

Continuous Delivery Using Azure DevOps Services





ACCENTIENT EDUCATION SERIES

Committed to training success

www.accentient.com

Continuous Delivery Using Azure DevOps Services

Course Number:	CDADS
Version:	2018.11
Software version:	ADSvc

Copyright © 2018 Accentient, Inc. All rights reserved.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Accentient, Inc.

Images and Artwork

Images and artwork in this book were licensed from Corbis or Getty Images, downloaded from Openclipart.org, or obtained through Flickr under the Creative Commons 3.0 license.

All trademarks referenced are the property of their respective owners

Disclaimer

While Accentient takes great care to ensure the accuracy and quality of these materials, all material is provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

DevOps: Continuous Delivery Using Visual Studio Team Services

Course Introduction

"We need to figure out a way to deliver software so fast that our customers don't have time to change their minds."

- Mary Poppendieck



Module Backlog

What are we forecasting for this module?

- Course overview
- Team formation
- Trainer & student introductions
- Environment setup
- Introduce case study
- Working agreement



Course Overview

This course demonstrates how the fusion of people, process, and products can enable the continuous delivery of working software to end users.

- Increasing flow at scale
- Increasing feedback within the team
- Increasing feedback with stakeholders
- Fostering a culture of experimentation
- Fostering a culture of continuous improvement



Prerequisites

- Ideally, you ...
 - Work on a team
 - Develop and deliver a software product
 - Use Microsoft tools and technologies
 - Are familiar with the Scrum framework
 - Want to improve product and delivery quality



Form Into Teams

 | 5 MIN

If necessary, your instructor will facilitate the creation of equally sized (5 team members or less), cross-functional, collocated teams.

- Identify yourself by role ...
 - Software development
 - Visual Studio Team Services/Team Foundation Server
 - ALM/DevOps
 - Azure
 - Scrum
- Form into cross-functional teams (of 5 members or less)
- Collocate
- Introduce yourself (if necessary)
- Decided on a team name (i.e. “Team Blue”, “Avengers”, “Backlog Boyz”, etc.)
- Write your name and team name where it is visible to others in the class

What is the name of your team? _____

Who are your team members? _____

Introductions

- Name, company, title/role
- Software development experience
- Scrum experience
- VSTS/TFS experience
- Expectations



Collaborating as a Team

- There are many opportunities for collaboration in this course
 - Some tasks, however, must be performed by *one* team member
- All tasks will be marked with an appropriate icon ...



The team can self-organize and execute the task however they decide

Only the "leader" should execute this task

Only the "followers" (not the leader) should execute this task

Everyone on the team should execute this task

Everyone on the team should execute this task (working in pairs)



In this activity, working on your own or as a pair, you will install the files required by this class.

Dependencies

- Signed in with *local administrator* permissions

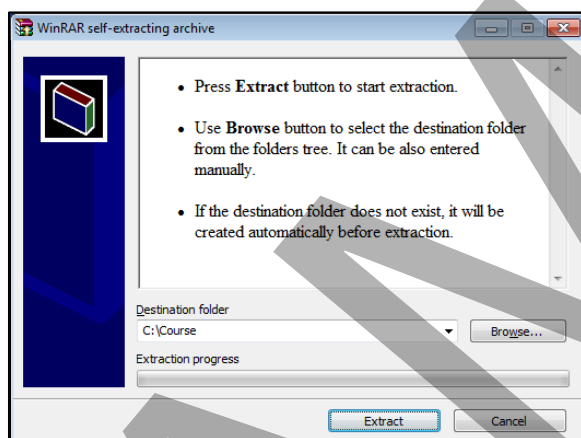
1. Verify that the **C:\Course** folder does not exist.

Note: If this folder already exists, then this computer may have been used for a prior training class. Please check with your instructor if you suspect that this is the case.

2. If necessary, copy the courseware (self-extracting) file to your desktop.

This file may already be on the desktop. If not, you may have to ask your instructor for help locating and/or copying this file. If you cannot locate this file, please email support@accentient.com to obtain a copy.

3. Double-click the courseware (self-extracting) file.
4. Specify **C:\Course** as the **Destination folder** and click **Extract**.



The folder **C:\Course** will be created during the process as well as one or more subfolders. Take a moment and explore the files that were installed.

Our Local Development Environment

- Visual Studio 2017 Enterprise Edition
 - All features installed
 - Current updates
- SQL Server 2014 or newer
 - Any edition, including Express/LocalDB
- Microsoft Office 2013 or newer (optional)

Accentient

Our Shared Development Environment

- We will be using a shared instance of Visual Studio Team Services
- We will be working in teams and each team will ...
 - Be collocated
 - Have its own team project
 - Collaborate on all work in this class
- Each team member will ...
 - Need a Microsoft Account



Accentient

Create VSTS Account and Team Project

 | 5 MIN

In this activity the team leader will create a new Visual Studio Team Services account and team project.

Who will be your team leader? _____

Which case study will you be working on? (ask your instructor) _____

Dependencies

- You have a Microsoft Account and know the password

1. Open your browser and navigate to <http://visualstudio.com/vsts>.
2. Follow the steps to sign up for a free **Visual Studio Team Services** account.



- Pick a memorable account name (perhaps a variation of your team's name).

Create a team project as part of the process, select ...

- Manage code using: **Git**
- Project name: **<case study>** (e.g. *AdventureWorks*)
- Organize work using: **Scrum**

Which Azure datacenter (region) will host your account? _____

What is your team project's URL? _____

3. Navigate to the team project's home page.
4. From the  settings menu at the top, select **Security**.
5. With the default team selected on the left, click the **Member of** link on the right.
6. Click **+ Add**, enter/select **Project Administrators**, and save changes.
7. Remove the **Contributors** group and click refresh .

The default team should only be a member of *Project Administrators* and *Project Valid Users* now.

8. Click the **Members** link.
9. Click **+ Add**, enter the **Microsoft Account** of one of your teammates, and save changes.
10. Repeat the above steps to add the rest of your teammates.

How many team members are listed? _____

In this activity the rest of the team members will join the newly created team project.

Dependencies

- Your team's Visual Studio Team Services account and team project have been created
- You have a Microsoft Account and know the password
- Your Microsoft Account has been added to the default team

1. Open your browser and navigate to the URL of your team project.

Tip: Ask your team leader for the exact URL. Consider bookmarking that page.

2. If prompted, enter your **Microsoft Account**, password, and any other required information.

If you have any difficulties signing in, double-check your account and password. If that doesn't solve the problem, ask your team leader to double-check the team membership and permissions. If you are still blocked, check with the instructor.

3. Close any announcement or other dialogs.
4. In the upper-right corner, click your avatar, click **My profile**, and verify that your display name and preferred email address are accurate.
5. If necessary, change your picture to something more social.
6. Save your changes.
7. If you are working in a pair, sign out, and have your buddy follow the previous steps to set his or her information and profile picture (optional).

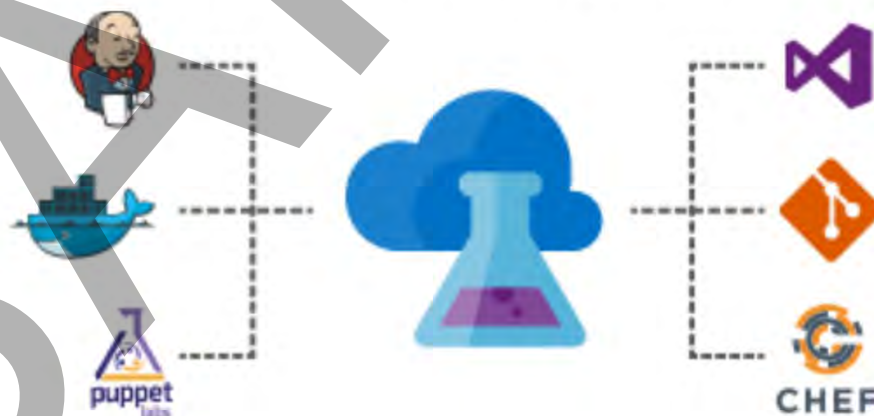
Using Azure for DevOps

- Platform-as-a-service (PaaS) environments hosted in Microsoft Azure enable the continuous delivery
 - Use automation technologies such as Chef, Puppet and PowerShell to speed up deployment and configuration of your solution across testing, staging and production environments
 - Embed performance testing as part of your release cycle
 - Use Application Insights to monitor environments and relentlessly improve quality

Accentient

Our Shared Deployment Environment

- Each team will be using their own Azure account to setup and configure the Continuous Deployment environment



Accentient

Identify Your Team's Azure Account

 | 5 MIN

In this activity the team leader will identify the Azure account your team will use during class.

Which Azure account will you be using in class? _____

Note: If your team does not have an Azure account, a free one can be quickly provisioned. This process requires a phone number and credit card number – for verification purposes only. For more information, or to start a free account, visit <https://azure.microsoft.com>.

Dependencies

- An Azure account with an active trial or subscription

1. Open your browser and navigate to <https://portal.azure.com>.
2. Sign in using the Microsoft Account that administers the Azure account.
3. Review your dashboard and subscription(s).

What active subscription(s) are listed? _____

What's the credit remaining/balance? _____

Note: For some accounts and subscriptions, it may be easier to find the above information by navigating to <https://account.azure.com/subscriptions>.

4. Close the browser.

We will return to Azure in a later lab.

Case Study Kickoff



5
MIN

- Meet the sponsor
- Introduce the case study
- Background information



Working Agreement

What should be our guidelines for these?

- Off-track discussions
- Lunch
- Break times and signals
- Electronics such as phones, tablets, and laptops
- End of day timing

Accentient

The Three Ways

- Flow
 - Accelerating development work to operations to the stakeholders
- Feedback
 - Creating ever-safer systems of work
- Continual Learning and Experimentation
 - Fostering a high-trust culture and openness to improvement
- Note: The “three ways” were mentioned in the *Phoenix Project* and are the principles underpinning DevOps ... and this course



Accentient

Module Review

What have we accomplished and learned?

- Teams formed
- Introductions made
- Instructor's backlog emerges
- Environment configured
- Case study introduced
- Agenda and logistics explained

Accentient

DevOps: Continuous Delivery Using Visual Studio Team Services

Integrating Continuously

"If it hurts, do it more often"

- Martin Fowler



Module Backlog

What are we forecasting for this module?

- Automated Testing
 - Unit tests
- Automated Builds
 - Running tests during build
- Continuous Integration
 - Team practices
 - Test Impact Analysis
- Continuous Integration +



Faster and Faster Feedback Loops

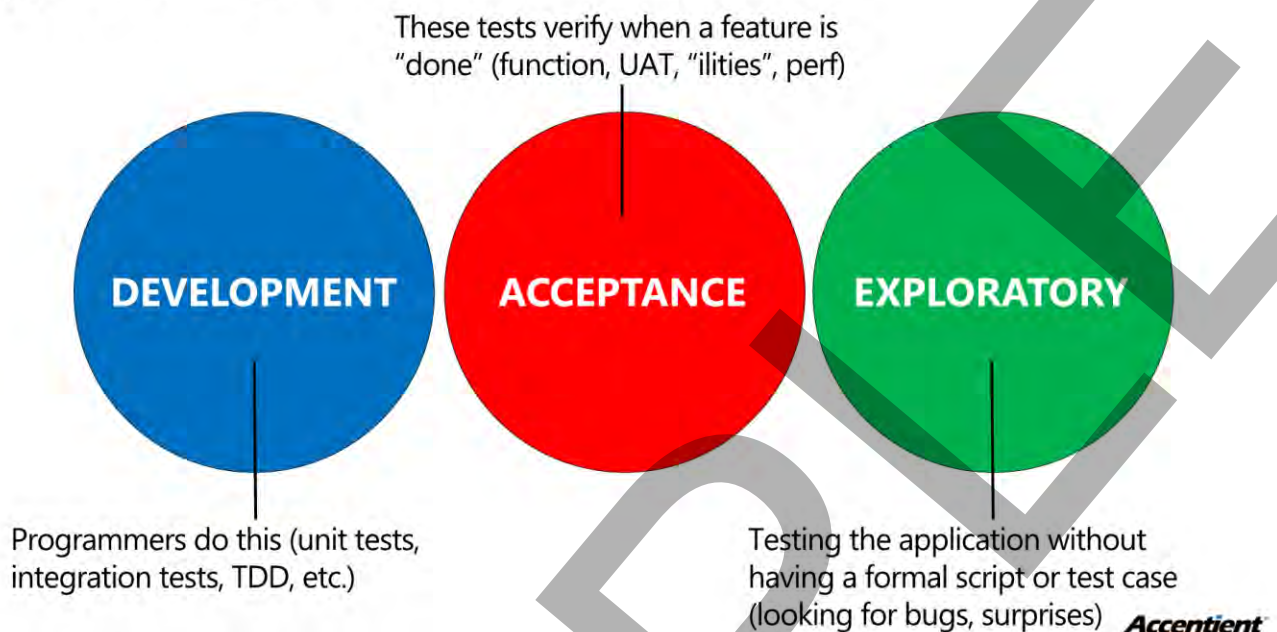
- One of your team's goals should be to create faster and faster feedback loops
 - Automated testing, building, and integration can help achieve this
- The team can be informed when a change is introduced that alters the functioning and deployable state of the software
 - Developers can fix the problems sooner before switching context
- Quality is increased while waste is reduced

Accentient

Automated Testing

Accentient

Types of Tests



Development Tests == Unit Tests

- A unit test is a fast, in-memory, consistent, automated, and repeatable test of a functional unit of work in the system
- A unit of work is any functional scenario in the system that contains logic
 - It should test a single method
 - Tests that span multiple methods are referred to as integration tests
- Path testing
 - Happy path tests use good data to validate the requirement
 - Sad path tests exercise validation and error handling logic

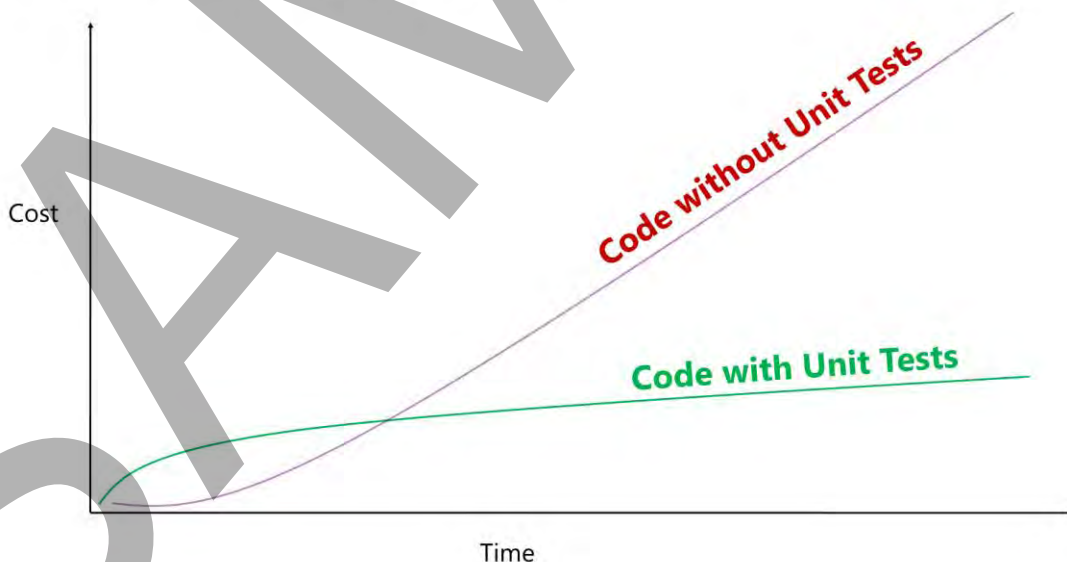
Accentient

Why Do We Write Unit Tests?

- Safety net – we know the code is working
- Reduce bugs
- Create better designs
- Enable meaningful refactoring
- Keep focus on getting to done
- Force developer to think about goals
- Instant gratification
- Verify the existence of a bug
- Documentation

Accentient

The Cost of Changing Code



Accentient

Unit Testing Frameworks

- Test Explorer can run any unit test framework that provides a Test Explorer adapter
 - First party (Microsoft) or third party (community)
- A Visual Studio solution can contain unit test projects that use different frameworks and that are targeted at different languages
 - Test Explorer runs them all



Accentient

What Frameworks Have You Used?



5
MIN

Discuss your experiences with any of these
or other unit testing frameworks

What were the pros and cons?

Accentient

Unit Test: Example

```
[TestMethod]
public void Withdraw_ValidAmount_ChangesBalance()
{
    // arrange
    double currentBalance = 10.0;
    double withdrawal = 1.0;
    double expected = 9.0;
    var account = new CheckingAccount("JohnDoe", currentBalance);
    // act
    account.Withdraw(withdrawal);
    double actual = account.Balance;
    // assert
    Assert.AreEqual(expected, actual);
}
```

Accentient

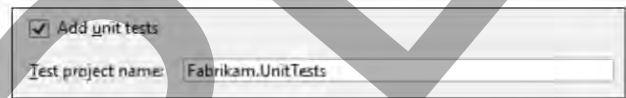
How Does a Unit Test Pass or Fail?

- Unit Tests are assumed to be in a passing state when they begin execution
- There are various ways a unit test can pass:
 - An *Assert* statement passes: **Assert.AreEqual(a, b);**
 - An *expected* exception was thrown
 - The test runs without error and without any *Assert* statement
 - The unit test contains no code
- There are various ways a unit test can fail:
 - An *Assert* statement fails
 - An *unexpected* exception was thrown

Accentient

Visual Studio Test Projects

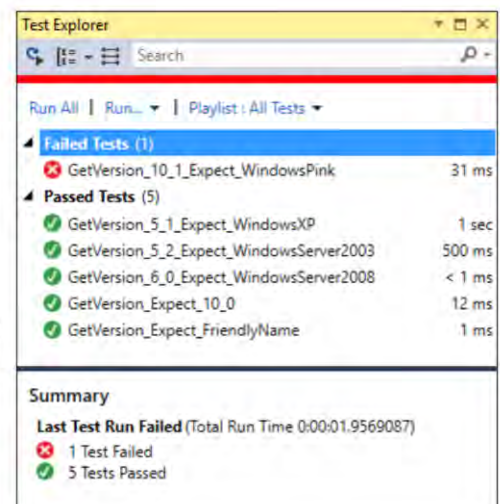
- Test projects are Visual Studio projects designed to hold the testing code used to test deliverable code
 - They are independent of the tested code
 - They reference the tested code, but the tested code does not reference the test project
 - They can contain different types of tests
- Tip: Consider creating a separate unit test project within the Visual Studio solution
 - Microsoft suggests exactly this



Accentient

Test Explorer

- Your unit testing “dashboard”
 - Run tests from multiple test projects
 - Use different unit test frameworks
- As you write and run unit tests, Test Explorer displays the results
 - In default groups of Failed tests, Passed tests, Skipped tests, and Not Run tests
 - You can change these groupings



Accentient

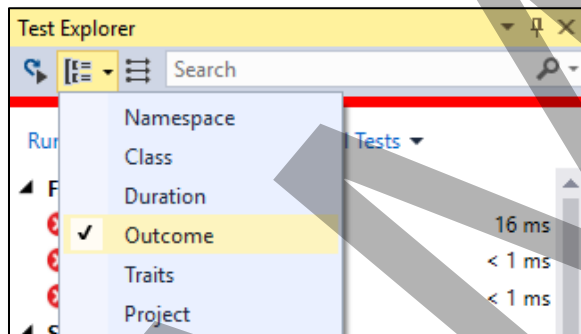
In this activity, working on your own or as a pair, you will build the case study solution and run all of the unit tests.

Dependencies

- Case study code has been imported into VSTS
1. If necessary, launch **Visual Studio 2017** and ensure that you are connected to your team project's primary repository and that you are in the **master** branch.
 2. From the **Sync** page in **Team Explorer**, click **Fetch** and then **Pull** any incoming commits from your colleagues.
 3. Open and build the case study solution.
 4. From the **Test** menu select **Windows > Test Explorer**.
 5. **Run All** of the tests.

How many tests passed? Failed? Were skipped? _____

Note: You can group the test results in various ways.



Take some time and look at the unit tests themselves. As you can see, they are not really testing anything.

- Passing tests – these are passing because the unit test method bodies are *empty*
 - Failing tests – these are failing because of *Assert.Fail()* statements
 - Skipped tests – these are skipped because of *Assert.Inconclusive()* statements
6. Minimize **Visual Studio**.

Automated Builds



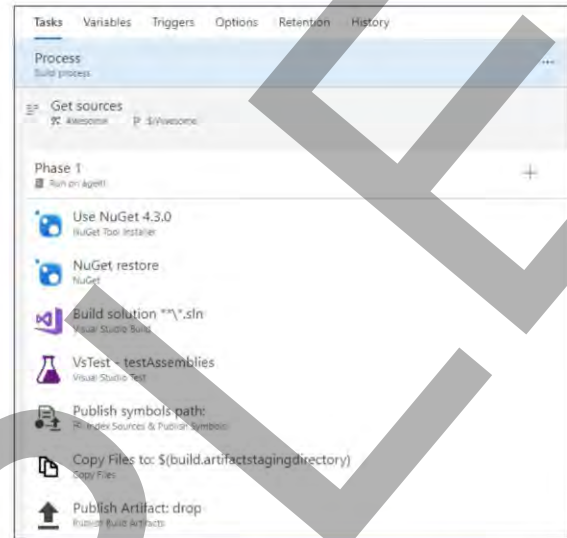
Team Build

- A component of TFS and VSTS
- The Build system enables you to:
 - Create, manage, and run build scripts (build definitions)
 - Automate the compiling and testing of simple to complex apps
 - Perform other tasks, such as deploying to various environments



Build Automates Everything

- Automates all kinds of tasks
 - Getting the code
 - Restoring NuGet packages
 - Building
 - Testing
 - Packaging
 - Deploying
 - Any other scripts and utilities

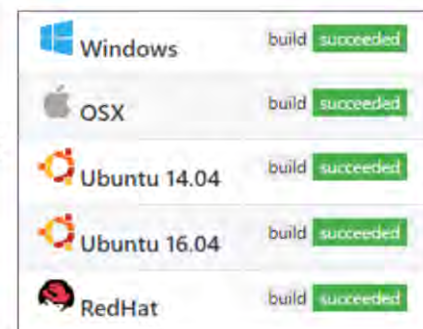


Visit <http://bit.ly/2eKvLuV> for more information



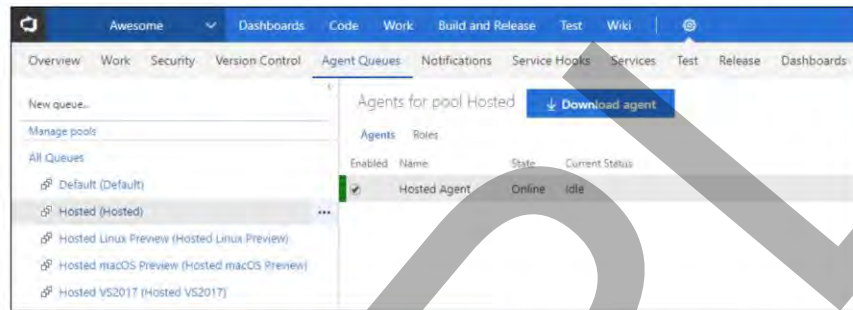
Build Agents

- To build your code you need at least one agent
 - An agent is installable software that runs one build at a time
 - You can add additional agents as you need them
- Agents can be hosted or private
 - Hosted agents are for VSTS and are maintained by Microsoft
 - Private agents can be installed on Windows, Linux, or OSX



Hosted Agent Pool

- When using VSTS, you can use the Hosted Agent Pool
 - Microsoft takes care of the maintenance and upgrades
 - All accounts have one *free* agent with a limited free build minutes
 - Hosted agents are deployed with specific software

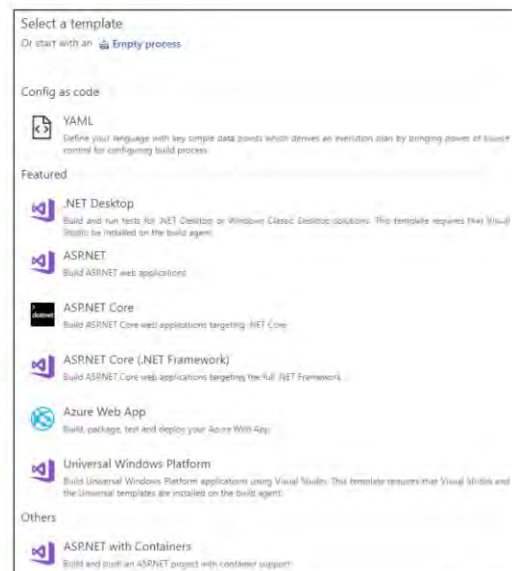


Visit <http://bit.ly/21Yn6uL> for more information



Creating a Build Definition

- Select a template
 - Configuration as code (YAML)
 - Can start with an empty solution
- When defining a new build, specify ...
 - Agent queue
 - Where to get sources (repository)
 - Phases
 - Build steps (tasks)
 - Variables
 - Triggers
 - Retention policy
 - Other build options



Create a Build Definition

 | 5 MIN

In this activity, your team will select someone to create a build definition.

Who will be creating the build definition? _____

Dependencies

- Updated case study code has been pushed to VSTS

1. In the browser, navigate to the **Build and Release** hub of your team project.
2. On the **Builds** page, add a new build definition, selecting the **ASP.NET** build template with default settings.
3. In the build editor, remove the **Visual Studio Test** and **Index Sources & Publish Symbols** tasks.
4. In the build editor, select the **Visual Studio Build** task and change the **MSBuild Arguments** to the following:

```
/p:DeployOnBuild=True /p:WebPublishMethod=FileSystem /p:DeployDefaultTarget=WebPublish  
/p>DeleteExistingFiles=True /p:publishUrl=$(build.artifactstagingdirectory)
```

Note: These changes alter the default deployment behavior. They will be used in a later activity.

5. In the **Process** settings, select the **Hosted** Agent queue.
6. In the **Get sources** settings, select your main repository (not Universal).
7. **Save** the build definition, giving it the name **Nightly**.

Queue and Run an Automated Build

 or  | 5 MIN

In this activity, working on your own or as a pair, you will queue and run an automated build.

Dependencies

- The *Nightly* build definition has been created

1. In the browser, navigate to the **Build and Release** hub of your team project.
2. On the **Builds > Definitions** page review all build definitions.

Do you see the *Nightly* build definition? _____

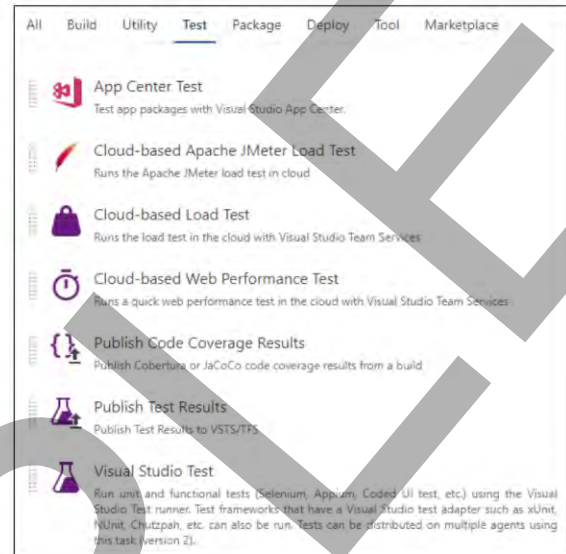
3. Queue a **Nightly** build using the default settings.

Did the build succeed? How long did it take to run? _____

Note: No test results were generated because the build definition doesn't have a test task.

Running Tests From Automated Builds

- As part of your definition, you can add one or more test tasks
 - Before running tests, you should confirm that the project contains tests, that they pass, and that they are also stored in the repository

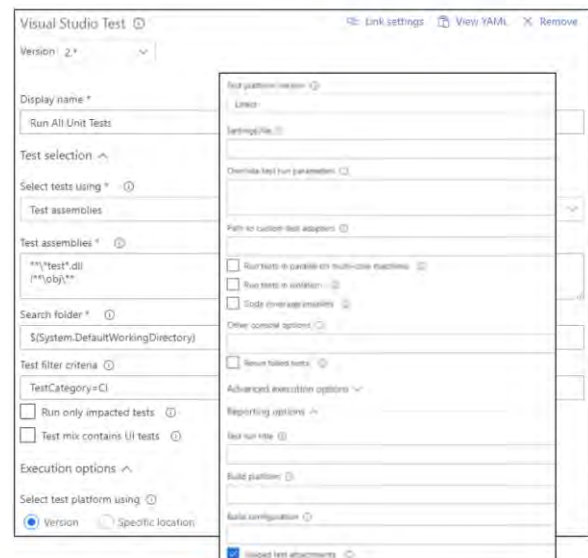


Visit <http://bit.ly/2naaWvy> for more information



The Visual Studio Test Task

- Run tests with the Visual Studio test runner
 - These are primarily unit or integration tests
 - Uses `vstest.console.exe` to run MSTest, NUnit, xUnit, Mocha, Jasmine, etc. tests
 - Test Impact Analysis (TIA) capability can be enabled



Visit <http://bit.ly/2xPqJ7f> for more information



Add a Visual Studio Testing Task

3 people | 5 MIN

In this activity, your team will select someone to clone a build definition and add a Visual Studio Testing task.

Who will be updating the build definition? _____

Dependencies

- The *Nightly* build definition has been created and run successfully
1. On the **Builds** page of the **Build and Release** hub, **Clone** the **Nightly** build definition.
 2. Add a **Visual Studio Test** task and drag it under the **Visual Studio Build** task.
 3. Change the task's display name to **Run All Unit Tests**.

Notice the default is to run all tests in any assembly with "test" in its name.

4. Change the name to **Nightly.UnitTests** and **Save** the build definition with an appropriate comment.

These comments are visible on the *History* page of a build definition.

Run Tests During a Build

2 or 3 people | 5 MIN

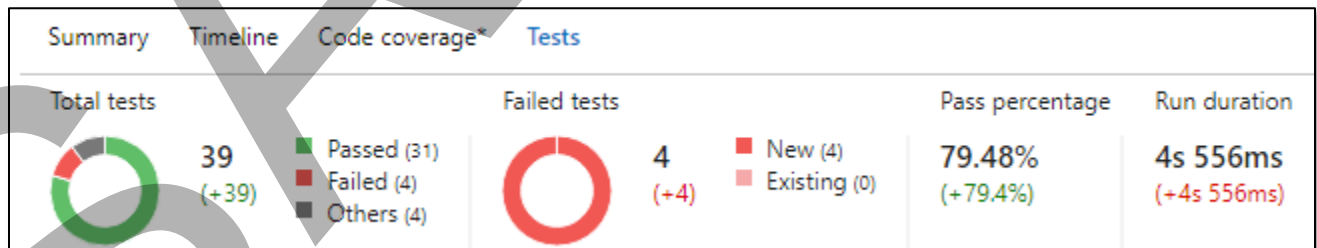
In this activity, working on your own or as a pair, you will run an automated build to include unit tests.

Dependencies

- The *Nightly.UnitTests* build definition has been created and configured to run all tests
1. On the **Builds** > **Definitions** page, queue a **Nightly.UnitTests** build using default settings.

Did the build succeed? How long did it take to run? _____

2. Click **Tests** to see details about how many tests passed, failed, or otherwise.



By default, 1 failed test = a failed build. Although it's always a bit discouraging to see this, it's good feedback that needs to be delivered continuously. For more information, visit <http://bit.ly/2oulpCE>.

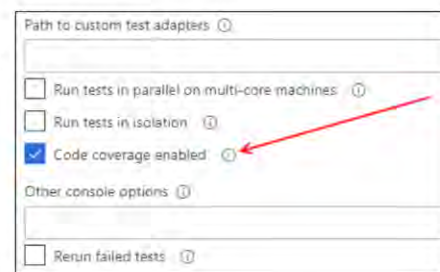
Code Coverage

- Code coverage lets you analyze how much of your code was exercised during the execution of a set of unit tests
 - Results are available as a percentage, as well as graphically
 - Applies to both managed and unmanaged (native) code
- Code Coverage is a way to measure the effectiveness of your unit tests
 - On a block-by-block or even line-by-line basis
- Requires Visual Studio Enterprise edition
 - *DotCover*, *OpenCover*, and *NCover* are 3rd party alternatives

Accentient

Gathering Code Coverage Data

- Code coverage data can be gathered during a build
 - Configure this by enabling Code Coverage on the Visual Studio Test task
- Visual Studio Enterprise edition must be installed on the agent machine
 - VSTS hosted agents already have this



Accentient

Enable Code Coverage

 | 5 MIN

In this activity, your team will select someone to clone a build definition and enable code coverage.

Who will be updating the build definition? _____

Dependencies

- The *Nightly.UnitTests* build definition has been created and run
1. On the **Builds** page of the **Build and Release** hub, **Clone** the **Nightly.UnitTests** build definition.
 2. Select the **Run All Unit Tests** task and set its **Code coverage enabled** option.

This option will enable the collection of code coverage information during the run and upload the results to VSTS. This is supported for .Net and C++ projects only.
 3. Change the name to **Nightly.UnitTests.CodeCoverage** and **Save** the build definition with an appropriate comment.

Analyze Code Coverage

 or  | 5 MIN

In this activity, working on your own or as a pair, you will run an automated build and review the code coverage.

Dependencies

- The *Nightly.UnitTests.CodeCoverage* build definition has been created to include running all tests and collecting code coverage information
1. Queue a **Nightly.UnitTests.CodeCoverage** build using default settings

The build outcome should be the same as before.
 2. Review the **Code coverage** results.

You may have to download the .coverage file and open it in Visual Studio. This requires Visual Studio Enterprise edition. See <http://bit.ly/2ocBeBT> for more information.

As you can see from the .coverage file, there isn't much detail here, as these are simulated unit tests.

Using Code Coverage as a Metric

- Don't use Code Coverage as a "feel-good" metric
 - Low numbers certainly tell you that you need more tests
 - High numbers don't necessarily mean that you've written good tests
- Just because you're fully covered doesn't mean you're fully tested
- Use the code coverage metric wisely



Continuous Integration



Automated Regression Testing

- After changing any code, you should build and run tests
 - You do this by re-running all applicable tests to determine whether the changes broke anything that worked prior to the change
- Automated builds running verification tests is a great way of automating regression testing

Accentient

Continuous Integration (CI)

- A development practice where developers integrate (check-in) their code changes regularly
 - Integration becomes a part of the natural rhythm of coding
 - Enables test-code-refactor and test-first development
 - The team can progress steadily forward by taking small steps



Visit <http://bit.ly/KEgGa> for more information

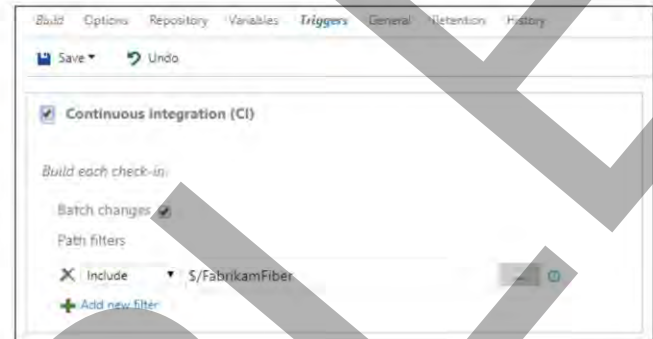
Accentient

Build Supports Continuous Integration (CI)

- Use the Continuous Integration (CI) trigger

- *Batch changes* option

- Ensures that only one such build is running at any time
- Any pushes/check-ins that occur while a build is running are combined and built together when the current build completes



Visit <http://bit.ly/2ntaYRu> for more information



Practice Continuous Integration

- Adopt a “collective ownership” attitude within the team
 - Integrate regularly, no long-lived branches, swarming
- Configure TFS to build and test on check-in
 - Optimize the build process for fast feedback (e.g. < 60 seconds)
 - Test in a clone of the production environment
- Be transparent
 - Fix problems as fast as you find them



Enable Continuous Integration

 | 5 MIN

In this activity, your team will select someone to clone a build definition and enable Continuous Integration.

Who will be cloning the build definition? _____

Dependencies

- The *Nightly.UnitTests* build definition has been created and run
1. On the **Builds** page of the **Build and Release** hub, **Clone** the **Nightly.UnitTests** build definition.
 2. On the **Triggers** page, enable **Continuous Integration**.

Notice that you can enable *Batch changes*. This is helpful when you have many active team members and you want to reduce the number of running builds. With *Batch changes*, when a build is running, the system waits until the build is completed and then queues another build of all the pending commits.

3. Add a **Path Filter** to **Include** the **Code** path.

You can set *Branch* and *Path* filters to keep commits in other branches and folders from triggering a build. For more information on path filters, visit <http://bit.ly/2yq2UD1>.

4. Change the name to **CI** and **Save** the build definition with an appropriate comment.

Fix the Failing Unit Tests

 | 5 MIN

In this activity, your team will self-organize, return to Visual Studio, “fix” the failing unit tests, and then push the changes back to Visual Studio Team Services.

How many failing unit tests are there? _____

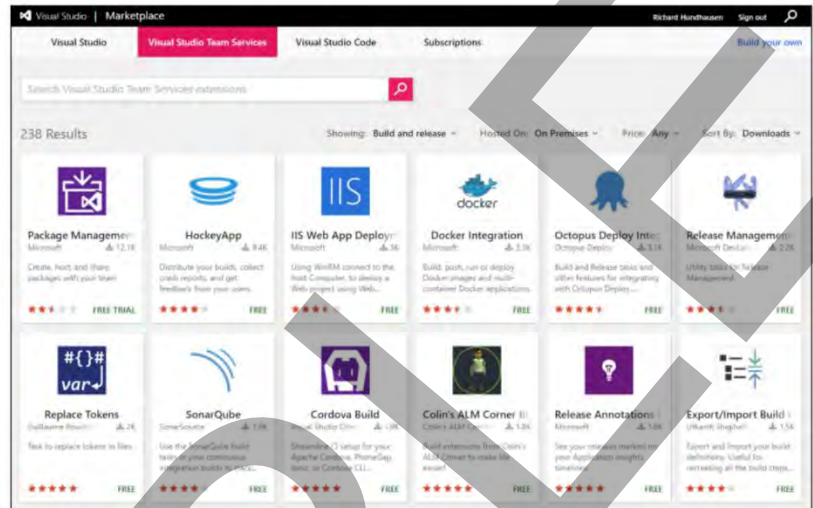
Dependencies

- The *CI* build definition has been created and contains a *Continuous Integration* trigger
1. Return to **Visual Studio** and ensure that you are in the **master** branch of the default repository.
 2. Locate all failing unit tests and comment out each **Assert.Fail();** statement.
 3. **Commit** and **Sync** changes back to Visual Studio Team Services.

Each push should trigger the CI build to run. Watch the build and see the number of passing tests increasing.

Build Extensions in the Marketplace

- The Visual Studio Marketplace has many Build extensions
 - Created by Microsoft and partners; many are free
 - <http://bit.ly/2ordS73>



Accentient

CatLight

- A notification app for developers
- CatLight shows the current status of continuous delivery and informs when attention is needed
 - See the status of the things you track in the tray




- See more details on the dashboard

Visit <https://catlight.io> for more information

Accentient

CatLight by Catlight.io is a build status notification that runs in your tray. It will notify you when builds complete and need your attention.

In this activity, working on your own or as a pair, you will install and use CatLight.

1. In the browser, click the  icon in the upper right and select **Browse Marketplace**.
2. Locate and install **CatLight** by **Catlight.io**.

This extension installs locally as a Windows application.

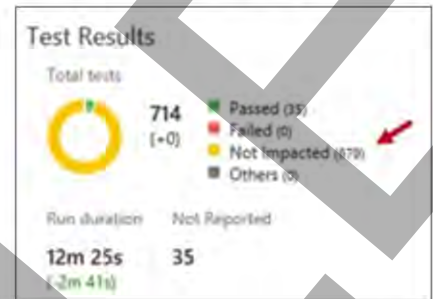
3. Configure CatLight for **VSTS**, specifying your account URL and authorizing permissions.
4. In CatLight settings, select all build definitions and click **Save**.
5. Create an account, specify other information, and then minimize CatLight to the tray.
6. Return to the **Builds** page of your team project's **Build and Release** hub.
7. Queue a **Nightly.UnitTests** build using default settings
8. As the build runs, review the status in CatLight.

You can see the build status in the tray (the cat's head will turn different colors). You can get notifications on build start and completion and also see the details on the dashboard. You can even queue a build from CatLight by right-clicking on it from the dashboard.

See <https://catlight.io/a/tfs-build-monitor> for more information.

Test Impact Analysis

- What if you have too many tests to run them all every time?
 - This is a good problem to have
- Test Impact Analysis (TIA)
 - Incremental validation by automatically selecting relevant tests
 - For a given code commit entering the pipeline TIA will select and run only the relevant tests required to validate that commit



Accentient

Configuring Test Impact Analysis

Test selection

Select tests using: Test assemblies

Test assemblies: `***test*.dll`
`!*\obj**`

Search folder: `$(System.DefaultWorkingDirectory)`

Test filter criteria:

- ☒ Run only impacted tests
- Number of builds after which all tests should be run: 50
- Test mix contains UI tests: ☐

Accentient

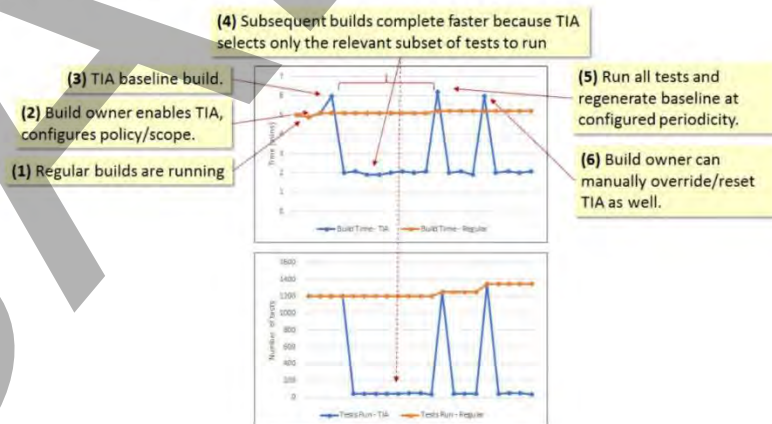
Test Impact Analysis: How Tests Get Selected

- Test Impact Analysis will look at an incoming commit, and select the set of relevant tests, as well as ...
 - Any previously failing tests
 - Any newly added tests
- Recommendation: Periodically bypass Test Impact Analysis and run all tests
 - This could be configured to run nightly

Accentient

Test Impact Analysis: Considerations

- TIA has overhead to build and maintain the mapping
 - Only use it in cases where a test run itself takes a fair amount of time to complete (e.g. > 10 minutes)



Accentient

Enable Test Impact Analysis

 | 5 MIN

In this activity, your team will select someone to enable Test Impact Analysis.

Dependencies

- The *CI* build definition has been created

1. On the **Builds** page of the **Build & Release** hub, locate and edit the **CI** build definition.
2. Select the **Run All Unit Tests** task.
3. Select the **Run only impacted tests** option.

After how many builds will all the tests be run again? _____

Test Impact Analysis stores the mapping between tests and source code. It is recommended to regenerate this mapping by running all tests, on a regular basis.

4. **Save** the build definition with an appropriate comment.

Note: The number and type of tests in our case study won't allow for much/any increase in performance, therefore we won't be evaluating the performance of Test Impact Analysis.

Specify Test Filter Criteria

 | 5 MIN

Since our case study isn't complex enough to reap the benefits of Test Impact Analysis, we will explore other ways to run a smaller scope of tests.

In this activity, your team will select someone to edit the CI build definition and specify test filter criteria.

Who will be editing the CI build definition? _____

Dependencies

- The *CI* build definition has been created and run

1. On the **Builds** page of the **Build & Release** hub, locate and edit the **CI** build definition.
2. Select the **Run All Unit Tests** task.
3. Clear the **Run only impacted tests** option.
4. In the **Test filter criteria** field, enter **TestCategory=CI**.

By filtering on this category, only those tests decorated with the "CI" trait will be run.

5. **Save** the build definition with an appropriate comment.

Identify and Run Only the “CI” Tests

👤👤👤 | 5 MIN

In this activity, your team will self-organize, return to Visual Studio, identify around 10 unit tests as “Continuous Integration” unit tests, decorate them as such using attributes, and then push the changes back to VSTS.

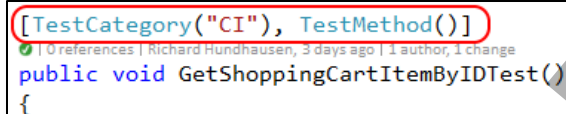
Dependencies

- The *CI* build definition has been created and contains a *TestCategory=CI* filter

1. Return to **Visual Studio** and ensure that you are in the **master** branch of the default repository.
2. Identify a unit test that, in your team’s opinion, should be part of the “Continuous Integration” test suite.

Make sure the test is not an inconclusive or failing test.

3. Add the **TestCategory("CI")**, attribute in front of the [TestMethod()] attribute so it looks like this:



```
[TestCategory("CI"), TestMethod()]  
public void GetShoppingCartItemByIDTest()  
{
```

0 references | Richard Hundhausen, 3 days ago | 1 author, 1 change

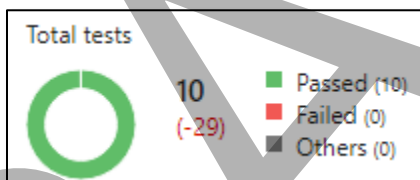
Note: You can also put a *TestCategory* attribute in front of a *TestClass* attribute if you want the category to apply to *all* of the tests in that class. This is a new feature in MSTest V2.

4. Make sure the solution builds and all the tests run without error.

Tip: Enter *Trait:"CI"* into the Search field at the top of the Team Explorer to see just those “CI” tests.

5. Repeat the above steps until around 10 unit tests have been identified.
6. **Commit** and **Sync** changes back to Visual Studio Team Services.

Each push should trigger the CI build to run. Watch the build and see the smaller number of tests emerge.



Continuous Integration +



Continuous Integration +

- Continuous Integration typically refers to
 - Integrating code from one or more developers/branches
 - Ensuring that the integrated code passes unit tests
- Continuous Delivery/DevOps, mandates that we also
 - Run on production-like environments
 - Passing acceptance tests



Acceptance Tests

- Verifies that the team is *building the right thing*
- Created and run by anyone on the development team
 - Traditionally a responsibility of the Quality Assurance (QA) or Business Analyst (BA) role
- Can be automated or manual
 - For Continuous Delivery, they must be automated
- There are many types of acceptance tests:
 - Functional tests
 - Performance tests
 - Load tests
 - Security tests

Accentient

Visual Studio Acceptance Tests

- Any of the Visual Studio test types can be used for automated acceptance testing
 - Some require Enterprise edition
- UI testing should be your last resort for acceptance testing
- Tip: Use Selenium and Appium rather than Coded UI tests for functional/acceptance UI testing
 - These are Microsoft's own words



Visit <http://bit.ly/2CjK1I1> for more information

Accentient

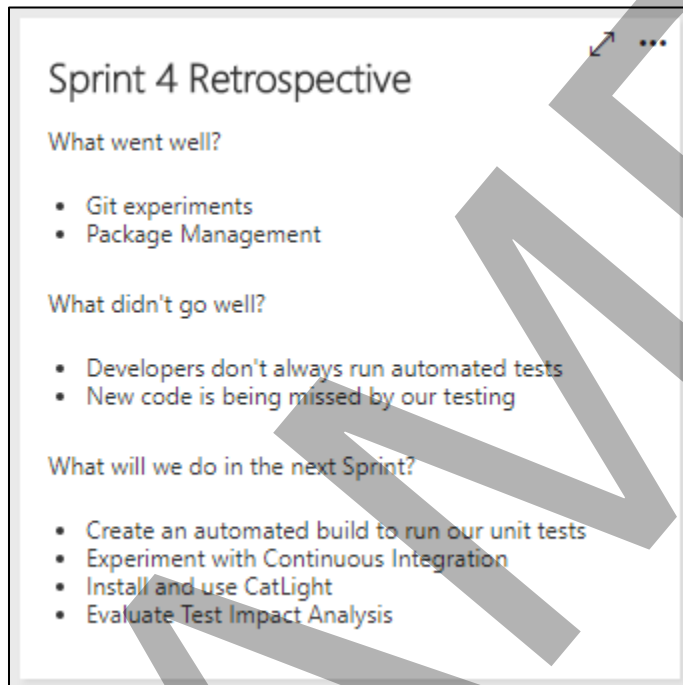
In this activity, your team will self-organize, adding another widget to the Kaizen dashboard, and listing the outputs of your team's last Sprint Retrospective.

1. Navigate to your team project's **Kaizen** dashboard.
2. Add a **Markdown** widget, and paste the contents from **C:\Course\Activities\Sprint 4\SprintRetrospective.txt**.

For more information on using markdown, visit <http://bit.ly/2d026Nh>.

3. If necessary, configure the size of the widget to properly display the markdown.

Here is an example:



Module Review

What have we accomplished and learned?

- Continuous Integration = automated building and testing
- Don't be afraid to commit changes frequently
 - Broken builds and failed tests are a type of feedback
 - The more often you integrate (commit), the less pain it will be
- Code coverage tells you where to invest in more testing
- Test Impact Analysis and test filters can shorten the feedback loop even more by only running the important tests

