-production

The new DevTest opportunity

50%

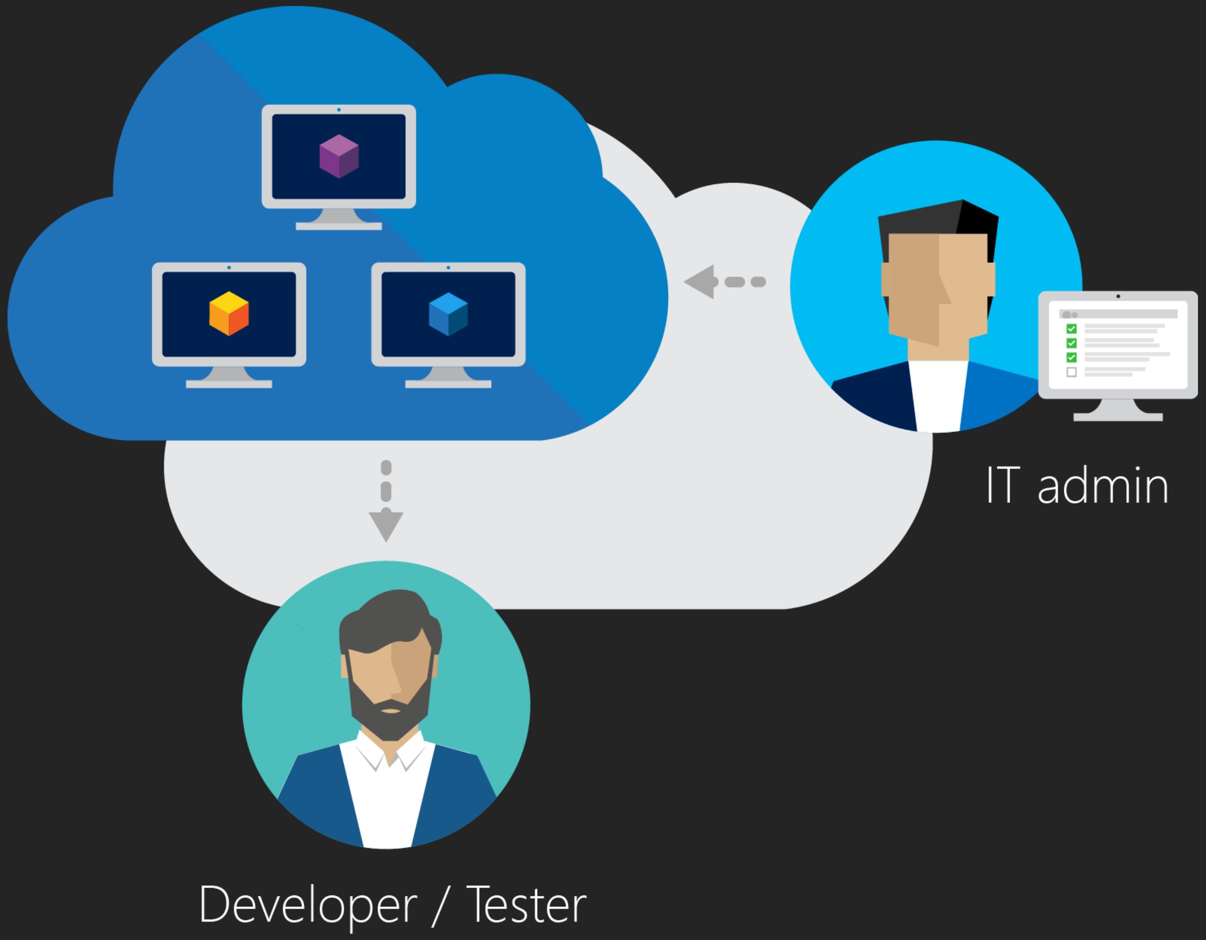
of infrastructure spent on non-production



Azure DevTest Labs

Solution for fast, easy, and agile dev-test environments in Azure.

- Fast provisioning
- Automation & self-service
- Cost control and governance



Worry-free **self-service**

Faster provisioning

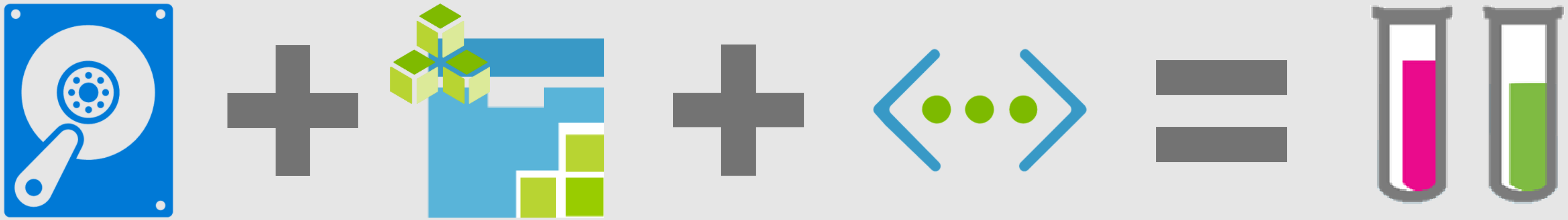
Create Once, **Re-Use** Everywhere, By Everyone

Sandboxed environment

Integrates with your **Existing Toolchain**

Provisioning machines

IT Admin



Base Image

VHD / VHDX file

Artifacts

Other applications

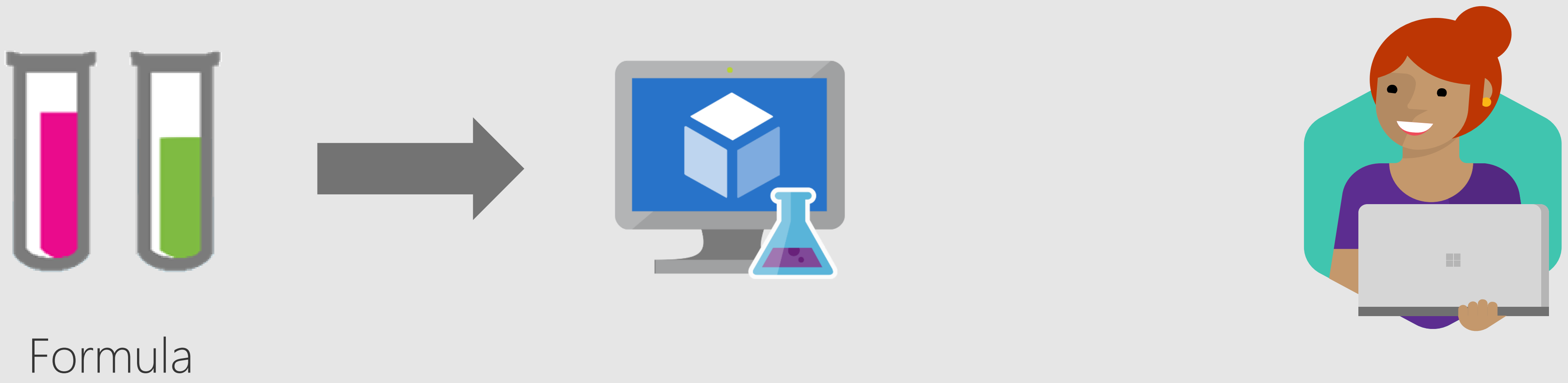
Settings

VM size,
VNets, subnets

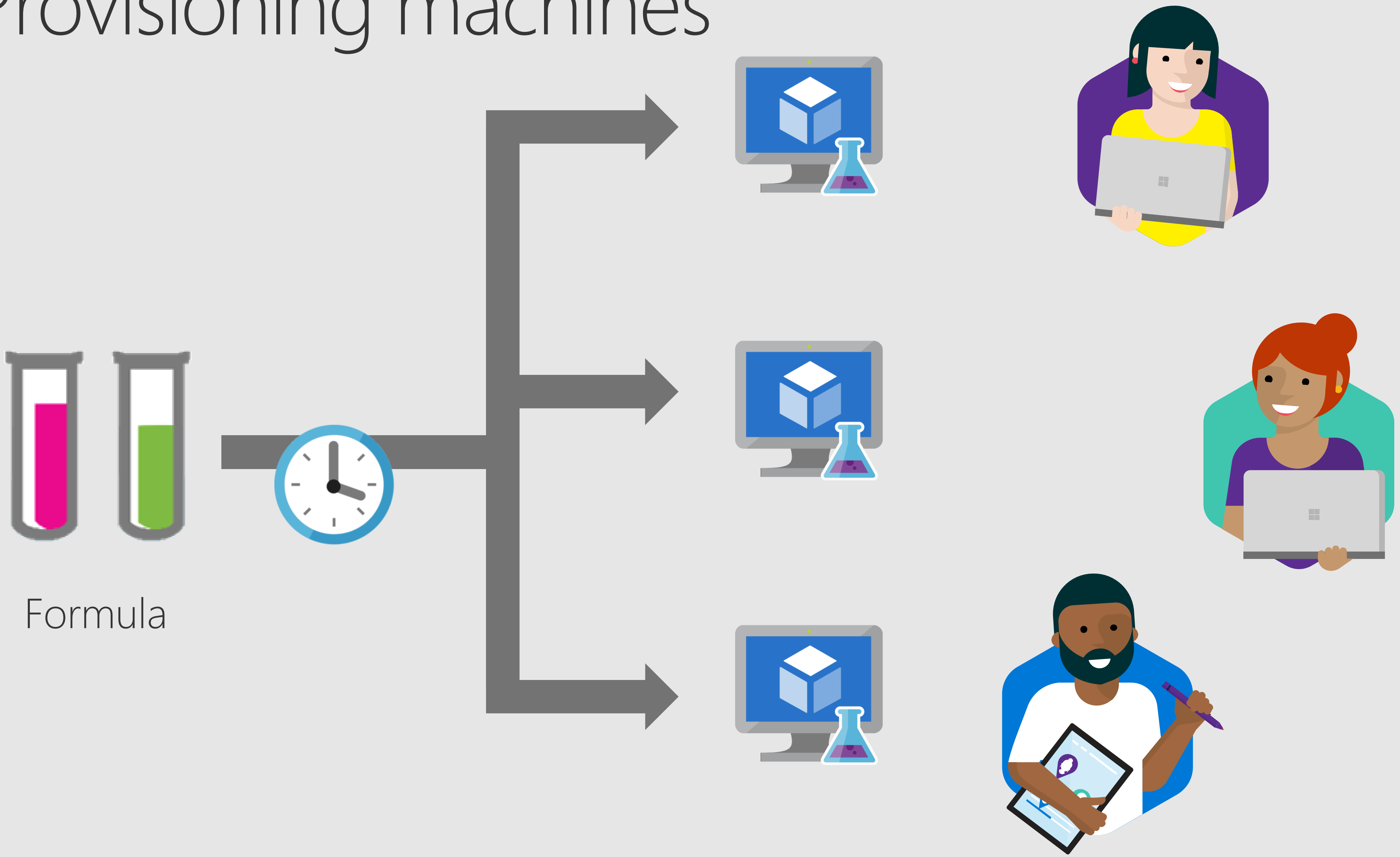
Formula

Template /
VM description

Provisioning machines

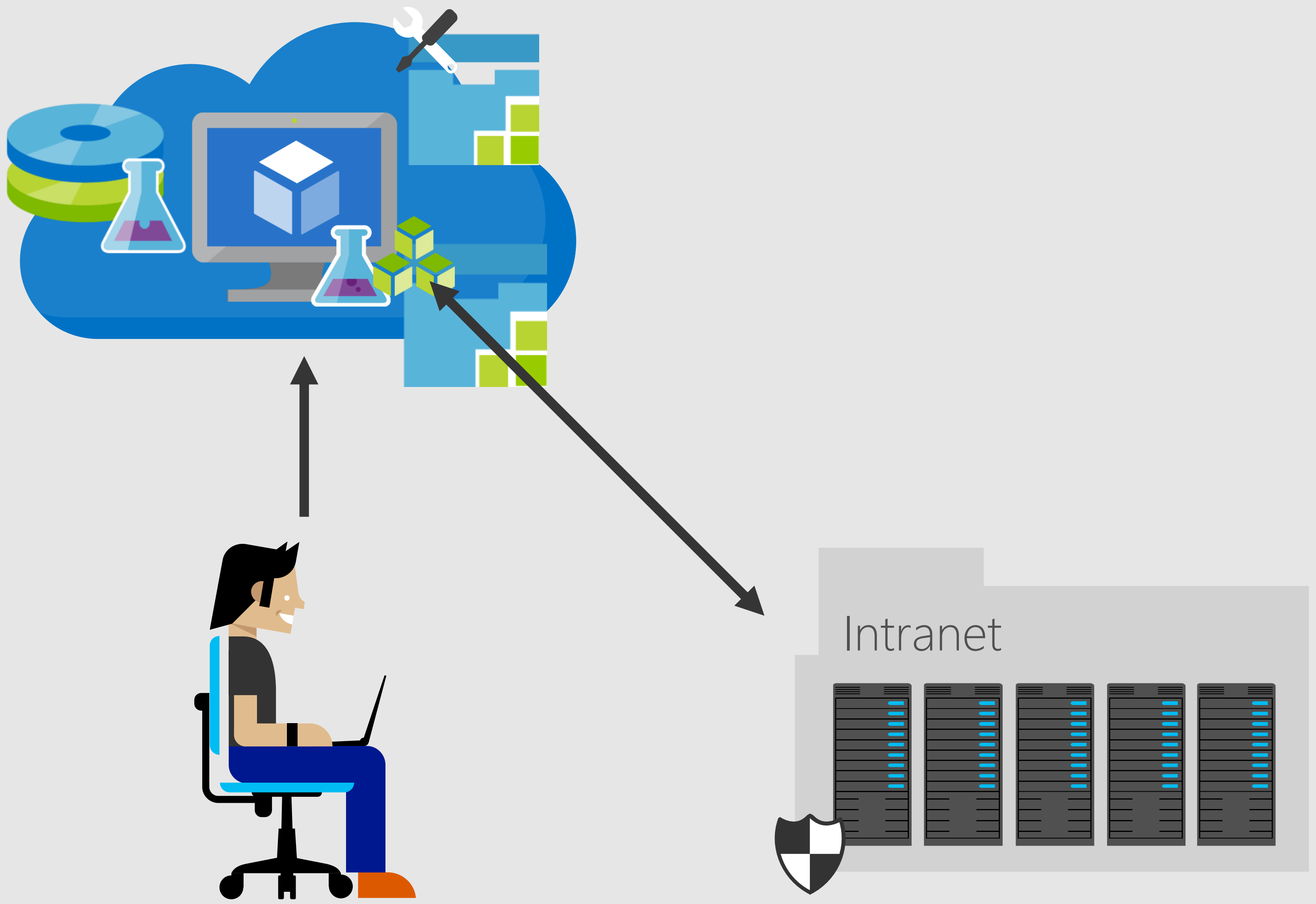


Provisioning machines

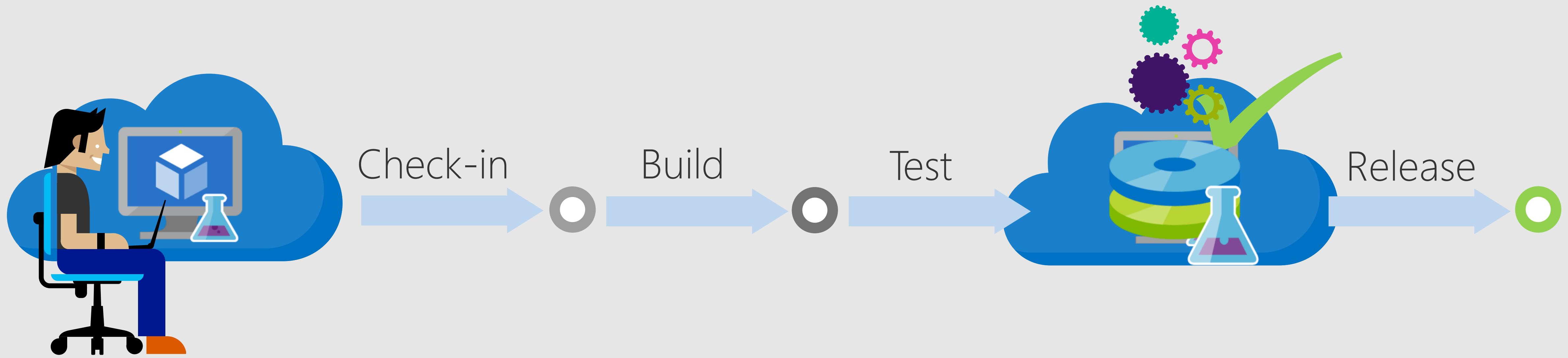


Scenarios

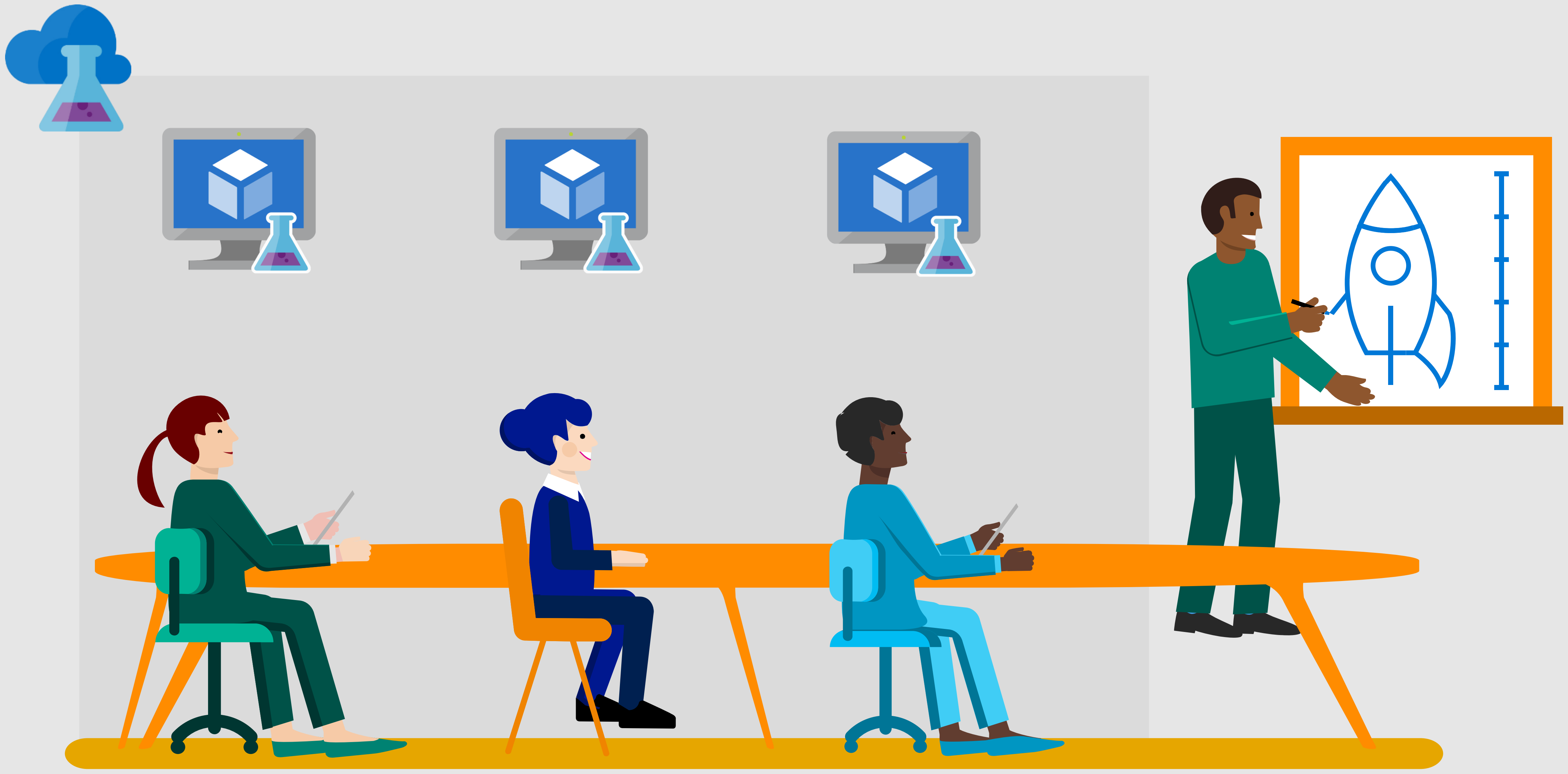
Developer machines



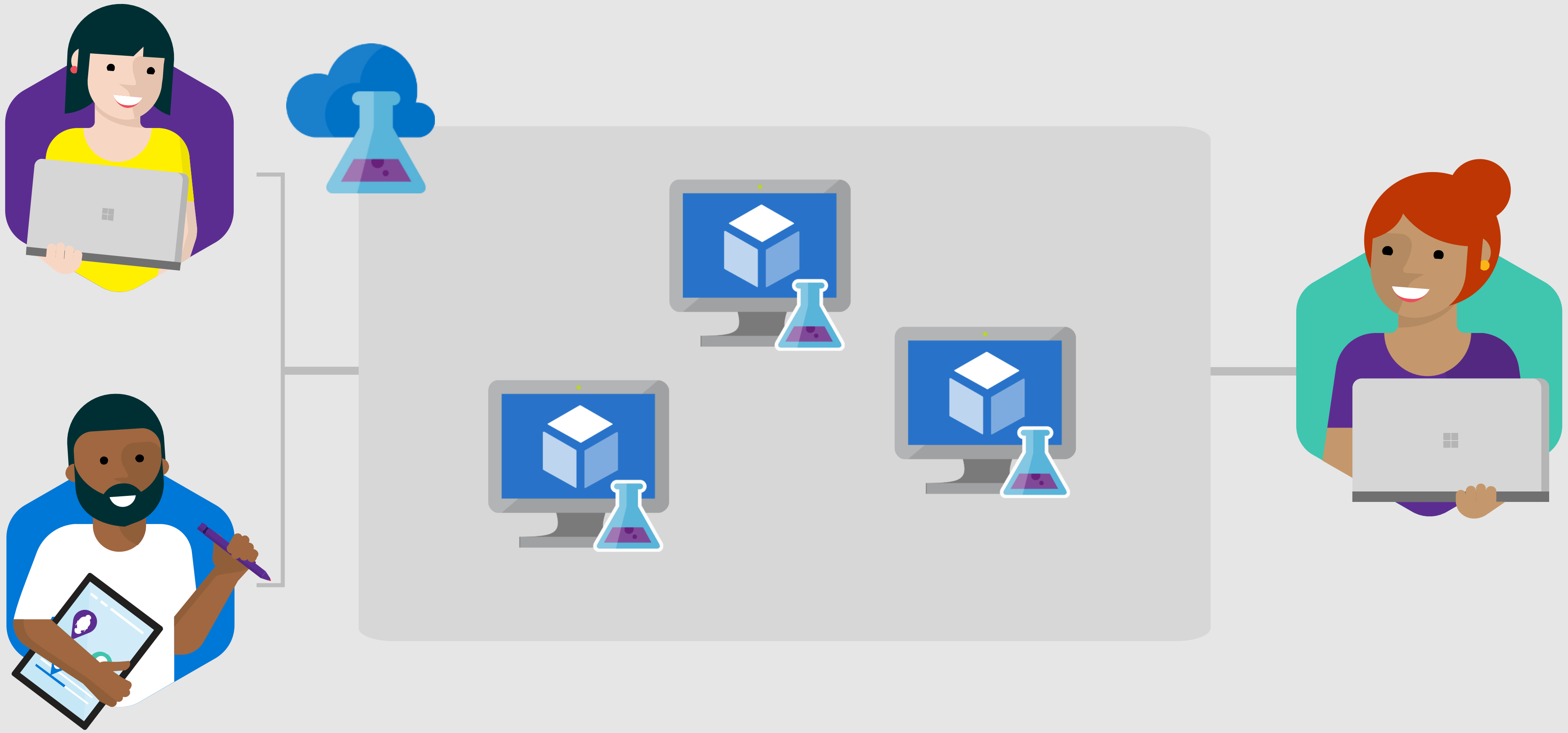
Test environments



Training / Education

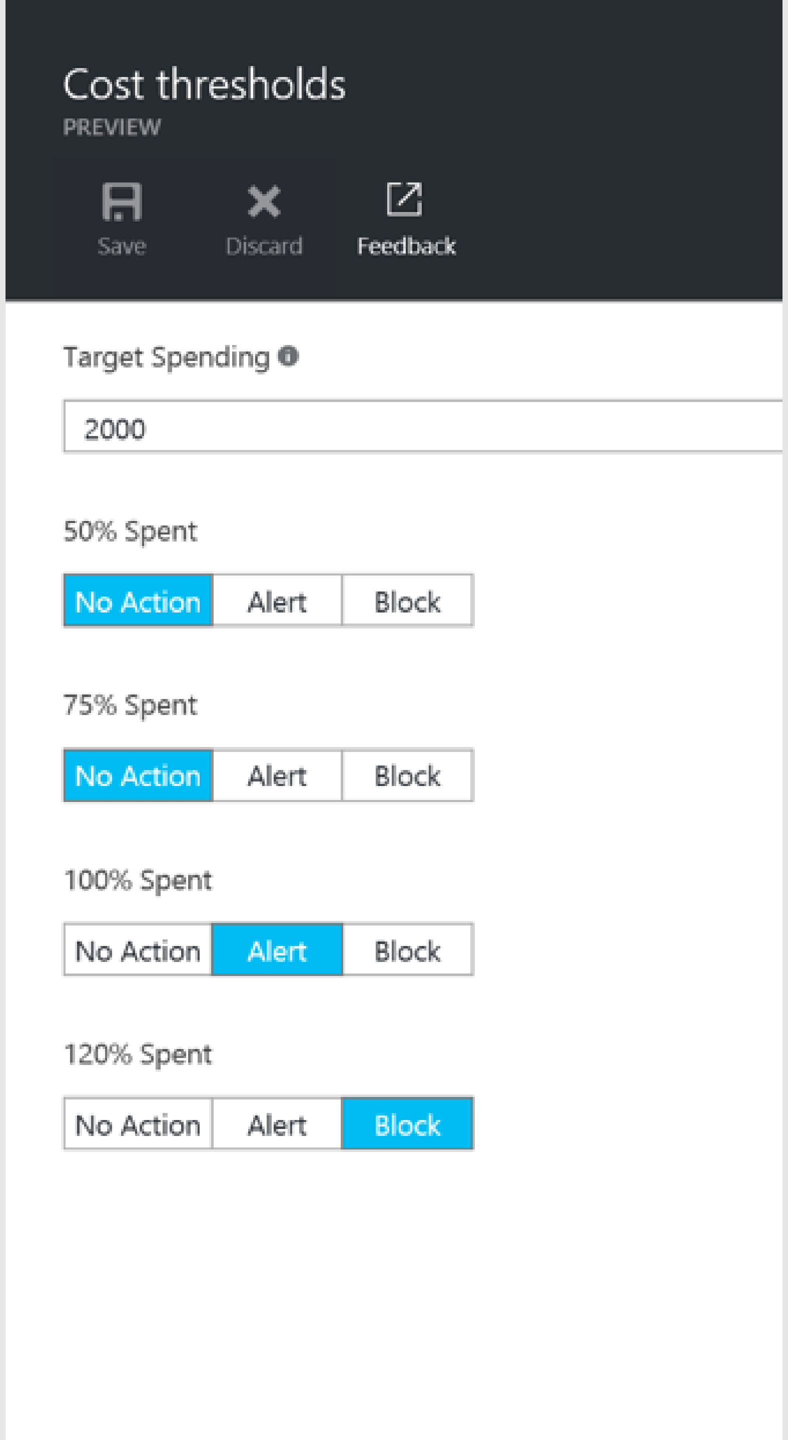


Trial / Hackathon / Demo



Key Features

Cost Threshold



Cost thresholds
PREVIEW

Target Spending: 2000

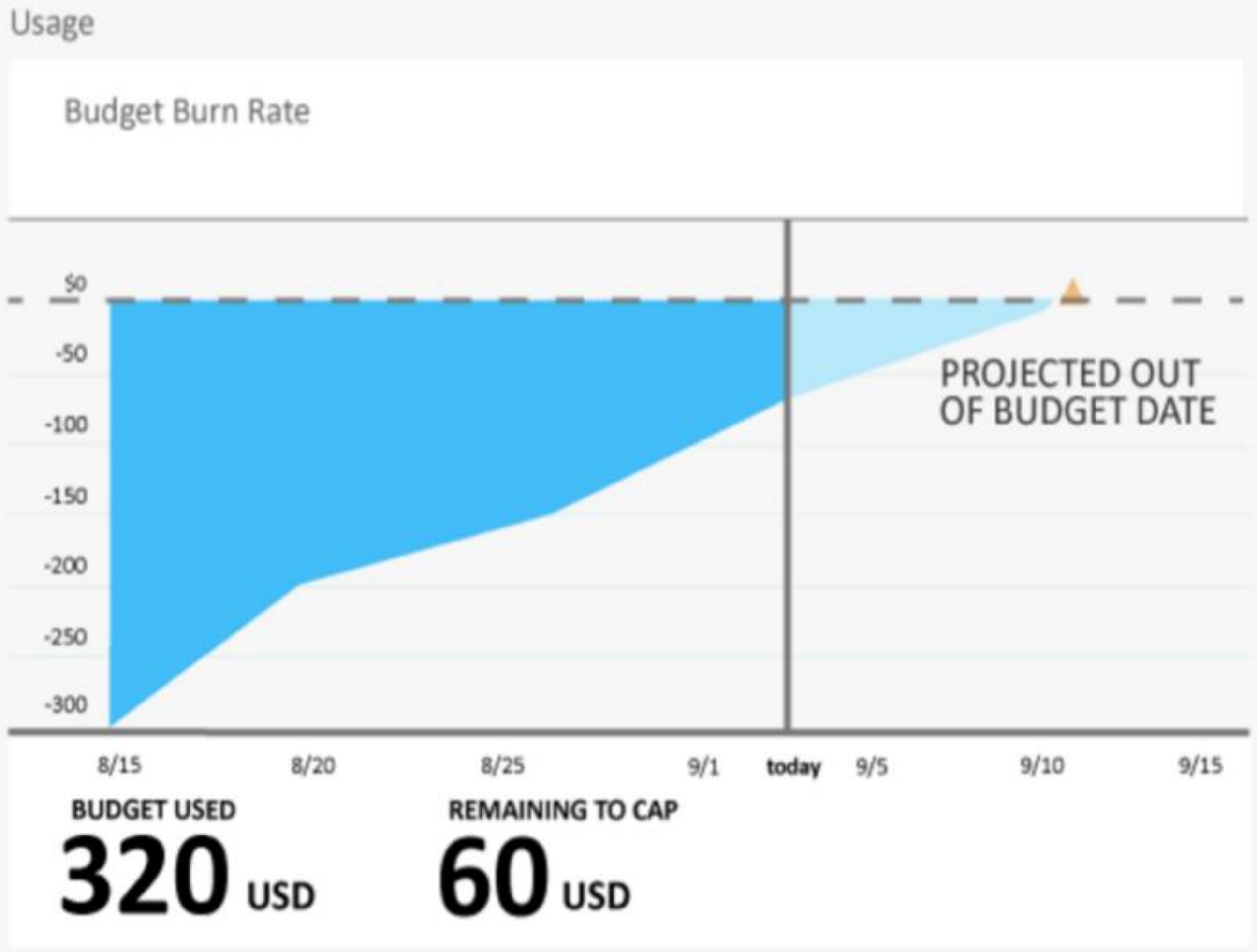
50% Spent: [✓ No Action] Alert Block

75% Spent: [✓ No Action] Alert Block

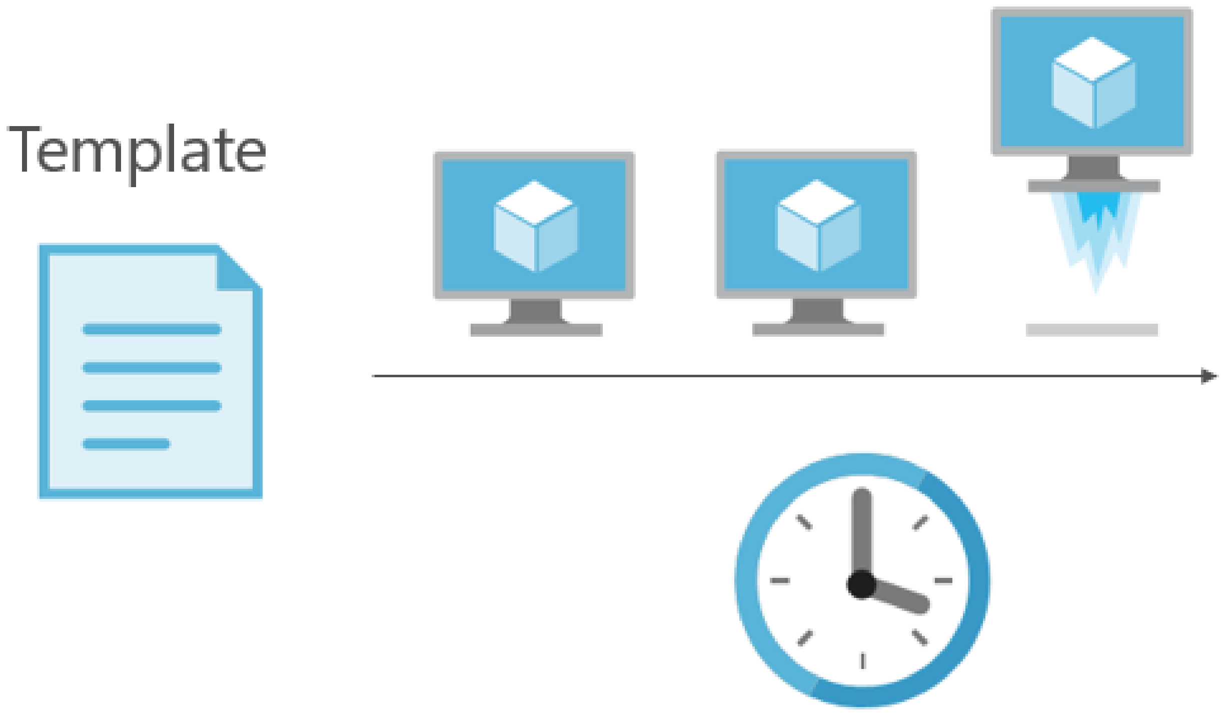
100% Spent: No Action [✓ Alert] Block

120% Spent: No Action Alert [✓ Block]

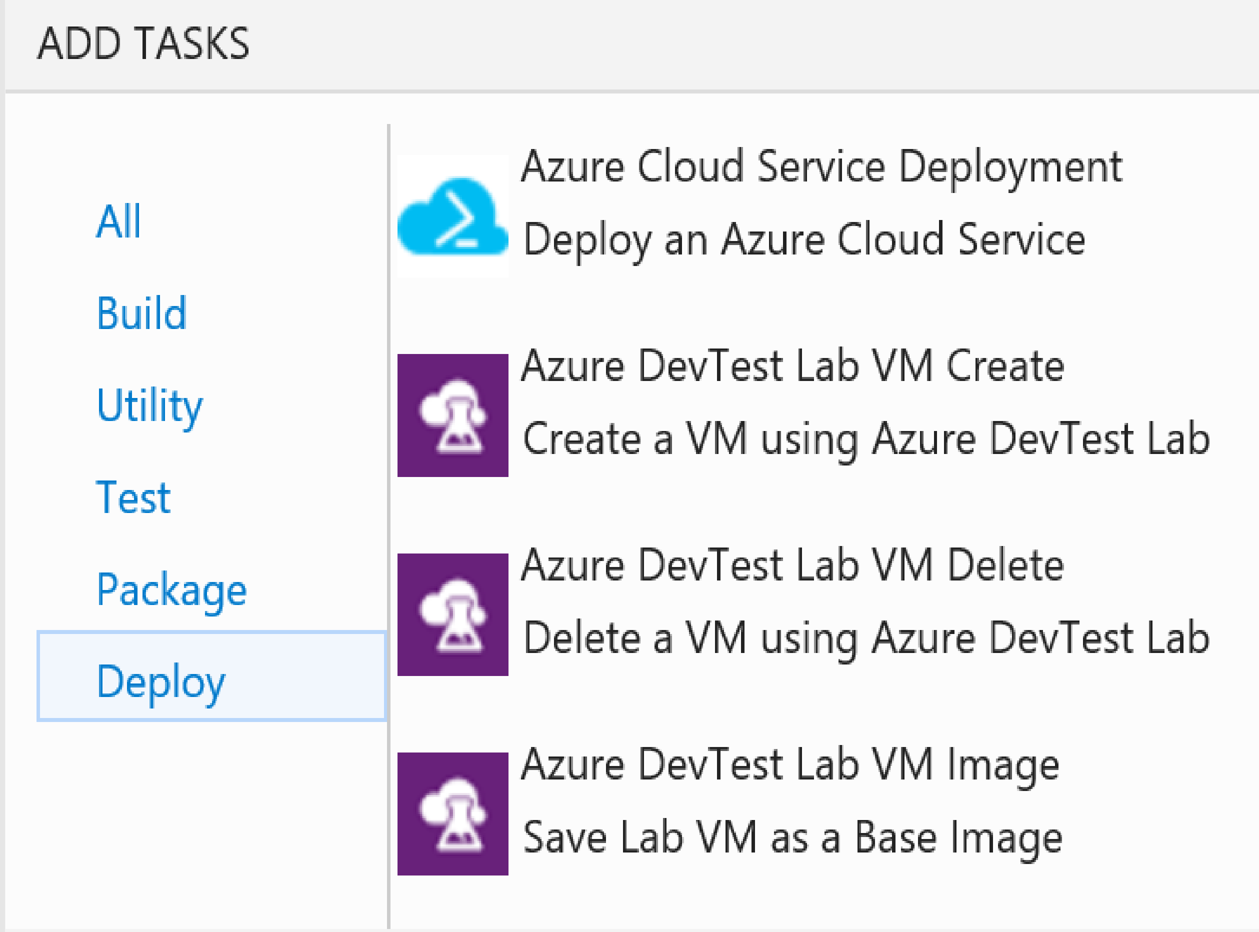
Cost Threshold Burn Rate



Claimable VMs



Azure DevOps Tasks



ADD TASKS

All
Build
Utility
Test
Package
Deploy

Azure Cloud Service Deployment
Deploy an Azure Cloud Service

Azure DevTest Lab VM Create
Create a VM using Azure DevTest Lab

Azure DevTest Lab VM Delete
Delete a VM using Azure DevTest Lab

Azure DevTest Lab VM Image
Save Lab VM as a Base Image

Microsoft