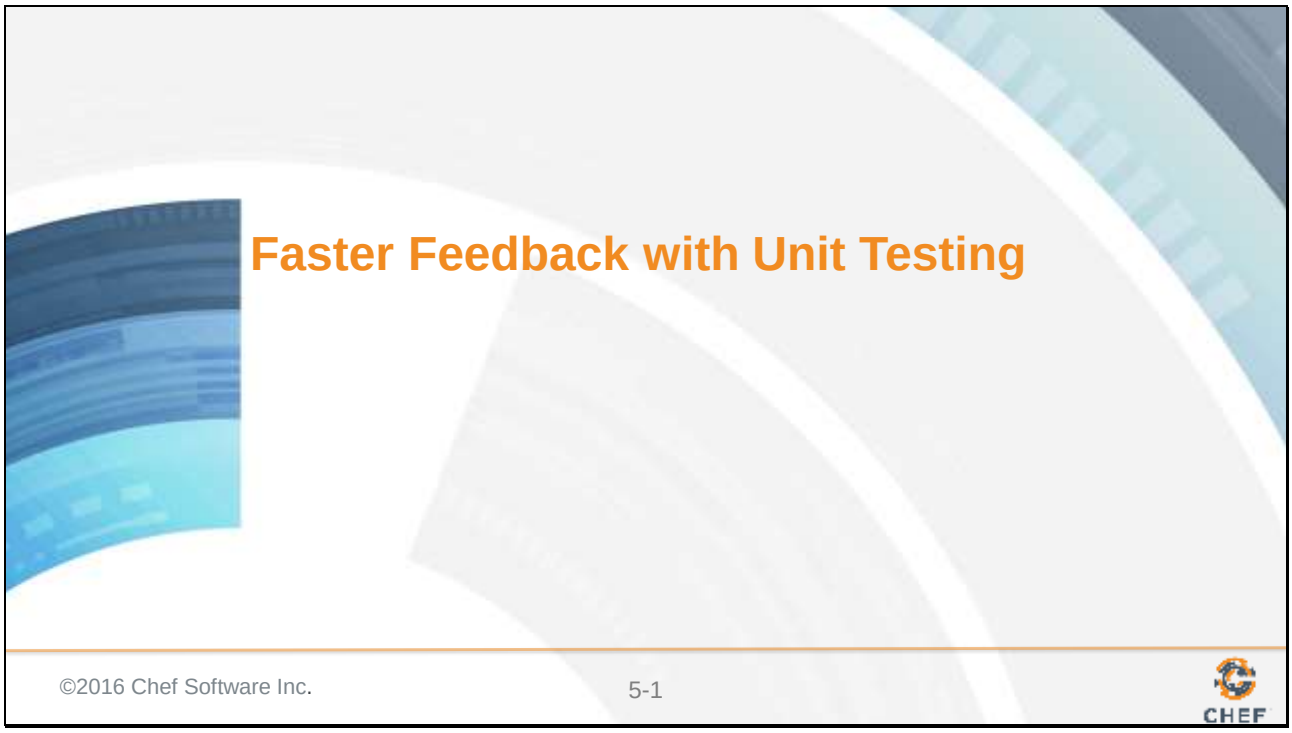
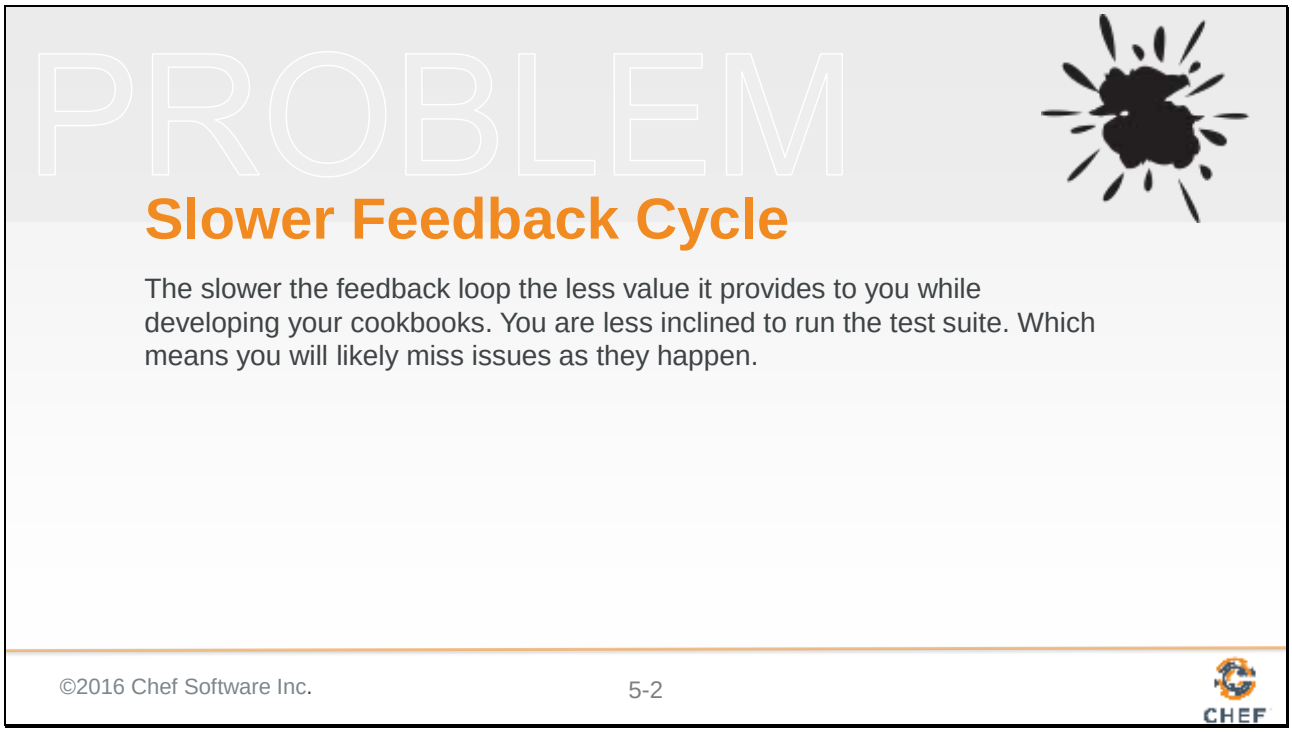
5: Faster Feedback with Unit Testing



If you are planning on adopting Test Driven Development and use it to validate most if not all all of the changes that you make to a cookbook you now have to are welcoming into your workflow the interruption of running the tests. Testing provides value as it validates the work that you accomplish but it is still an interruption.

Slide 2



Interruptions are not conducive to helping you building a flow. To help reduce the interruptive nature of testing we can look at ways to decrease the amount of time you have to wait to receive the feedback from the tests. A faster feedback cycle will increase your likelihood of seeking that feedback again for smaller sets of changes. Slower feedback cycles will increase your likelihood of seeking feedback less often. Causing you create larger sets of changes which has the chance of masking potential issues.

Slide 3

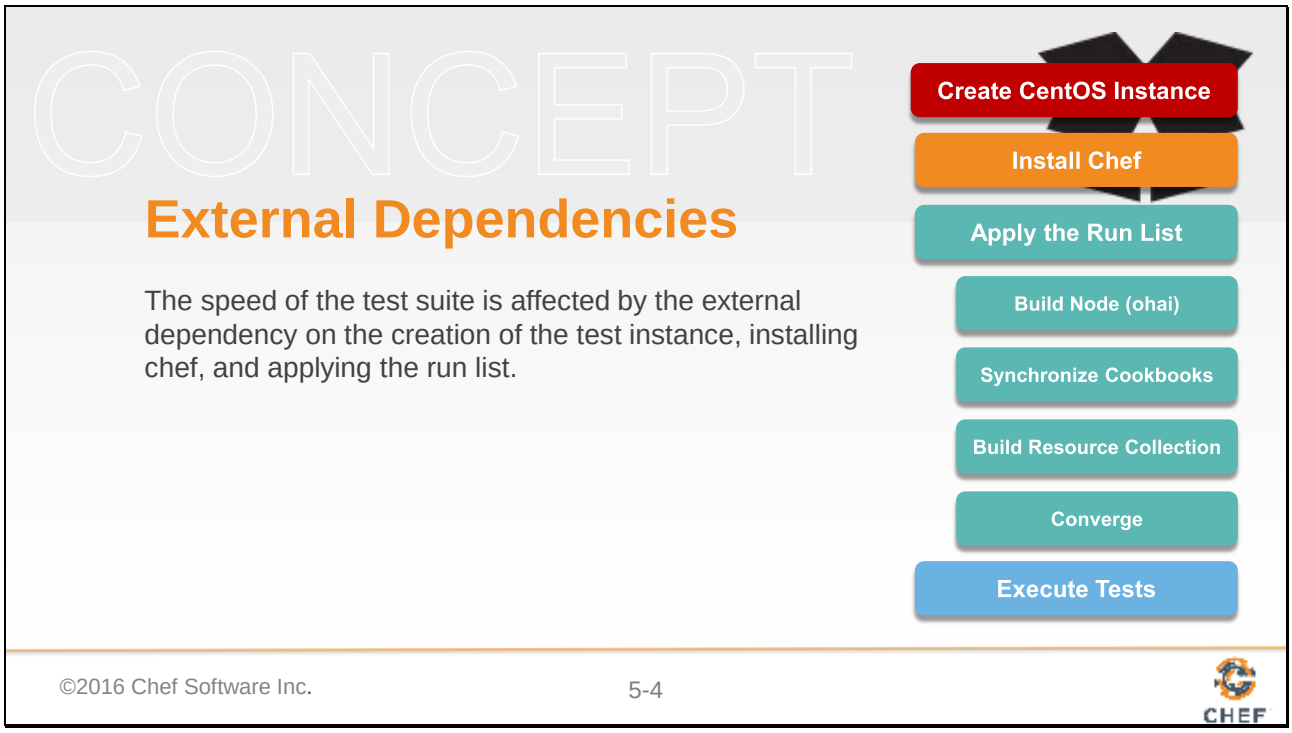
Objectives

After completing this module, you should be able to:

- Explain the importance and limitations of unit testing
- Write and execute a unit test

In this module you will learn the importance and limitations of unit testing as you write and execute unit tests to help increase the rate at which you receive feedback.

Slide 4

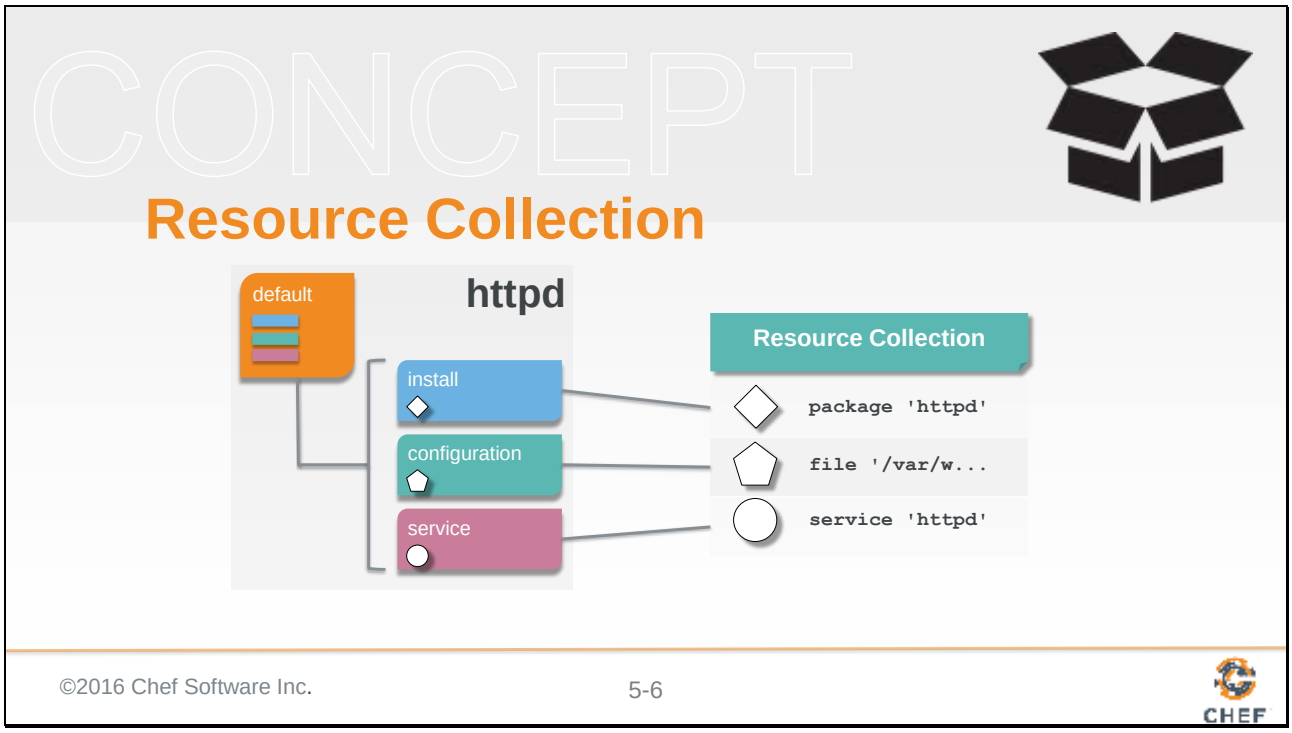


The reason that the feedback cycle takes as long as it does with Test Kitchen is because of the external requirements. Creating the test instance, installing chef, and then applying the run list provide real value because we are able to see the recipe being applied to a virtual instance. However, all these external dependencies incur a time cost as we wait for the network to download images or packages, the test instance's processor to calculate keys or data, or the file-system to create files and folders.

Slide 5

When we mutated our code and executed the test suite we created issues with the resources that we defined and recipes that we included. These changes affected the resources that were applied to the system by omitting resources from the 'Resource Collection'. If we were able to remove the external dependencies and focus on the state of the Resource Collection we would be able to determine if there were problems with the recipes we wrote without the need of any of those external dependencies.

Slide 6

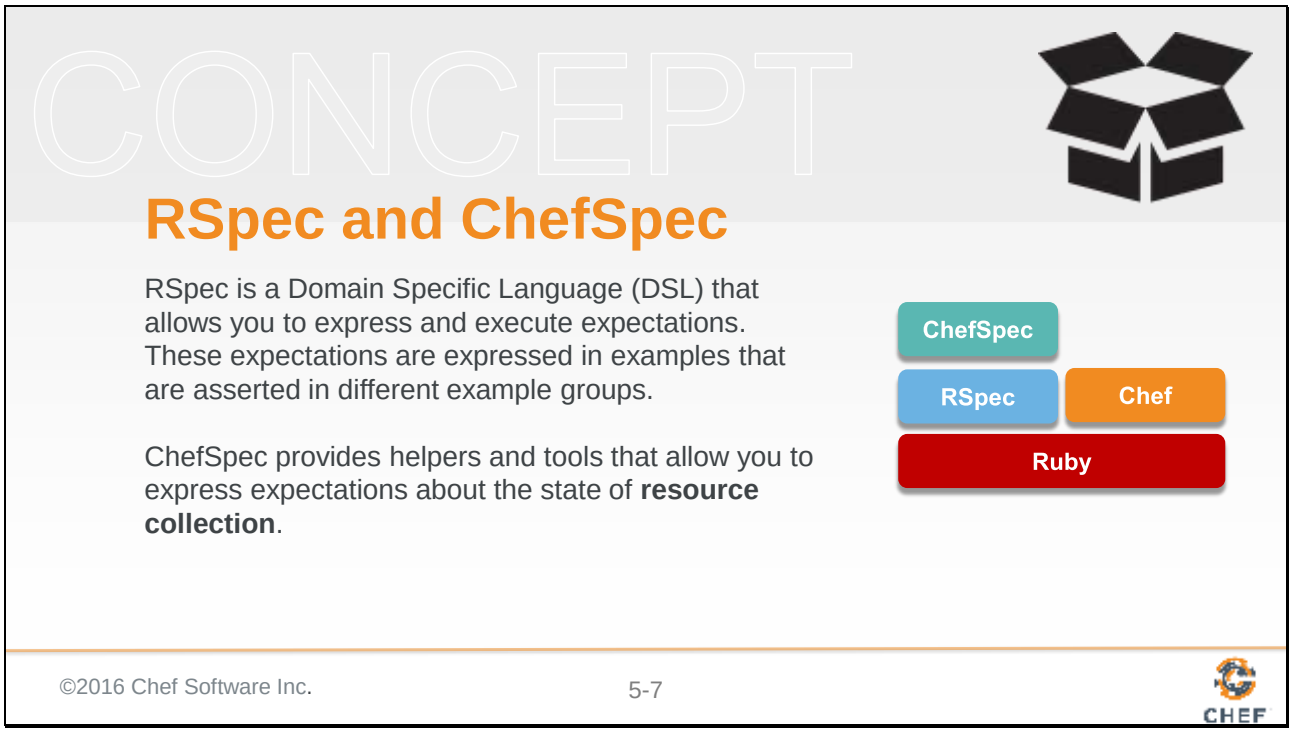


But first let's talk more about the 'Resource Collection' ...

After a cookbook and its recipes have been synchronized the majority of the cookbook content is loaded into memory by 'chef-client'. The recipes defined on the run list are evaluated during this time and the resources found within the recipes and any included recipes, are added to a resource collection. They are not immediately executed like one might assume.

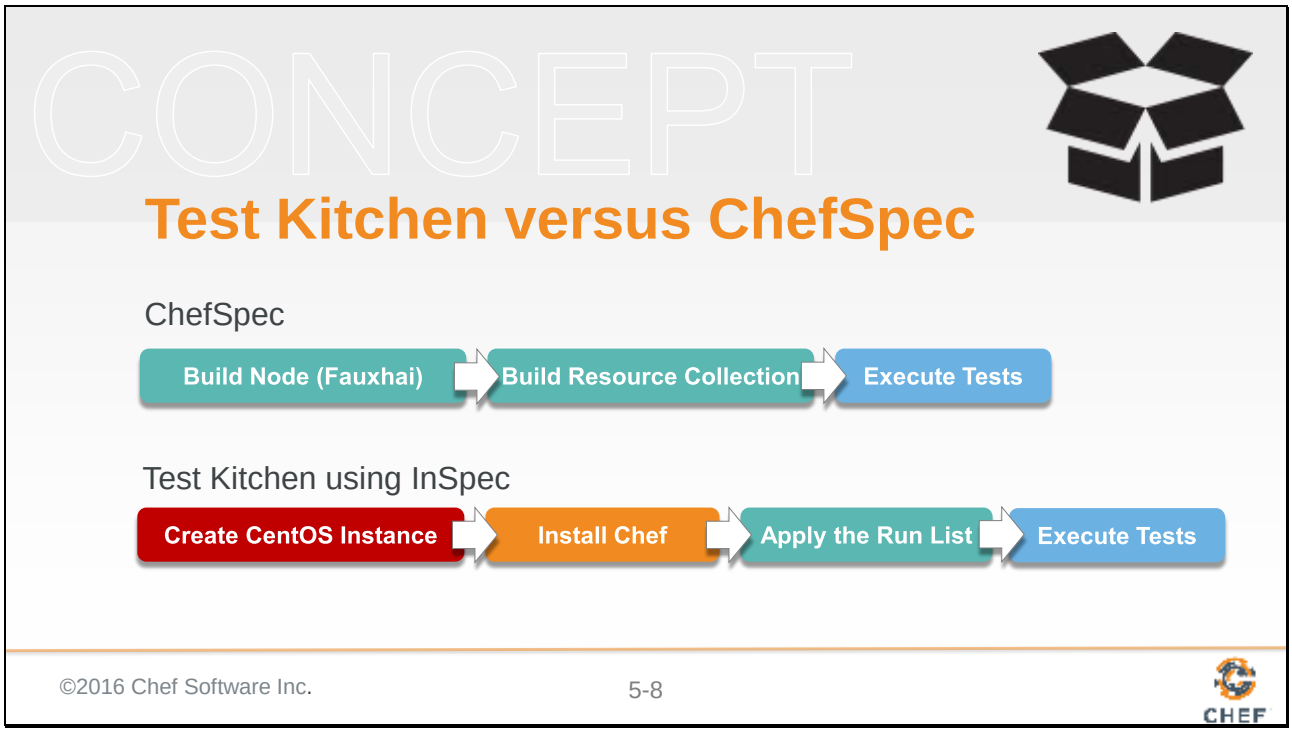
The 'Resource Collection' is almost like a to-do list for the node. It contains the list of all the resources, in order, that need to be accomplished to bring the instance into the desired state. Later, in the converge step, the resources defined in the Resource Collection are executed and perform their various forms of test-and-repair to bring the instance into the desired state.

Slide 7



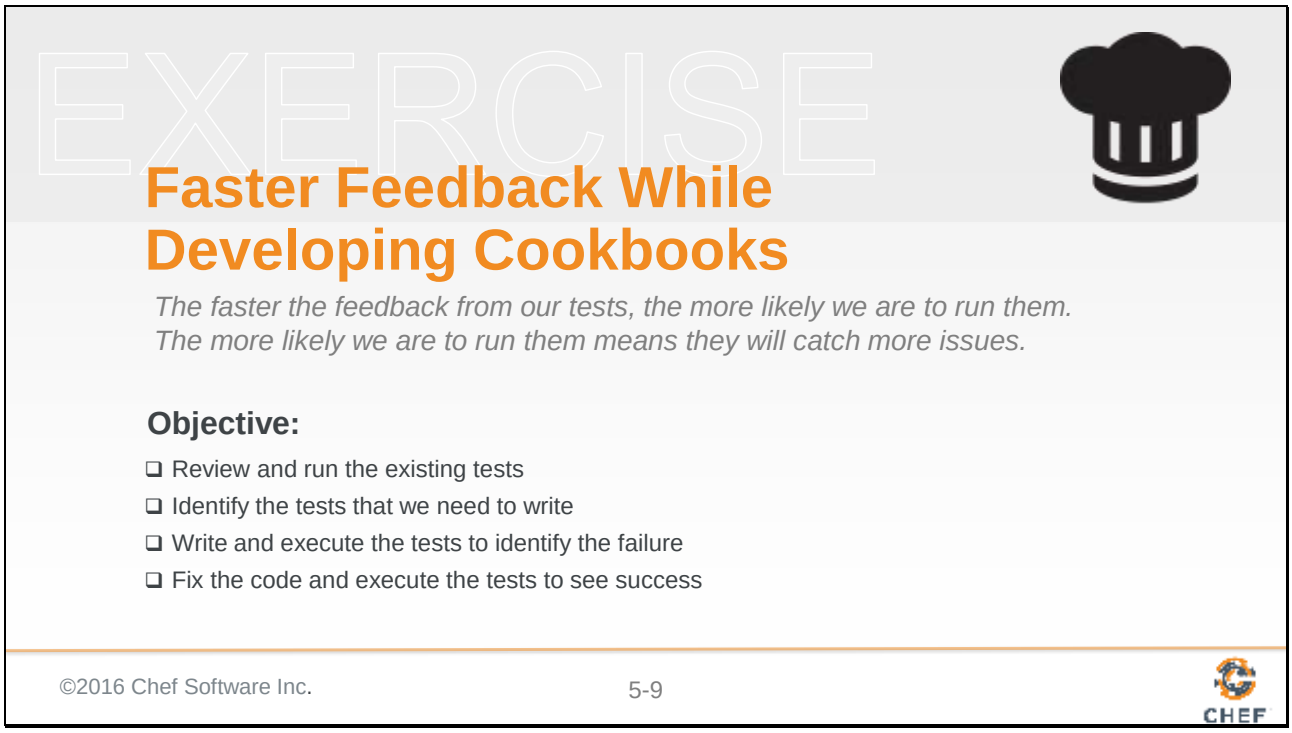
ChefSpec provides a method for us to create an in-memory execution of applying the run list, building the resource collection, and then setting up expectations about the state of the resource collection. ChefSpec, similar to InSpec is built on top of RSpec; relying on it to provide the core framework and language. The benefit to us is that a lot of the same language constructs are employed.

Slide 8



Verifying the resource collection with ChefSpec requires far fewer external dependencies and that allows us to get feedback faster but at the cost of not applying the recipes we write against a test instance. This opens us up to situations were we could compose recipes and execute examples that are shown to work because they were correctly added to the resource collection but fail when it comes time for the recipes to apply the desired state against a test instance.

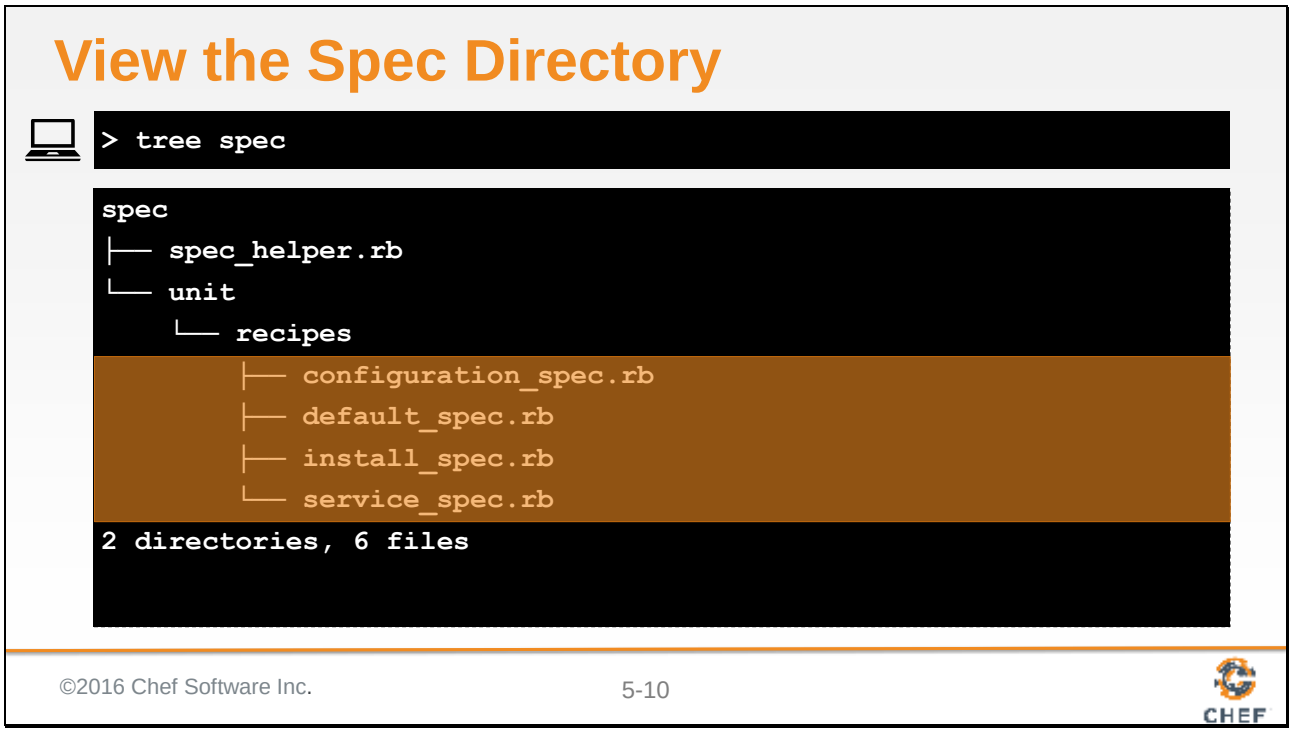
Slide 9



We have the integration test, the one defined in InSpec, executed through Test Kitchen to ensure the recipes we write behave as we expect on the test instances we define. The benefit of writing tests focused around the Resource Collection will allow us to gain feedback quickly and build a better development workflow.

This next group exercise we will review the existing ChefSpec specifications defined for us and how we can expand them to capture our additional expectations about the Resource Collection.

Slide 10



When generating recipe files we were also given a matching specification file in the 'spec/unit' directory. The ChefSpec defined specifications are all contained within this directory.

Slide 11

View the Test for the Default Recipe

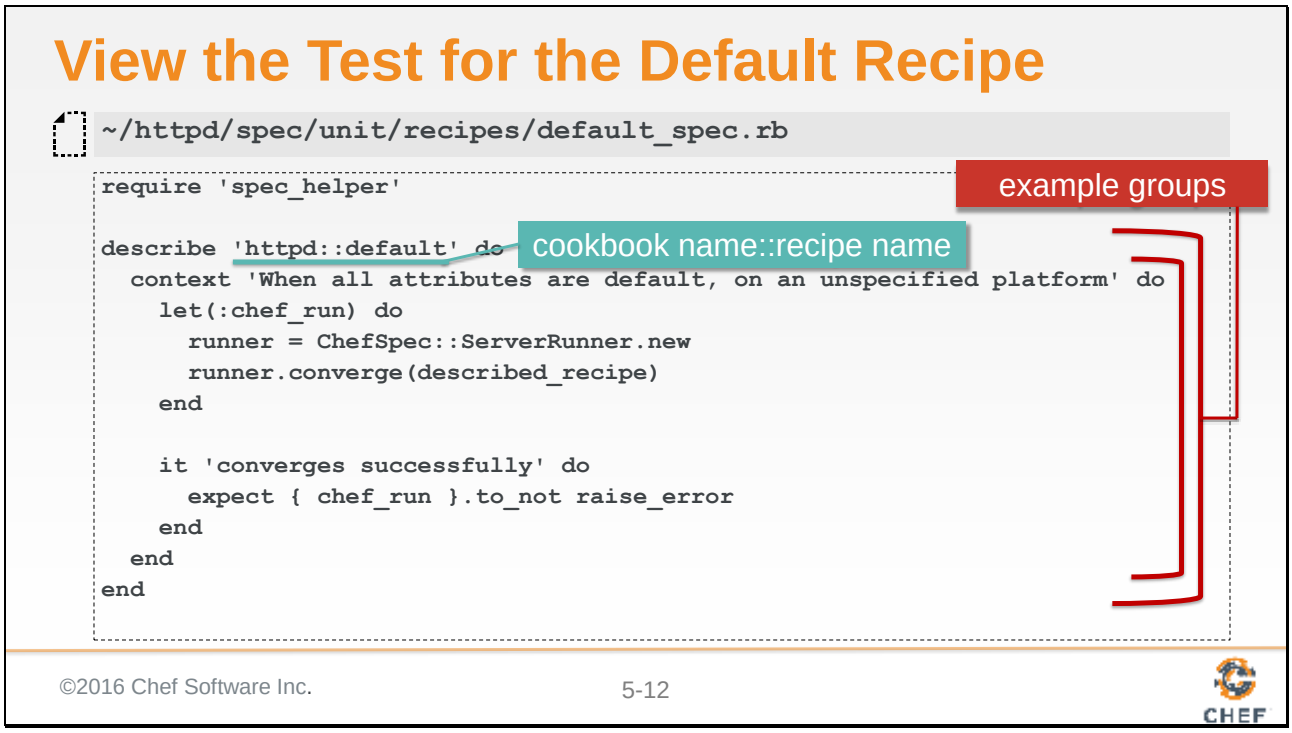
~/httpd/spec/unit/recipes/default_spec.rb

```
require 'spec_helper'

describe 'httpd::default' do
  context 'When all attributes are default, on an unspecified platform' do
    let(:chef_run) do
      runner = ChefSpec::ServerRunner.new
      runner.converge(described_recipe)
    end

    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end
  end
end
```


Open up the default specification file and lets read through and begin to understand the initial expectation that is automatically defined.

The expectations defined in this initially generated specification file should look a little familiar. This is because ChefSpec is built on Rspec. Similar to how InSpec is built. ChefSpec requires a little more setup as we are creating an in-memory execution.

Slide 12

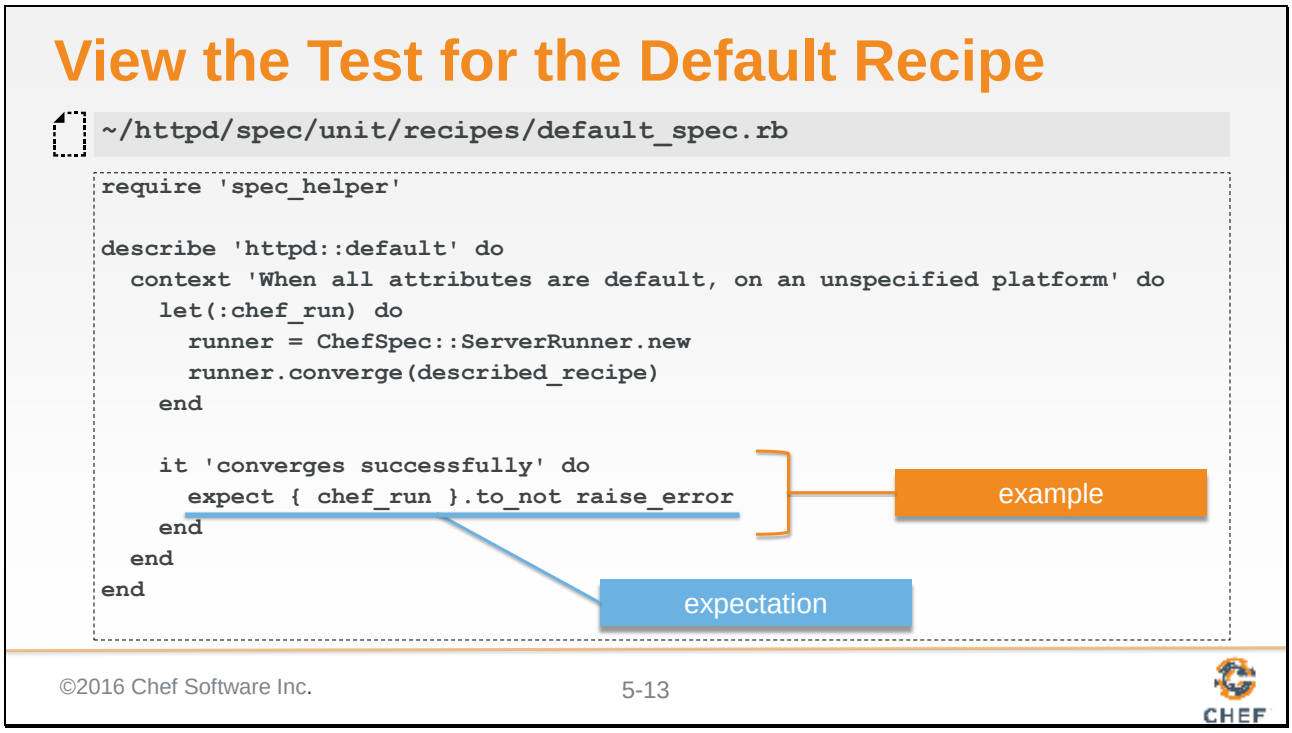


It is often common for specification files to share similar functionality. As your suite of examples grows you will often move common, shared expectations and helpers to a common file that is required here at the top of the file. This will load the contents of the 'spec_helper' file found within the root of the 'spec' directory.

ChefSpec employs RSpec's example groups to describe the cookbook's recipe. This is stating that the examples we defined within this outer example group all relate to the httpd cookbook's default recipe. Within this example group we see another context that is defined. This time using the method 'context'. 'context' and 'describe' are exactly same in almost every way. A lot of developers like to use context as it more clearly states that the example group is focused on a particular scenario. In this instance the particular scenario we are going to be specifying examples in a scenario where all the attributes are default on an unspecified platform.

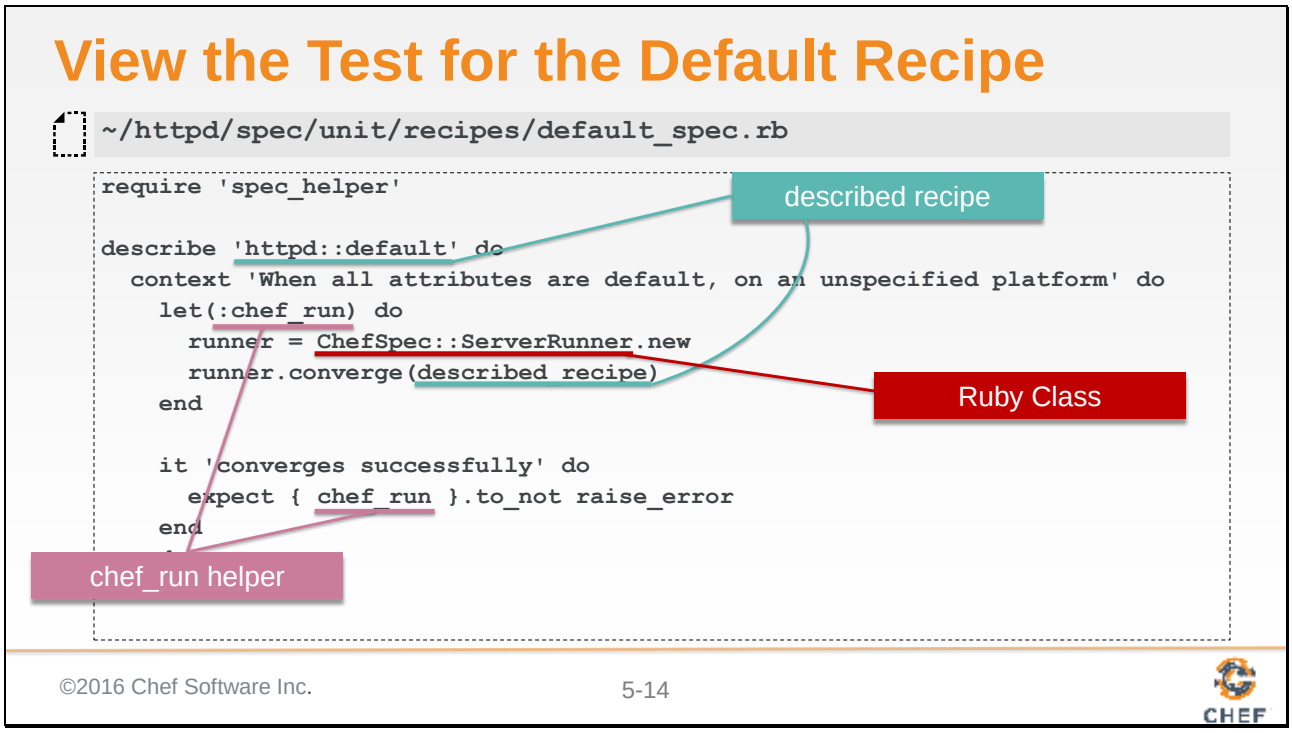
Instructor Note: 'describe' and 'context' are almost completely interchangeable with one exception. 'context' cannot be used as the outermost example group.

Slide 13



Within the inner context we finally set the stage for us to define our examples with their expectations. There is a single example defined and that is stating that when the chef run evaluates and creates the resource collection it should do so without raising an error. A situation that might raise an error is if we included a recipe that does not exist or if we were to use a resources type that does not exist.

Slide 14



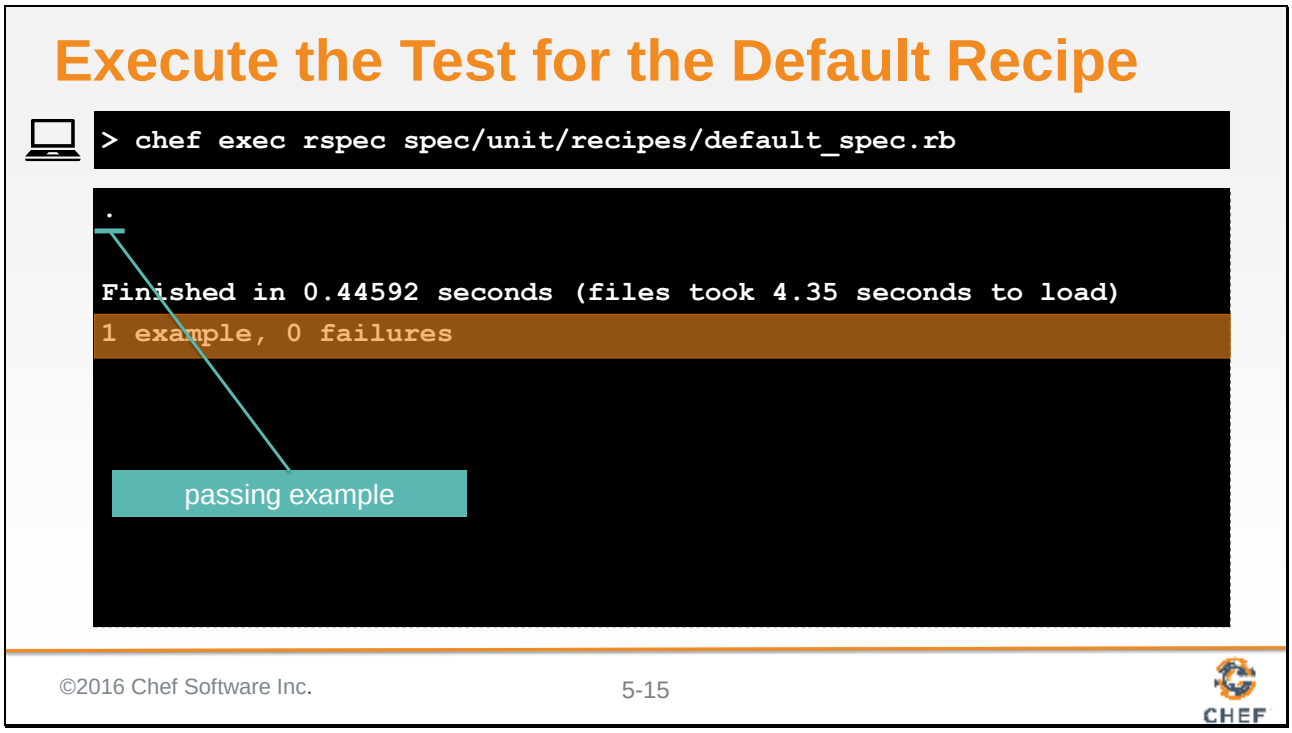
The 'chef_run' helper there is being provided by the 'let' defined above the example within the same context. Defining the 'chef_run' in the 'let' above is done with a Ruby Symbol. This is simply naming it so that it can be used within any of the examples in the current context and even sub-contexts. The helper is simply executing some code that sets up an in-memory chef-client run with a Chef Server.

The 'ServerRunner' is a class defined within the 'ChefSpec' namespace. All Ruby classes have the method 'new' which will return an object which is a new instance of that described class. The object is stored in a local variable, named 'runner', which immediately invokes a method 'converge' with a single parameter 'described_recipe'

The parameter 'described_recipe' refers to the recipe defined in the outermost describe. This is mostly for convenience so that we do not have to redefine the same String multiple times within the same specification file.

The goal of this single, boilerplate example is very simple: perform a chef-client run and ensure there are no errors. Now, let's execute this specification.

Slide 15

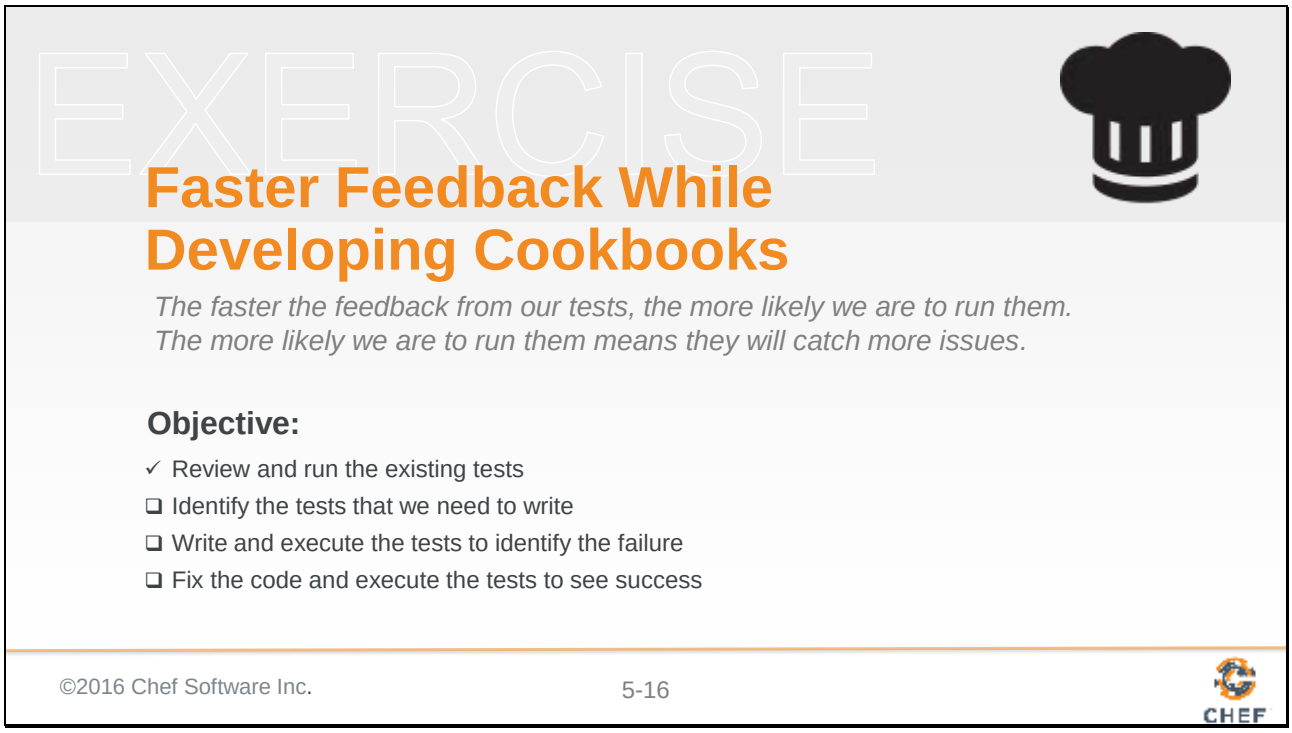


To execute the specification file defined you will need to run the command 'rspec'. The 'rspec' command was installed with the Chef Development Kit (ChefDK) on the workstation. It is contained in an additional folder of tools embedded within the ChefDK that are not added to the system path. This is because some Chef developers are Ruby developers and may already have a version of RSpec installed. Specifying the 'chef exec' as a prefix loads the context of all these embedded tools and allows them to be executed on the command-line.

The 'rspec' command accepts many parameters. The most important one is used here and that is specifying the file path to the specification we want to execute. When the command executes a summary of the executed examples will be displayed at the bottom. At this moment it looks like the one expectation completes successfully. The chef run completes without any errors.

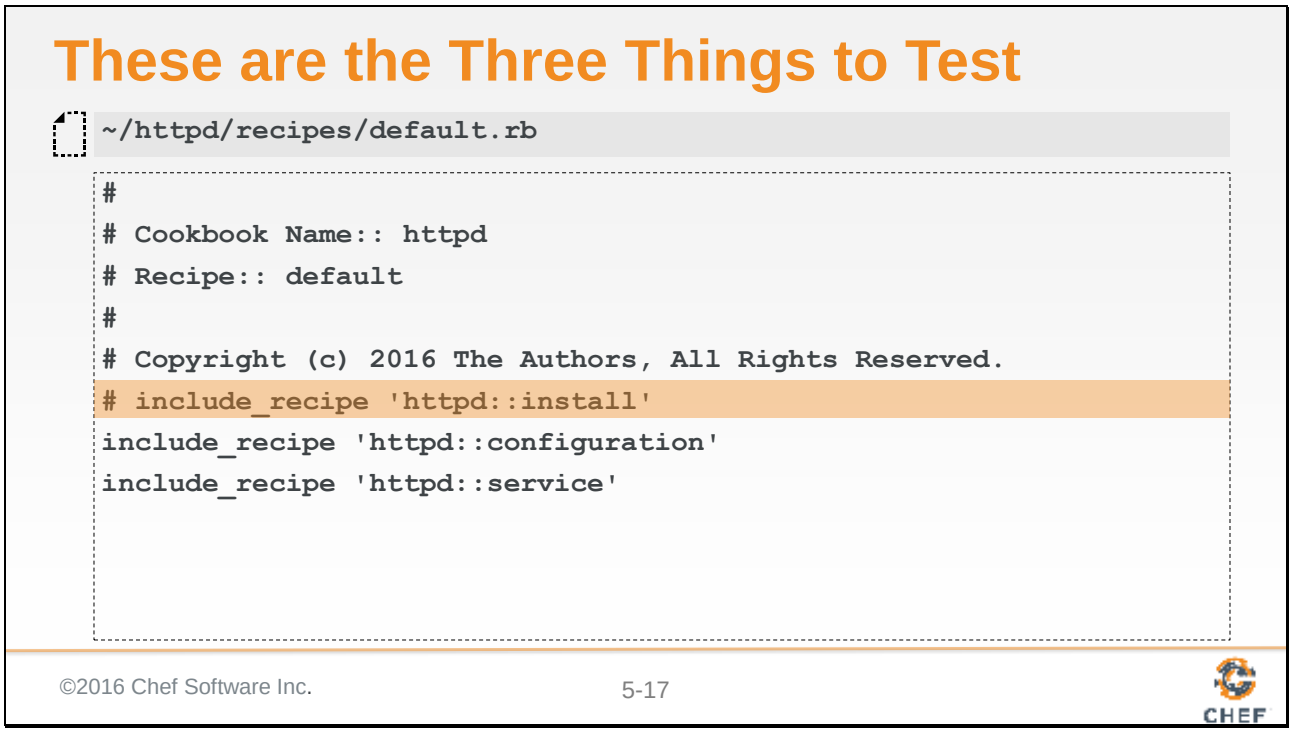
Instructor Note: On the workstations the learners do not need to prepend the rspec command with 'chef exec'. This is because 'rspec' and all the other tools embedded in the ChefDK have been added to the path. On a learner's local system this is likely not the case and so they will need to type this entire command with prefix.

Slide 16



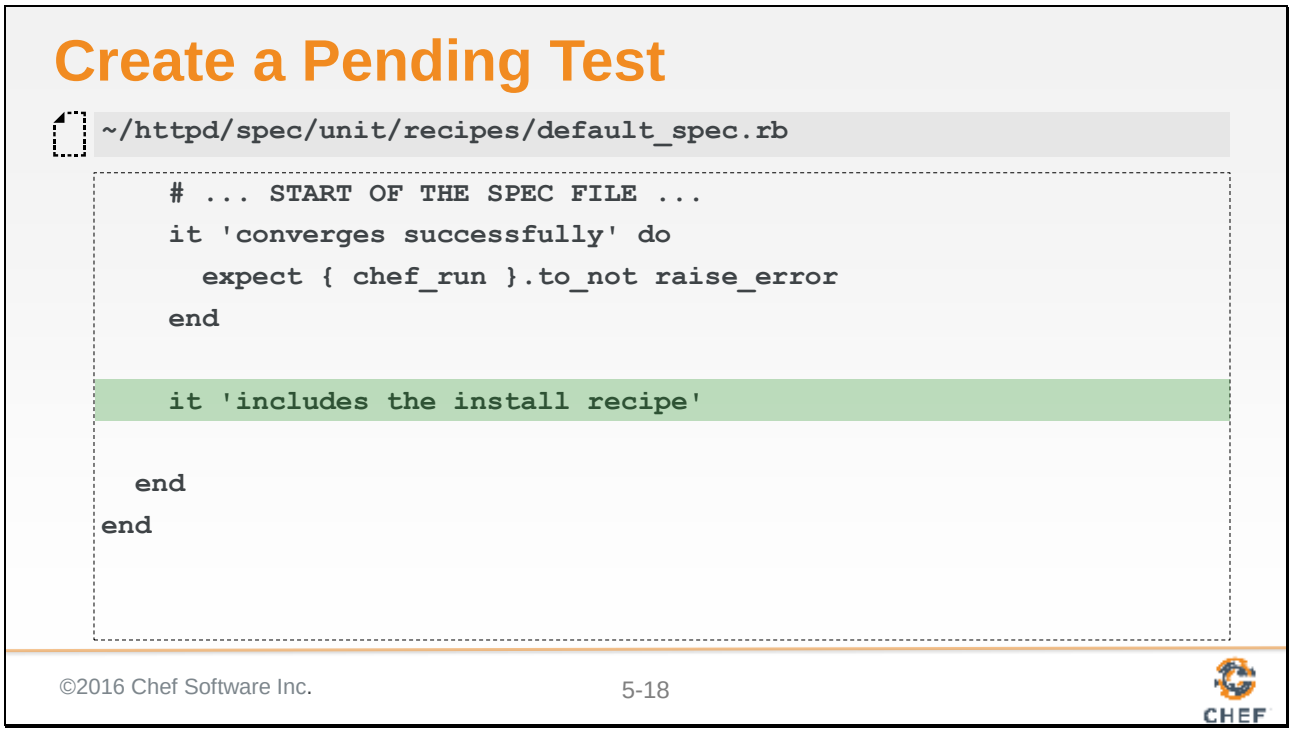
We have the language and the tool that will allow us to express our expectations. We now need to examine the recipe again to see what example or examples we want to define within the specification file.

Slide 17



Within the default recipe we commented out the line that included the install recipe from the httpd cookbook. This seems like an expectation that we want to define. When converging the default recipe we expect that the install recipe from the httpd cookbook would be included.

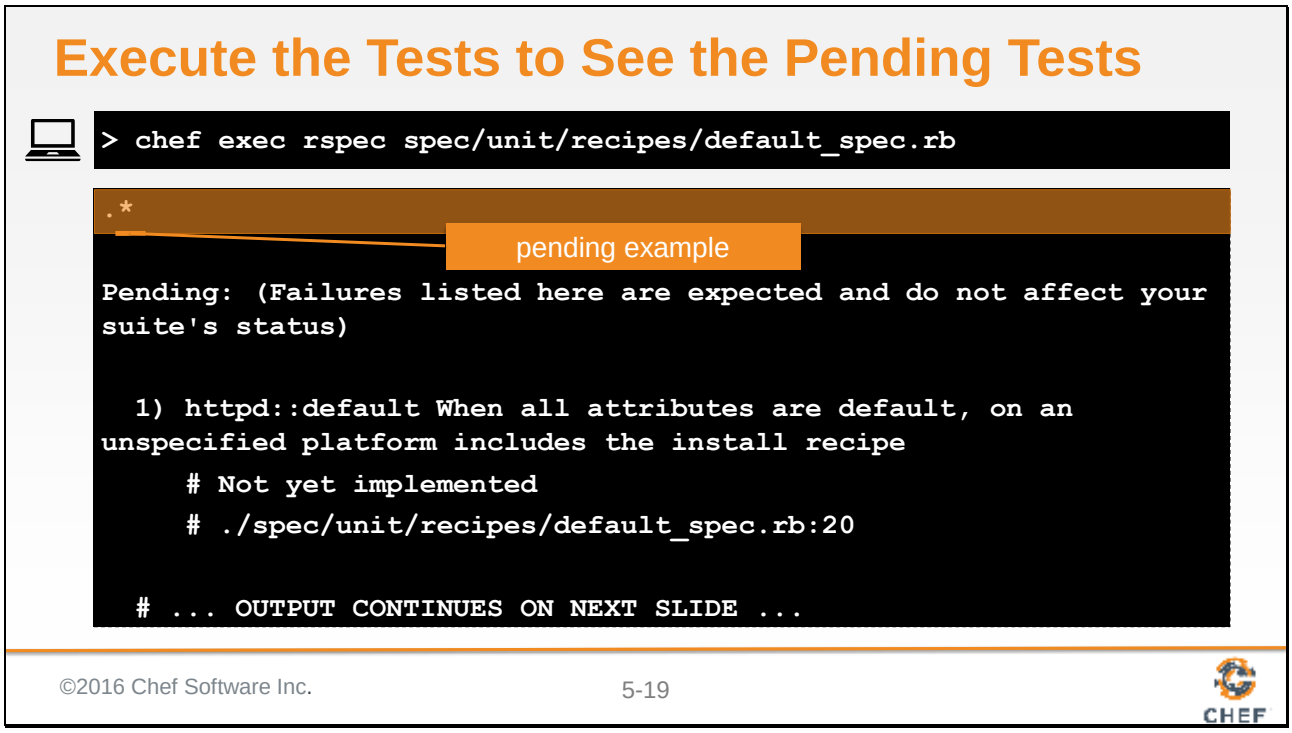
We do not yet know how to define this expectation but we know the work that we want to accomplish. So lets take this one step at a time then and first capture the description for the example even if we do not yet know how to express the expectation.



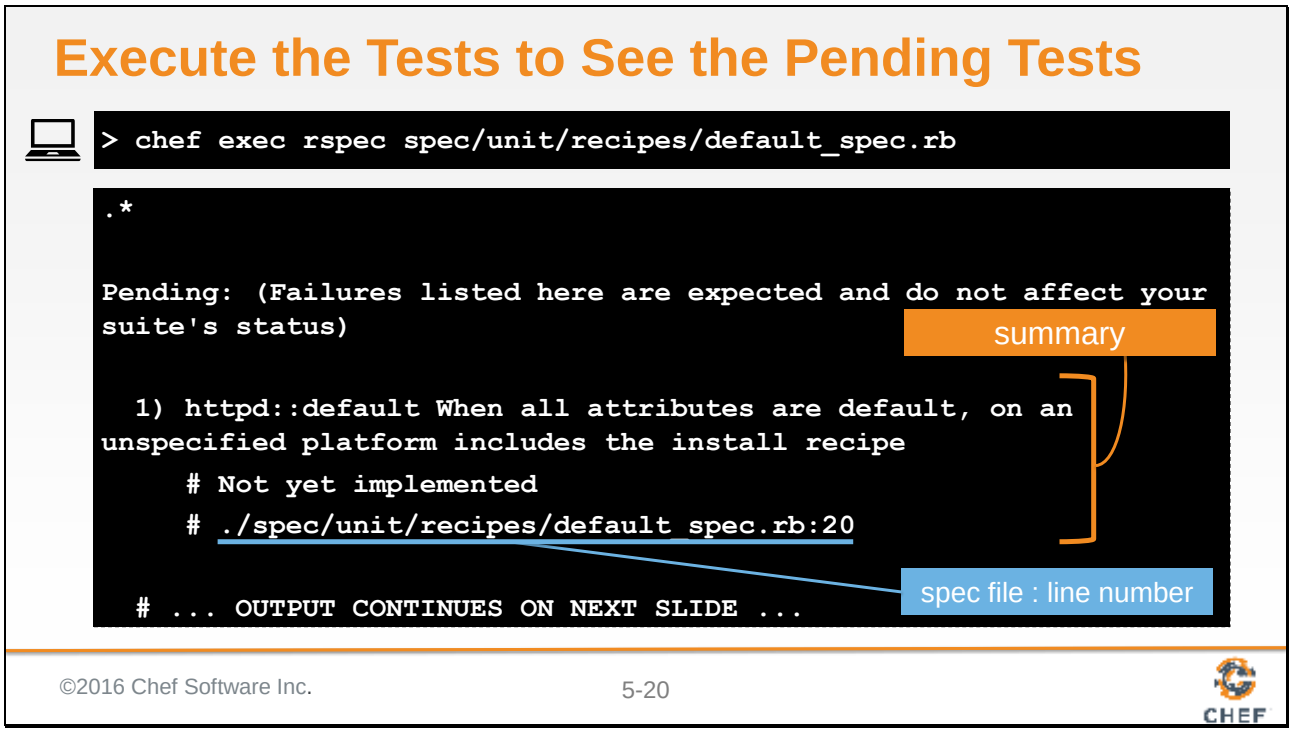
Returning to the specification we can describe the example that we want to create without having to know how to define the expectation by defining an example without the block. RSpec treats these examples without the associated block as a pending test.

This is an incredibly useful feature when you want to start expressing your examples. This allows you to quickly identify all the examples without getting mired in the details of their implementation.

Slide 19

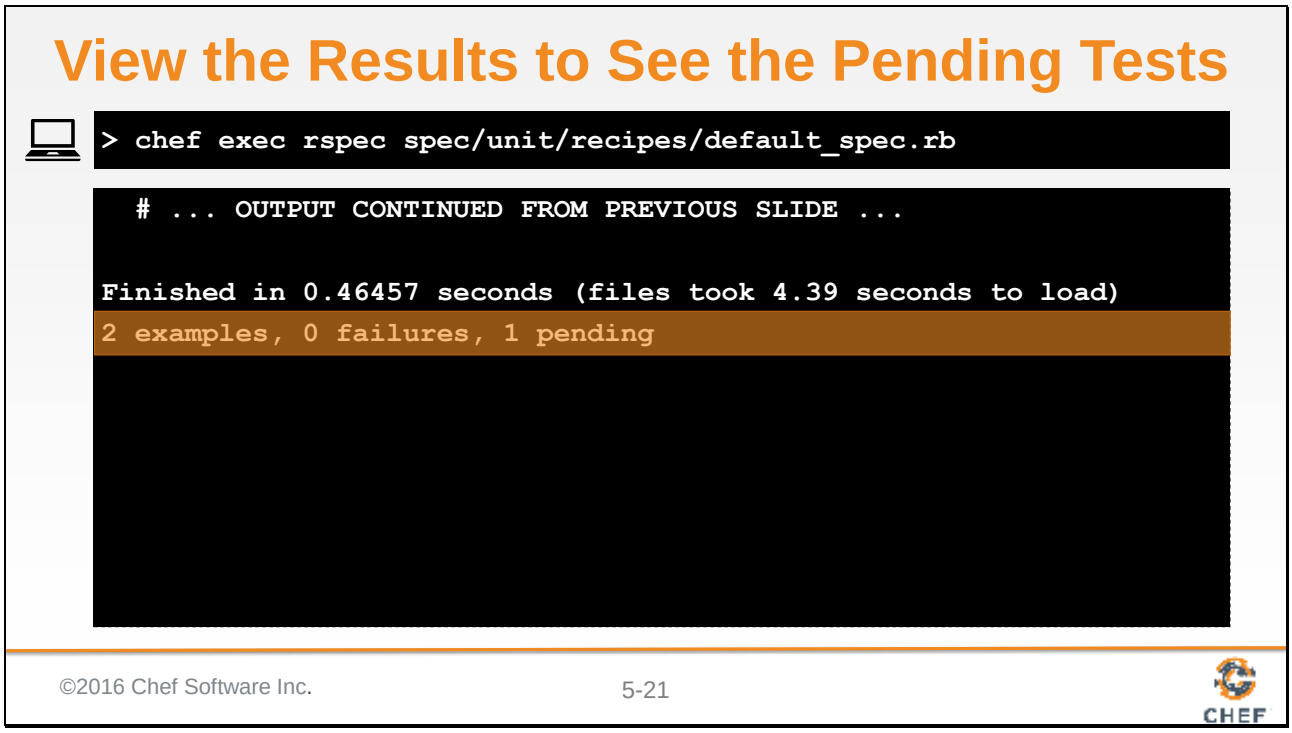


When executing 'rspec' again we should see the new pending example that we defined within the specification file.



RSpec's pending summary is similar to the failure summary. The pending examples are identified and then finally they are collected together in list. Each pending example will show the words you used in the description text in a single sentence. Below that it will state the example is not yet implemented and then finally display the file path and line number of where it can be found.

Slide 21



The summary will now display that an additional example has been added and it will be reported as being set to pending.

Slide 22

EXERCISE

Faster Feedback While Developing Cookbooks

The faster the feedback from our tests, the more likely we are to run them.
The more likely we are to run them means they will catch more issues.

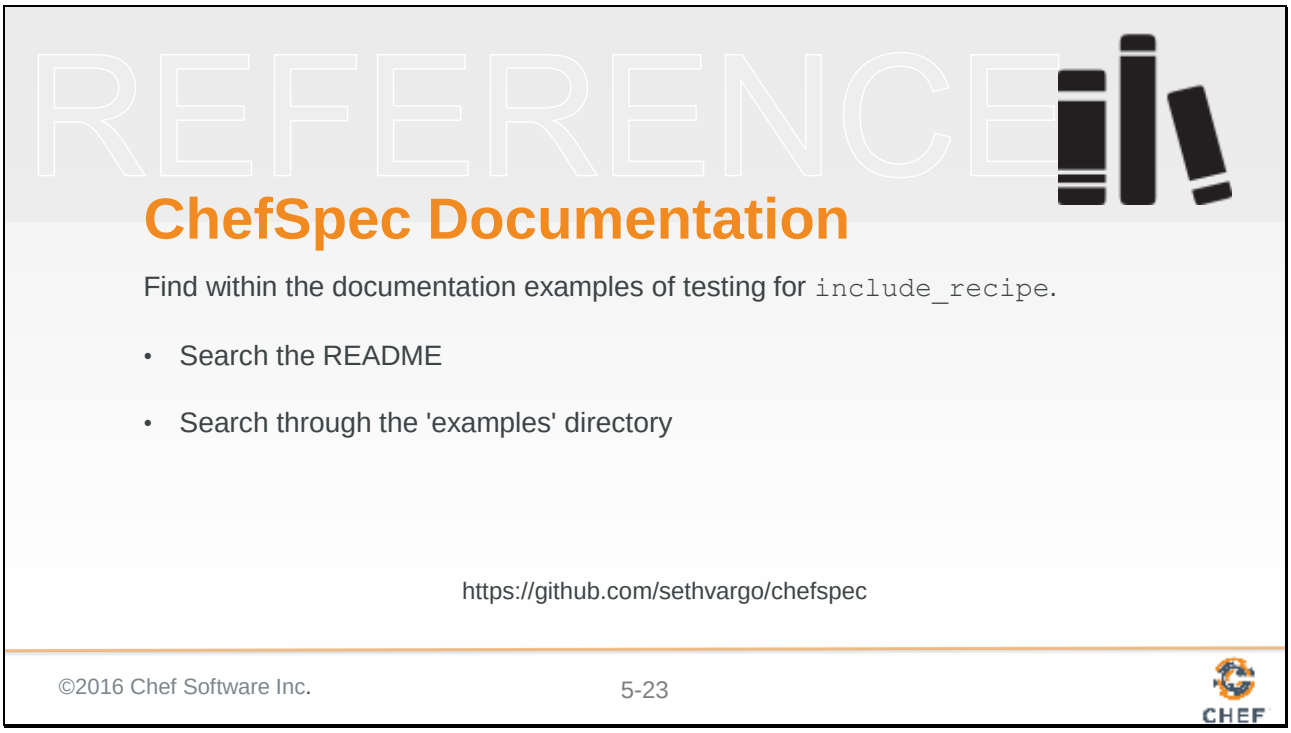
Objective:

- ✓ Review and run the existing tests
- ✓ Identify the tests that we need to write
- ❑ Write and execute the tests to identify the failure
- ❑ Fix the code and execute the tests to see success

CHEF

Now that we have defined the pending example, setting up the work for ourselves, it is time to learn how to express the expectation.

Slide 23



To understand how to express an expectation we need to go to the documentation. The ChefSpec README provides a wealth of examples in the README. In the past an 'include_recipe' example has been one of the many examples shared in the README. Use the search feature of your browser to find it within the document.

If it is not there, the ChefSpec project has a top-level folder named 'examples' which contains examples for nearly every feature that ChefSpec is able to define expectations. Searching through there you will find a folder titled 'include_recipe', within it should a folder the shows the recipes and the matching specifications.

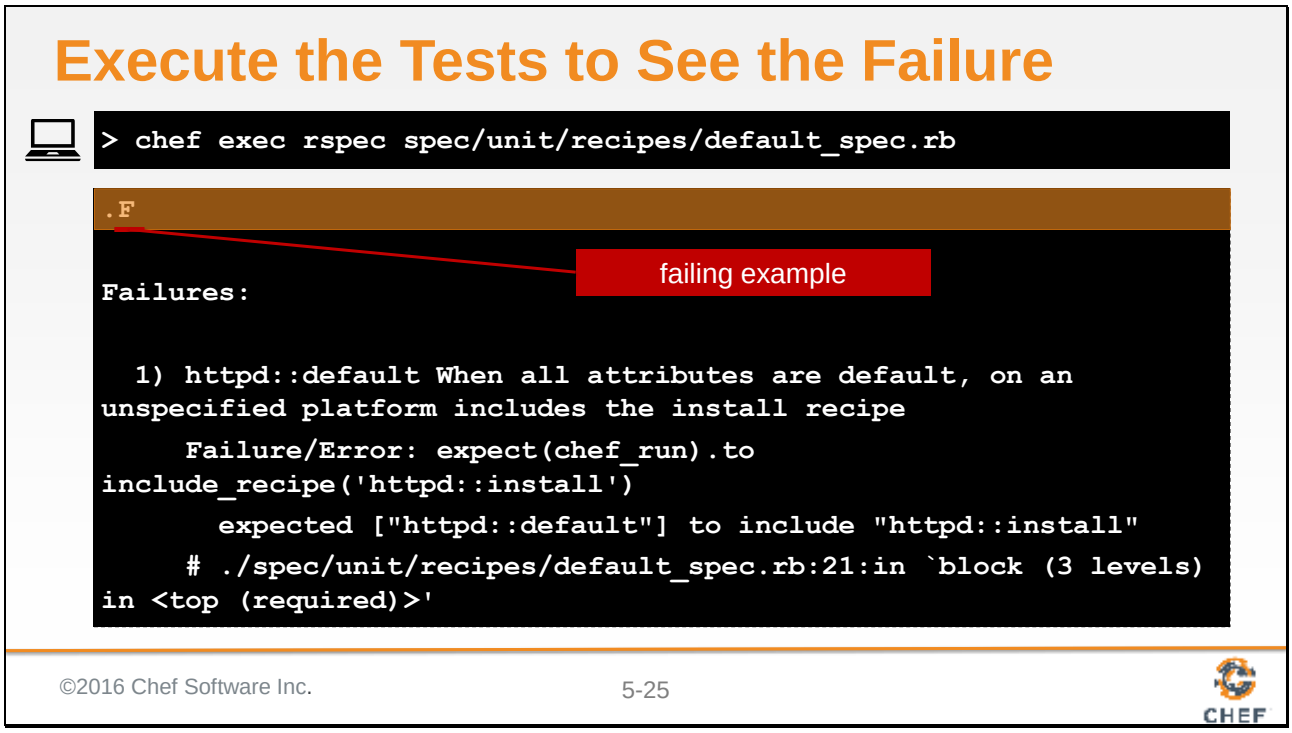
Write the Test that Verifies the Include Recipe

~/httpd/spec/unit/recipes/default_spec.rb

```
# ... START OF THE SPEC FILE ...
    it 'converges successfully' do
      expect { chef_run }.to_not raise_error
    end

    it 'includes the install recipe' do
      expect(chef_run).to include_recipe('httpd::install')
    end

  end
end
```


Returning to the specification file we now need to expand the example to include the expectation we want to write. To do that we add a 'do' to the end of the example. We move to the next line, indent two spaces and then define the following expectation. The expectation uses a natural language way of expressing the expectation. Here we are expressing the expectation that the 'chef_run' includes the recipe with the specified name.

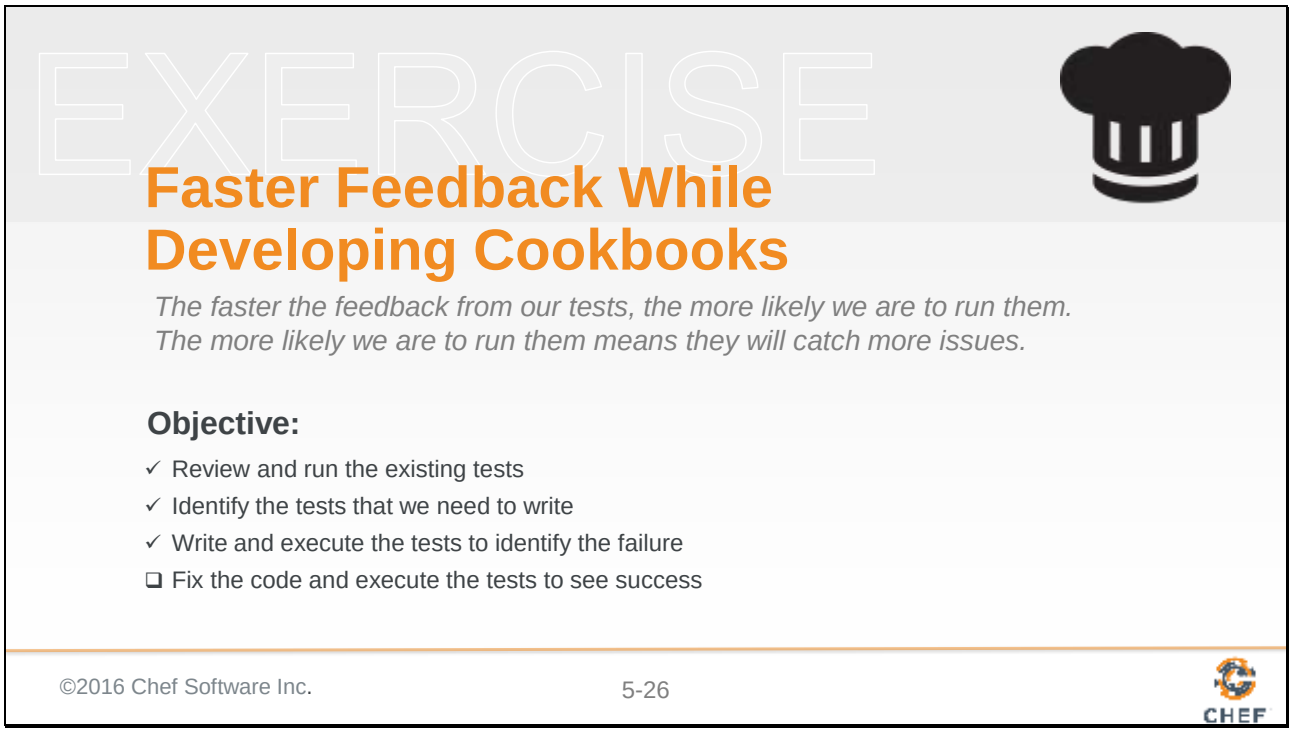
Slide 25



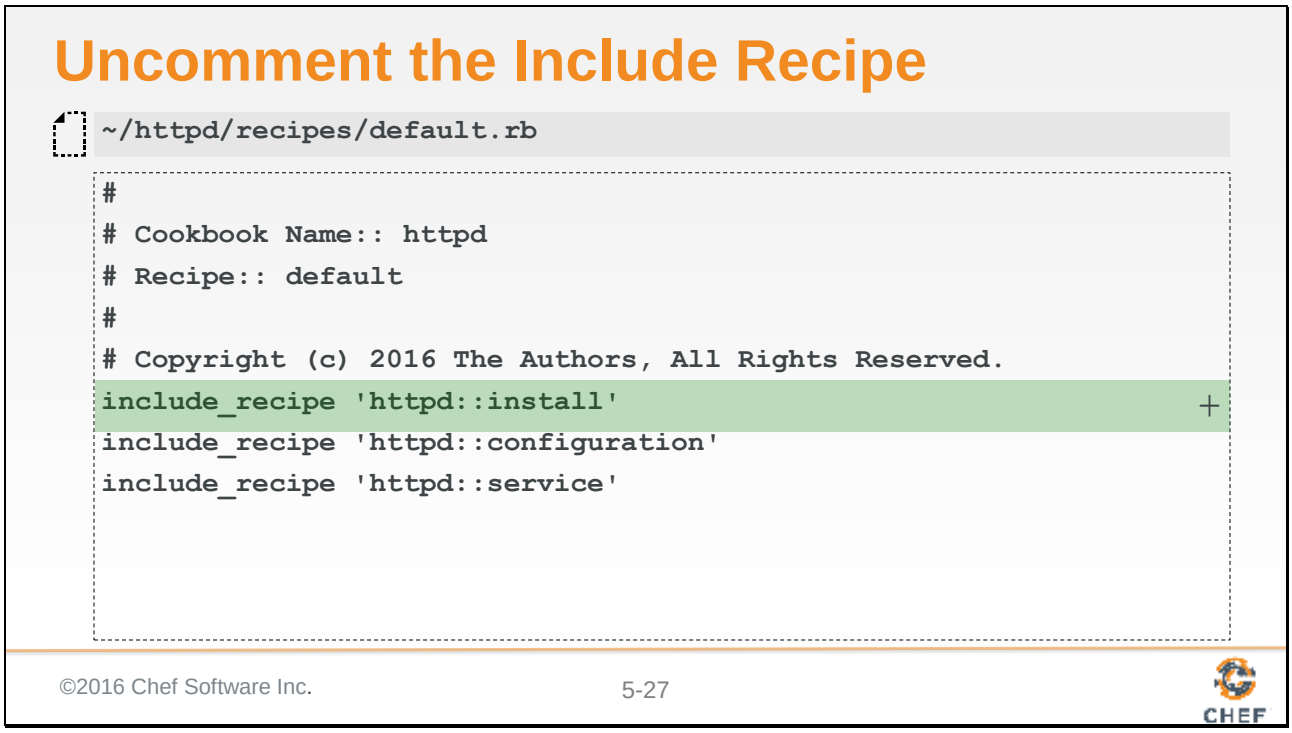
With the example defined with the expectation when we execute 'rspec' we see the failure that eluded us we ran 'kitchen converge & verify' on an existing very quickly.

The failure summary here is similar to the failure summary return by RSpec when employed by Test Kitchen. The example is displayed, the expectation is expressed, the failure to meet expectation and file name and line number within the file where to find the expectation.

Slide 26

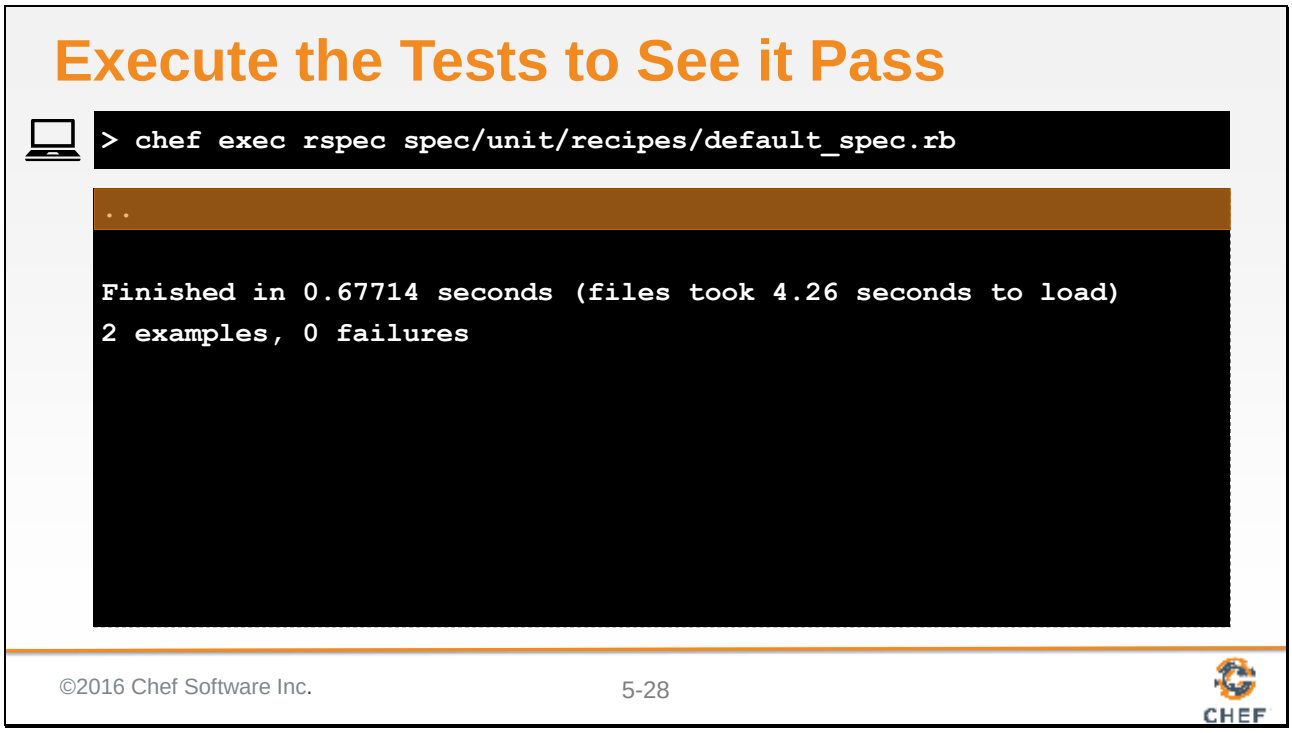


Now that we have a failing test it is time to fix the problem.



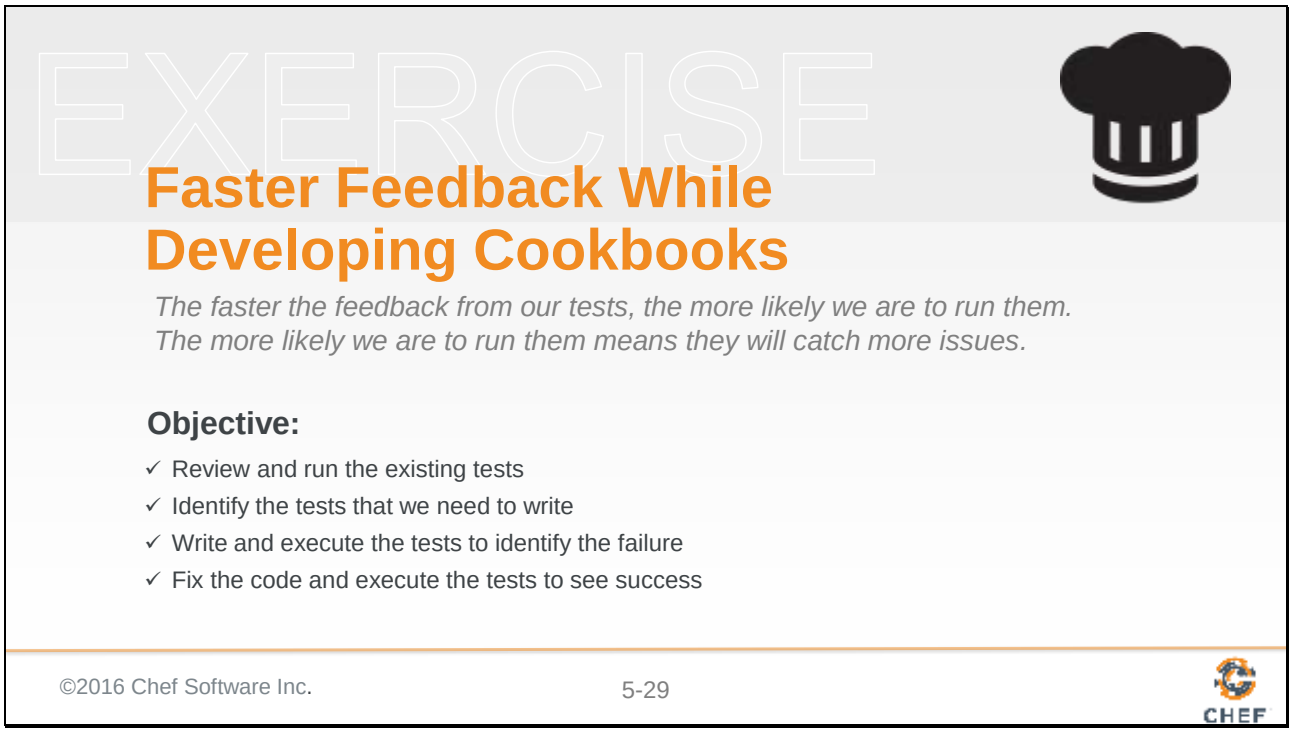
Returning to the default recipe it is time to restore the code that we previously commented out.

Slide 28



Executing 'rspec' one more time should show the previous failing example now as a passing example.

Slide 29



Now we can confidently state that the default recipe includes the install recipe and we can receive this verification in a faster feedback cycle then we saw with running 'kitchen test'.

Mutation testing is not Test Driven Development (TDD) but the act that we performed was fairly close. This is a tactic that is useful when you are writing expectations for already defined recipes for existing cookbooks or when it feels near impossible to start with the tests first. This process does one of the important aspects of TDD which is ensure the expectations we set correctly capture the state of the code.

Slide 30

LAB

Continue with Mutation Testing

- ❑ Comment out the next line in the httpd cookbook's default recipe
- ❑ Write the example with expectation that will generate a failure
- ❑ Verify that one example generates a failure
- ❑ Restore the code in the recipe
- ❑ Verify that all examples pass

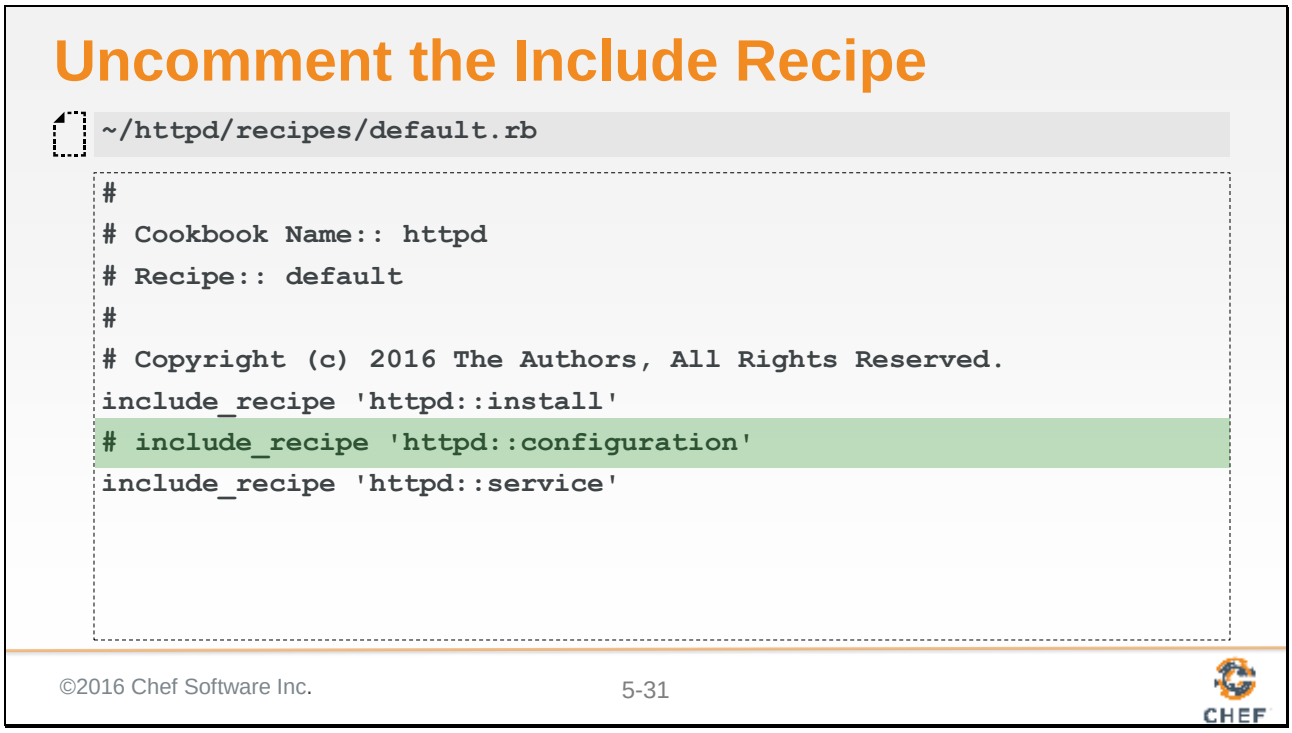
❖ **Repeat this series of steps for each line within the default recipe**

There are few more chances to reinforce this process. As an exercise continue mutating the code within the default recipe, defining the expectations, and then fixing the code. Create a single mutation at a time and become focus on understanding the process of moving between files and executing commands.

Instructor Note: Allow 10 minutes to complete this exercise

Instructor Note: The learners could accomplish both of tasks at the same time. They likely will want to do that. I would encourage you have them perform the steps separately as it will emphasize the activity of moving between the recipe, the specification file, and their shell.

Slide 31



Let's review by walking through one more example within the default recipe. Another line within the recipe is similar to the first one except it is concerned with the inclusion of the configuration recipe. Here it is commented out.

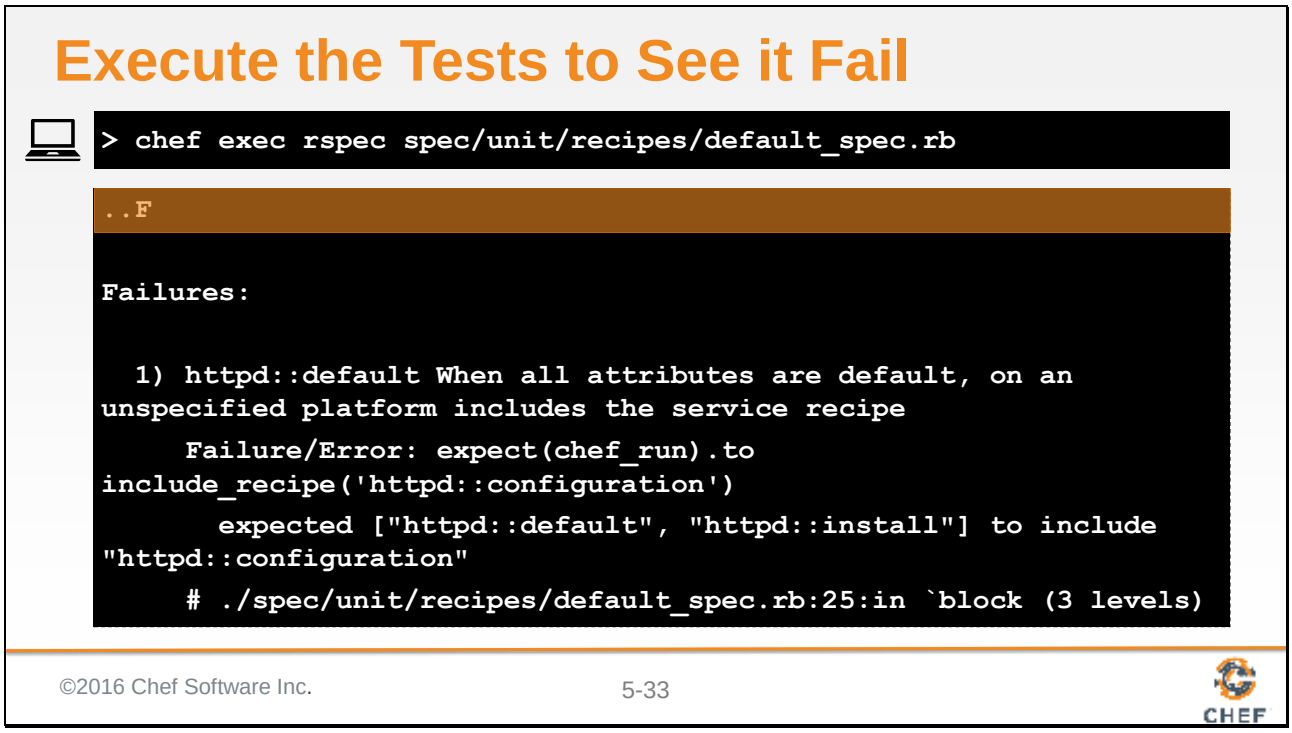
Slide 32

Write the Test that Verifies the Include Recipe

~/httpd/spec/unit/recipes/default_spec.rb

```
# ... START OF THE SPEC FILE ...

   it 'includes the install recipe' do
     expect(chef_run).to include_recipe('httpd::install')
   end

   it 'includes the configuration recipe' do
     expect(chef_run).to include_recipe('httpd::configuration')
   end

  end
end
```


Returning to the specification file to define the example and the new expectation.



Seeing the failure when executing te 'rspec' command.

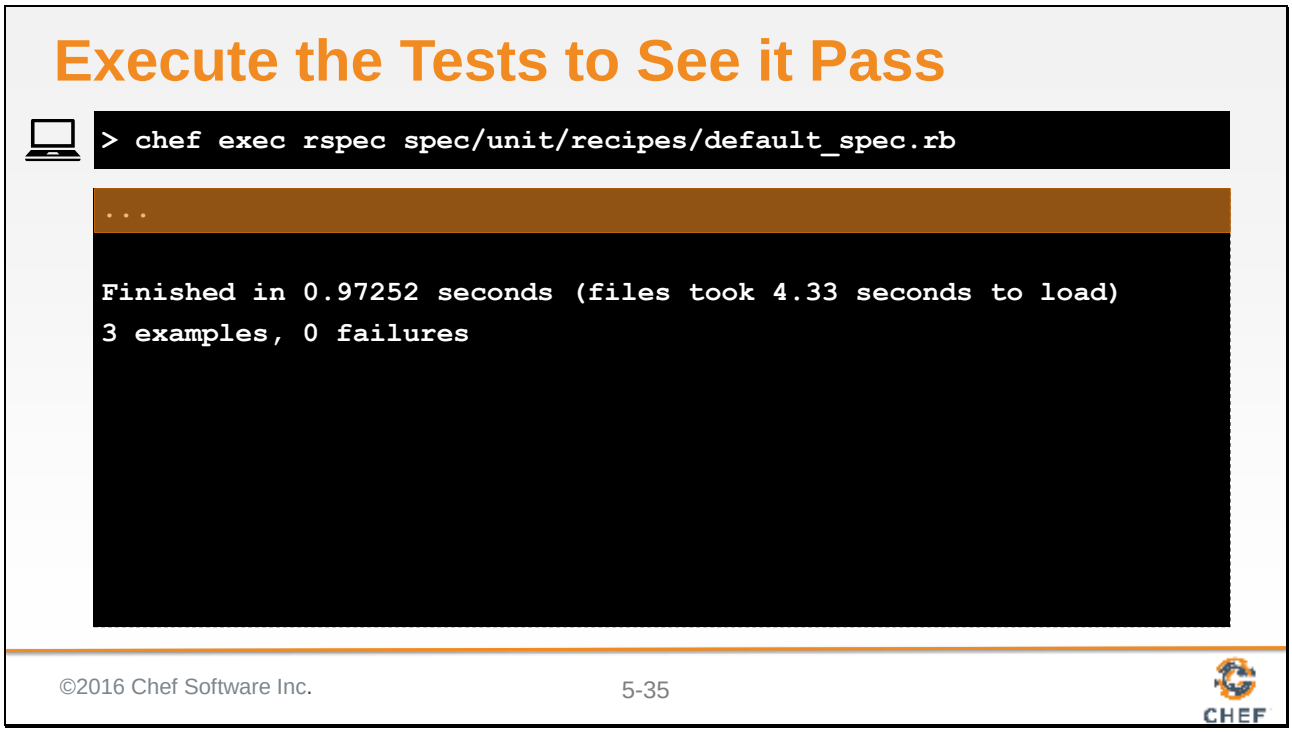
Slide 34

Uncomment the Include Recipe

~/httpd/recipes/default.rb

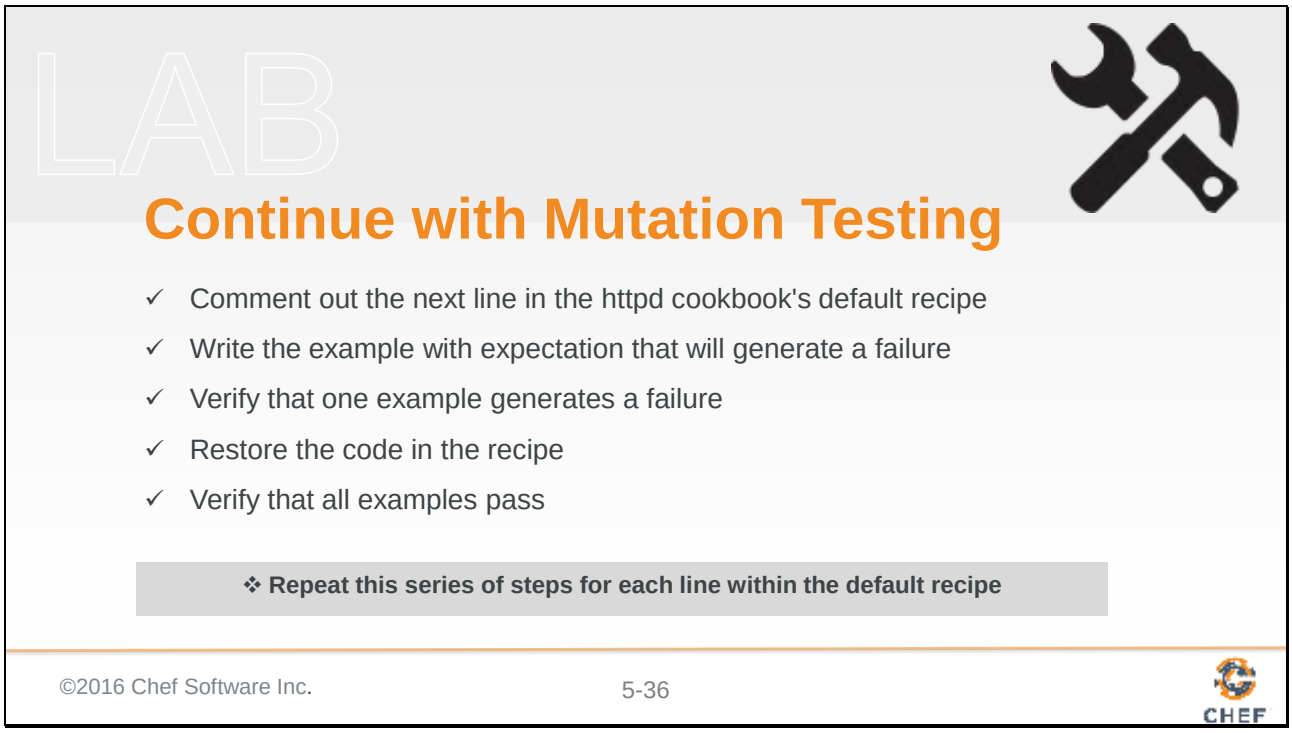
```
#
# Cookbook Name:: httpd
# Recipe:: default
#
# Copyright (c) 2016 The Authors, All Rights Reserved.
include_recipe 'httpd::install'
include_recipe 'httpd::configuration'
include_recipe 'httpd::service'
```


Restoring the code to its previous state



Executing 'rspec' again to verify that the expectations have been met successfully

Slide 36

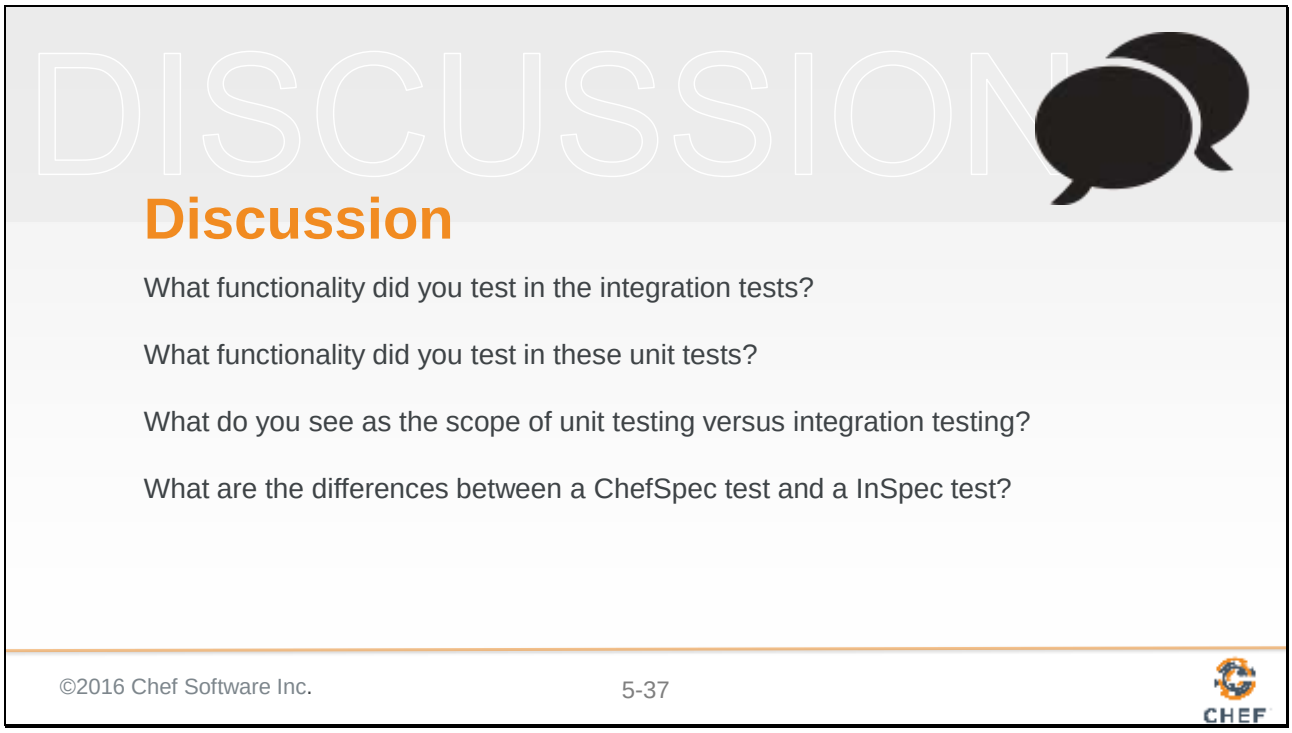


There are more mutations that you could try within the default recipe and other recipe files that exist within the cookbook but this is a good point to stop and enjoy the work that you have accomplished.

The feedback cycle on using Rspec to execute ChefSpec examples returns results faster than we saw with Test Kitchen and gives us a good understanding of what is being added to the 'Resource Collection'.

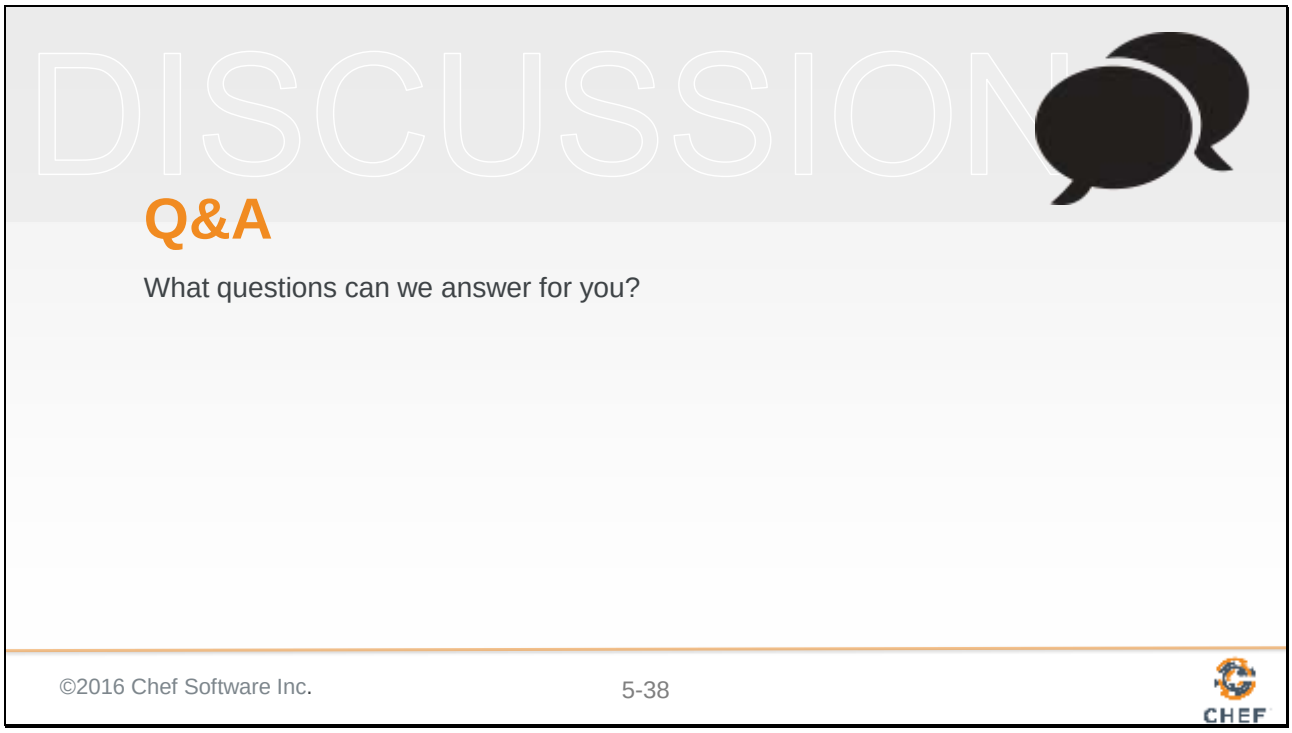
Let's have a discussion.

Slide 37



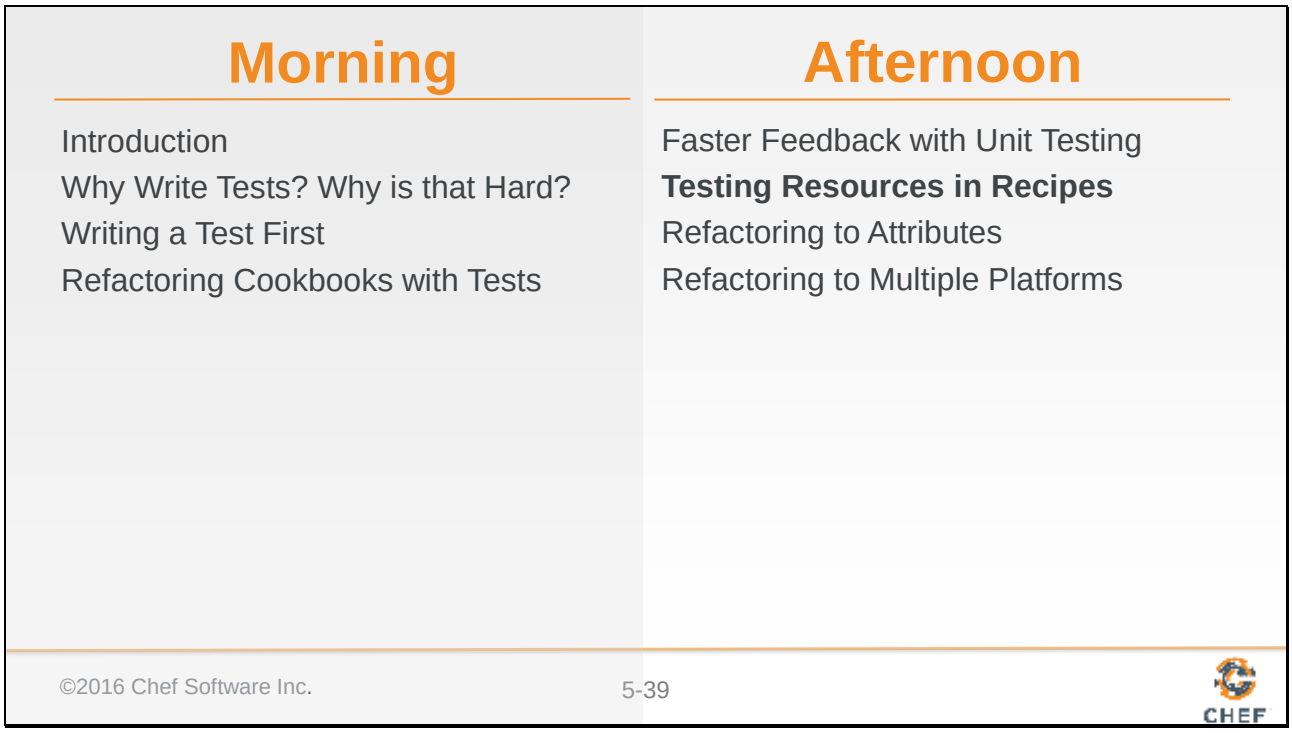
Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.

Slide 38



Before we complete this section, let us pause for questions.

Slide 39



We have the faster feedback that we set out to create for us at the beginning of this section. We were able to verify the work being performed in the default recipe. Now it is time to focus our attention on the remaining recipes with in the cookbook and set up expectations on the resources that they define.

Slide 40