Welcome to the afternoon's workshops everyone - what we're going to look at is the notion of software development at many different levels. I've called these levels small, medium and large here but that is purely an arbitrary distinction I've made and is in fact fairly meaningless.

The key message is that the traditional notion of development is to solve business problems by coding up a solution to that problem. This is far too narrow an application of software development techniques if you think of coding as being a way of automating manual processes. With this mindset, you can use your coding skills for a far wider and more varied set of activities than just solving the business problem presented to you by your clients or customers.

And an introduction to some of these activities is exactly what these workshops are about.

Objective

1. Solve a (trivial) business problem
2. Check we've solved the problem
3. Automate the previous step
4. Let everyone see our solution

So what exactly are we going to do over the next couple of hours? 4 main things...

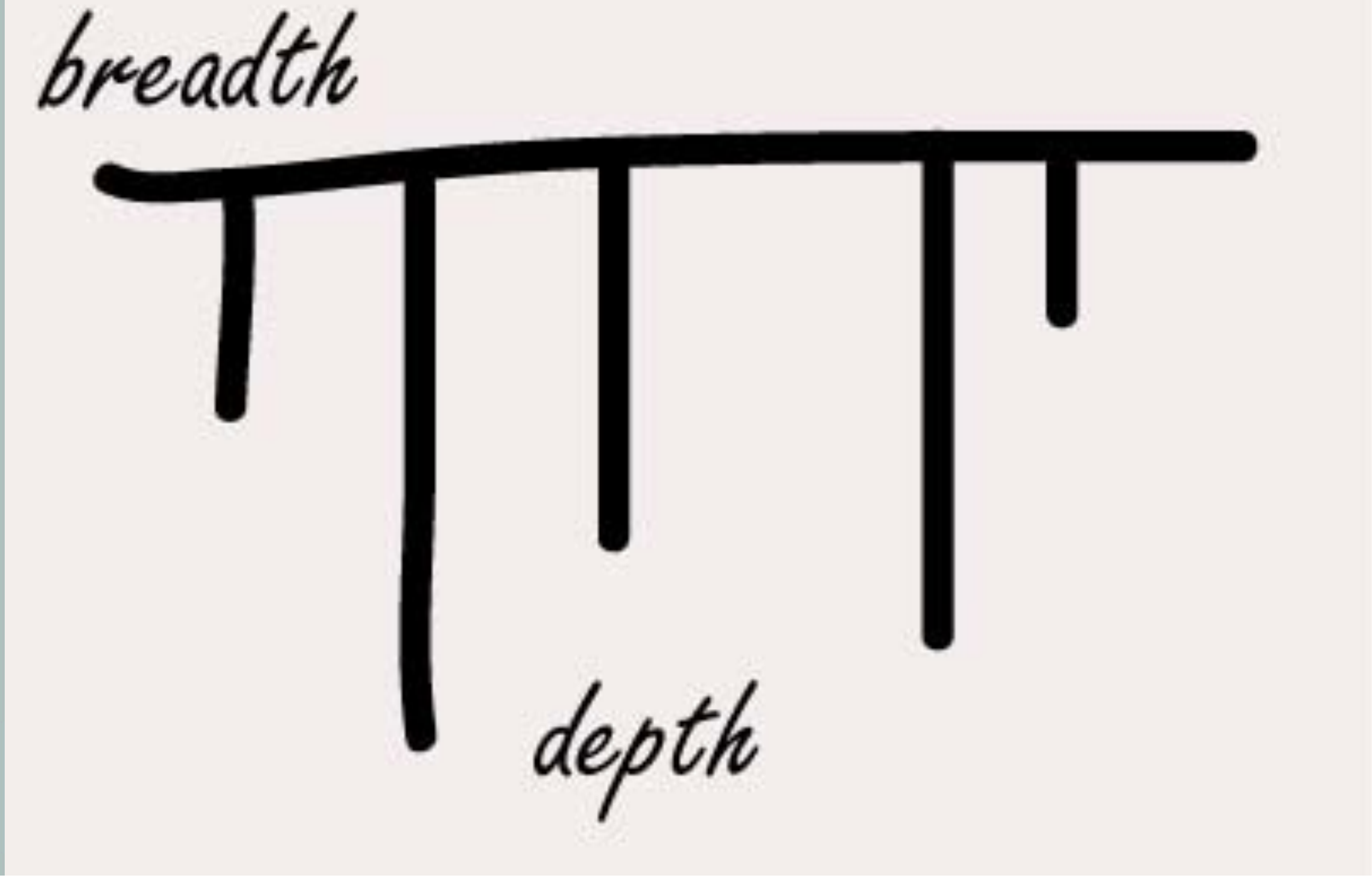
The important part about each of these activities is that we'll use code to for each of them. The first one is the narrow definition of development I spoke about on the previous slide, the remaining three are concerned with using software development skills to support our main job - solving business problems.

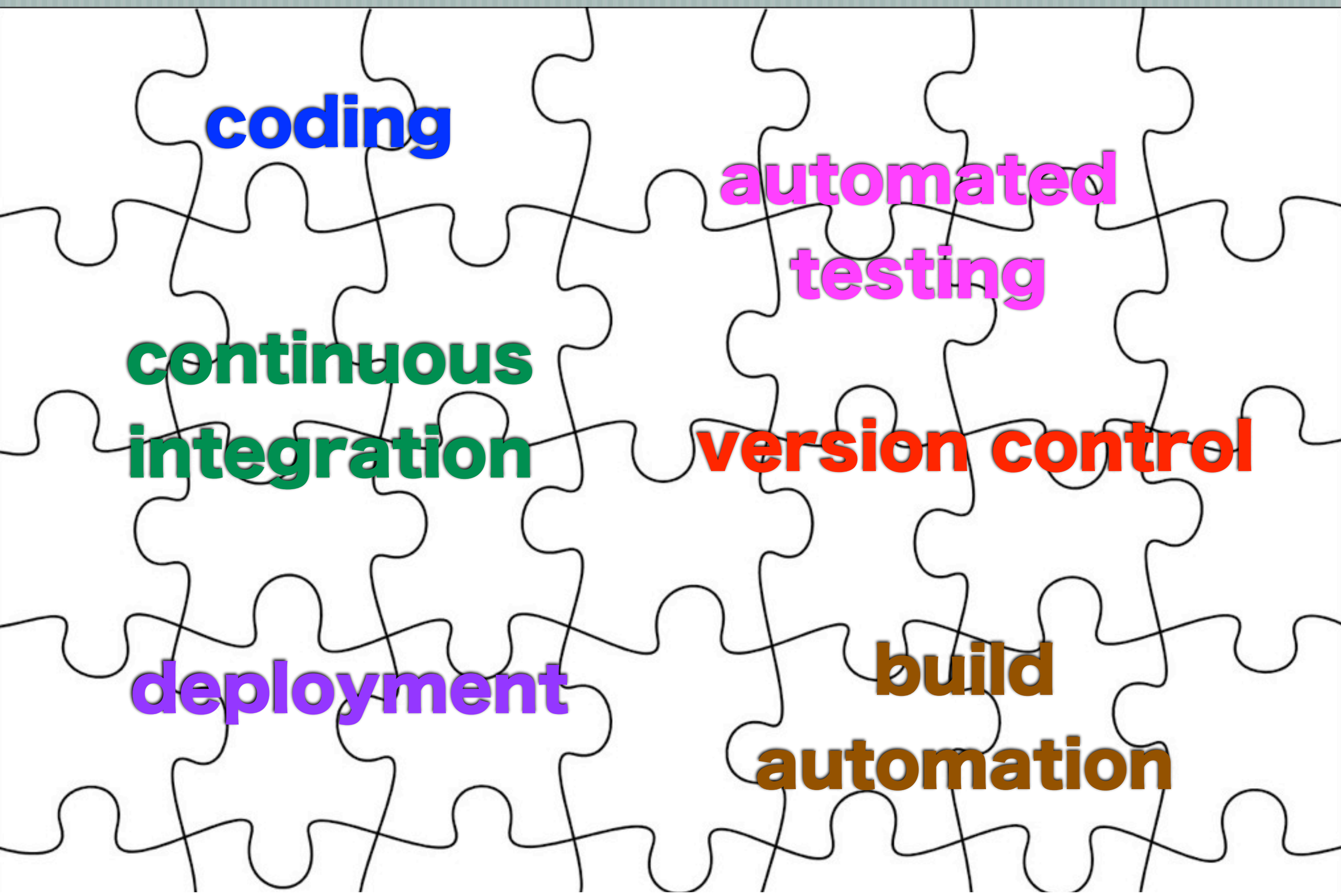
We will spend about half our time working through the first three points and the remaining half on the last one if all goes according to schedule.

Knowledge gained

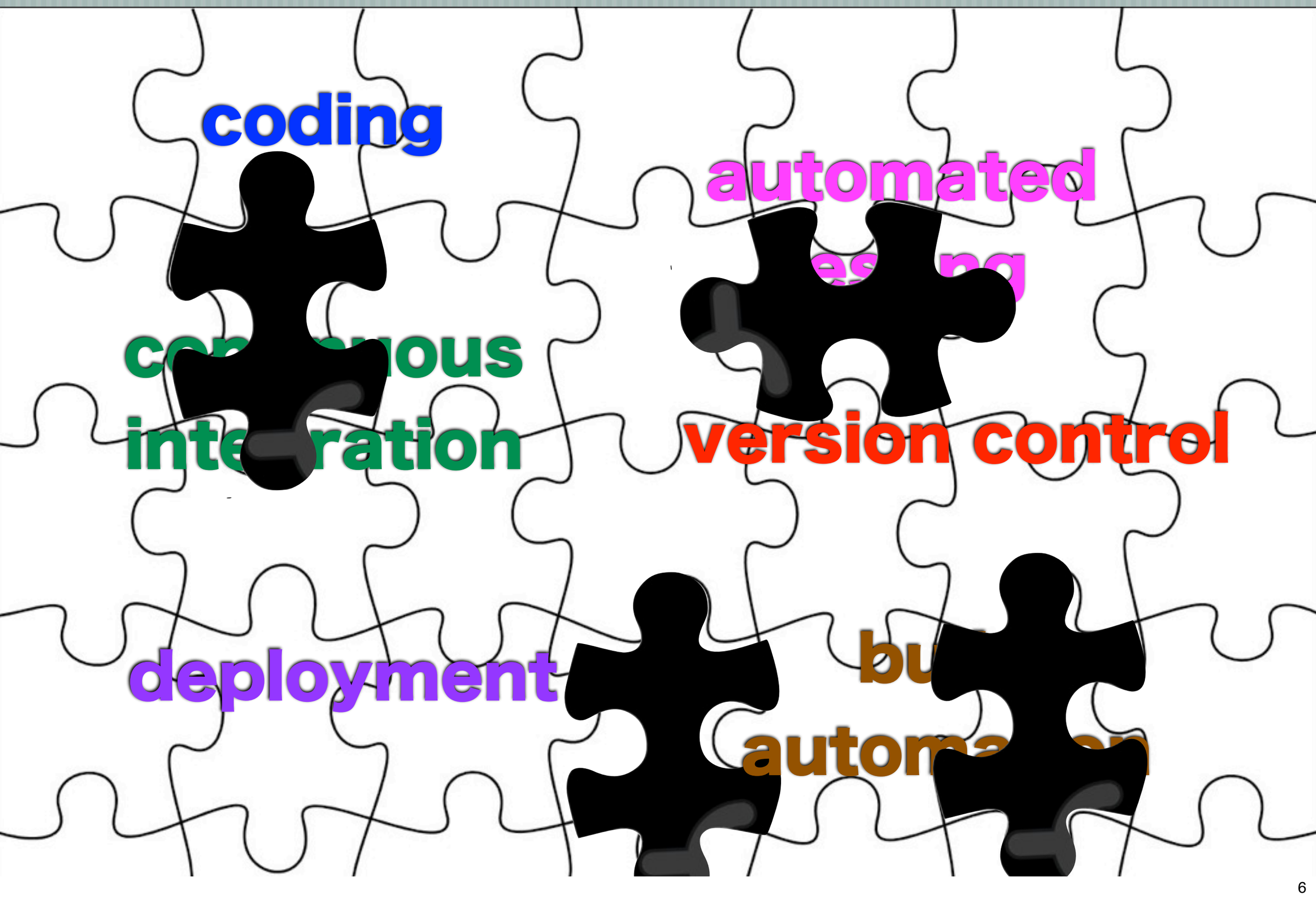
- Basic ruby coding
- Test automation using RSpec
- Build automation using Rake
- Continuous Integration using Travis CI
- Code hosting/versioning using GitHub
- Application hosting using AWS

And this is what I hope you will get out of taking part in these workshops. To be perfectly honest, I should probably have added “basic” to each of these points because time will prevent us going into great detail on any of these, but I hope to give each of you at least a basic idea of what each of these tools are and what problems they are intended to solve.


breadth
depth



And following on from the previous slide, it's important to note that I've structured these workshops to emphasise exposing you to a wide range of technologies. And to do this in a relatively tight timeframe I've had to provide a lot of the fundamental elements of the solution pieces...



and leave a few suitably sized gaps for you guys to work on during the exercises. So for those people who prefer to work in a depth first manner, I apologise for this structure but I think it's the best way to get a large-ish group across a number of topics in short time.

All the moving parts

And here's how we'll build out all those elements.

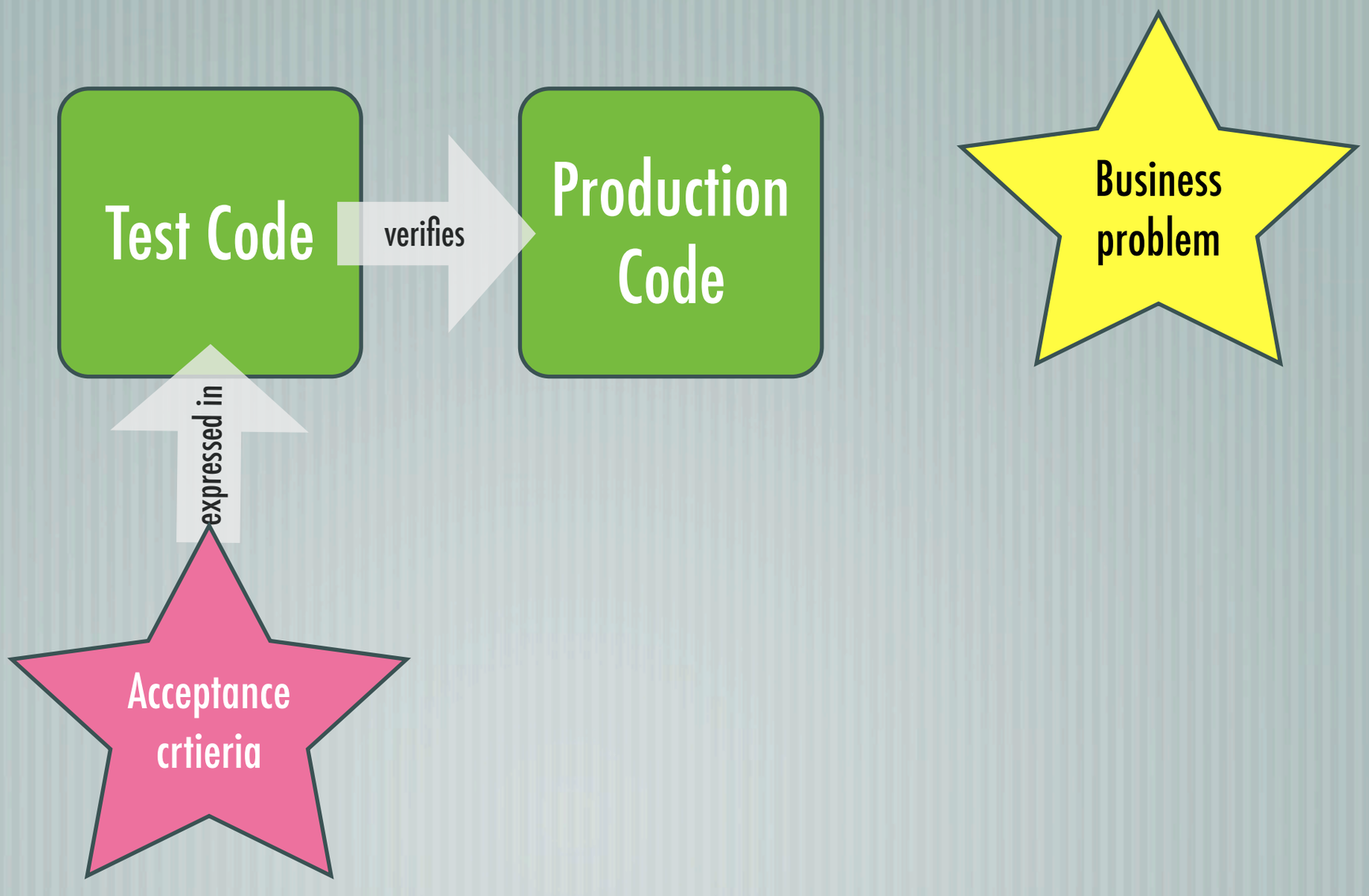
All the moving parts

And here's how we'll build out all those elements.

All the moving parts

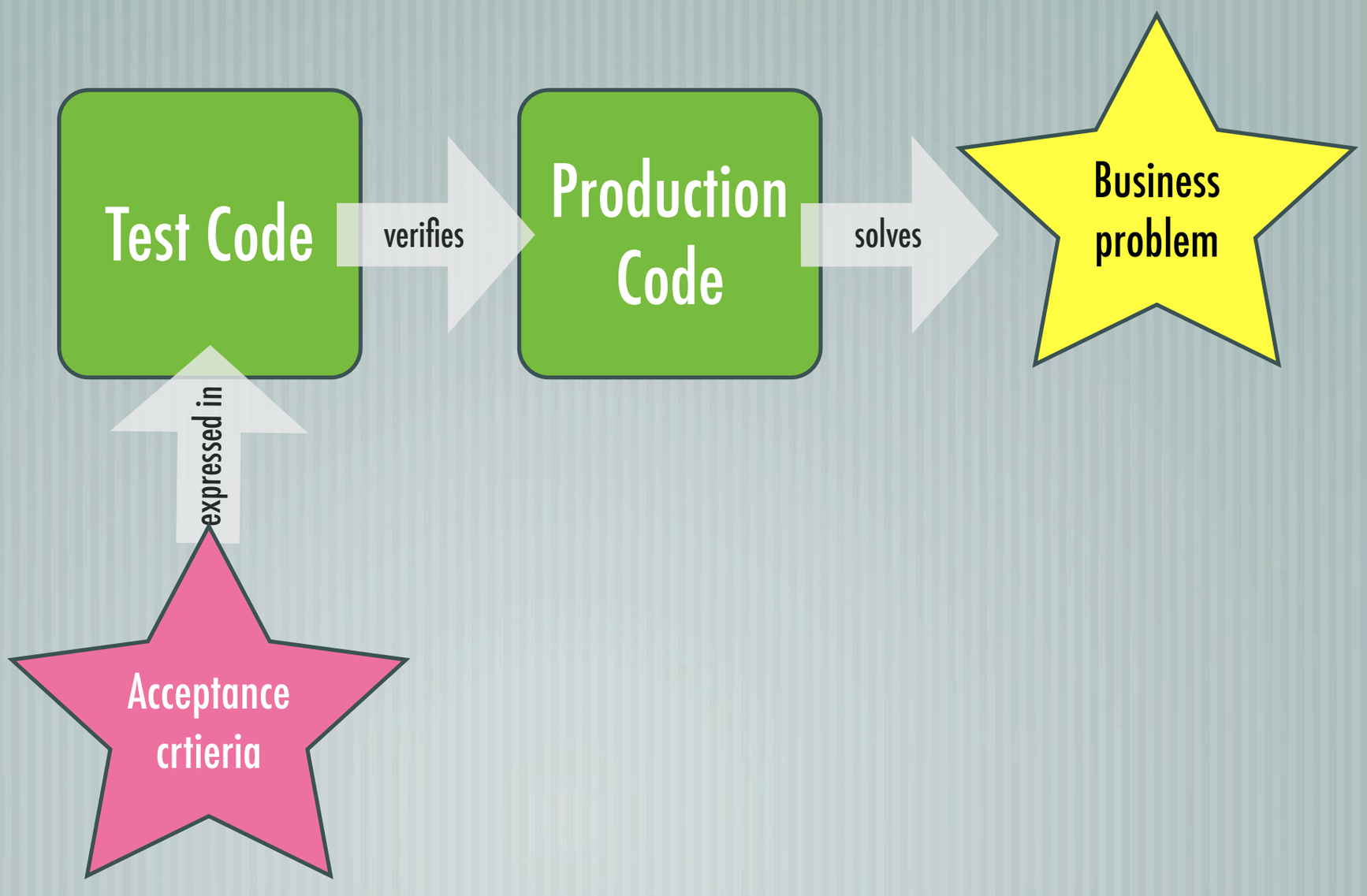
And here's how we'll build out all those elements.

All the moving parts



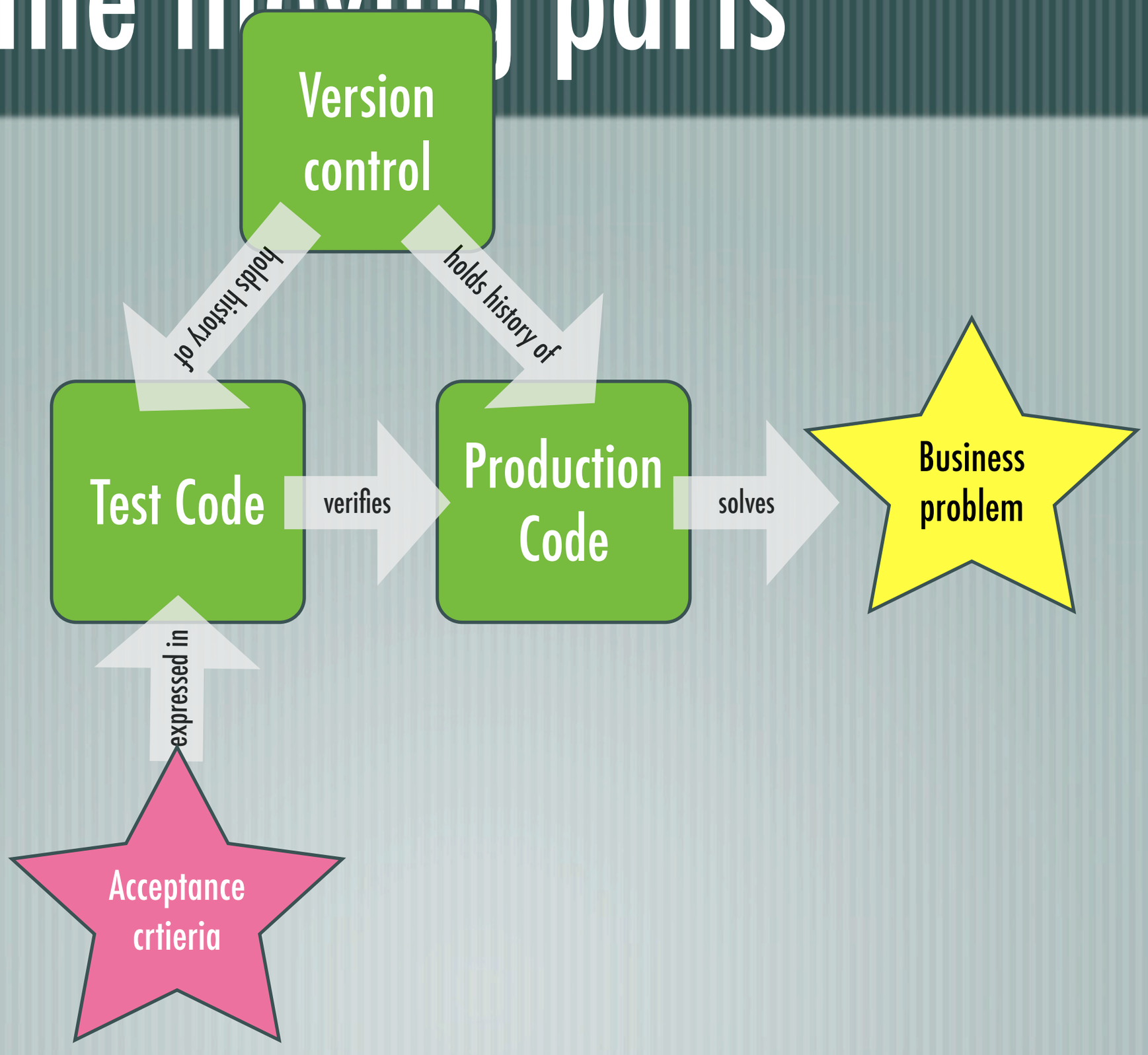
And here's how we'll build out all those elements.

All the moving parts



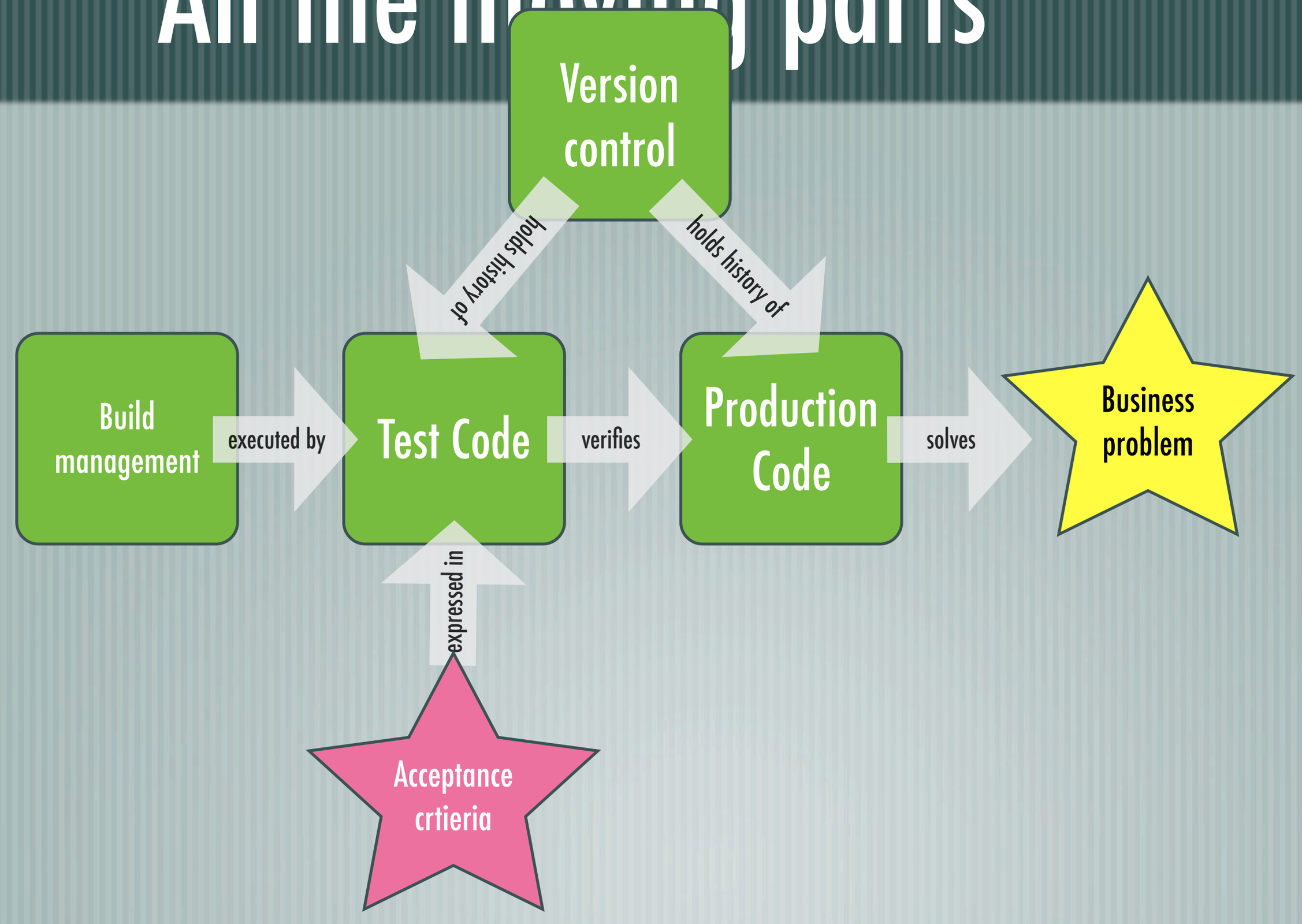
And here's how we'll build out all those elements.

All the moving parts



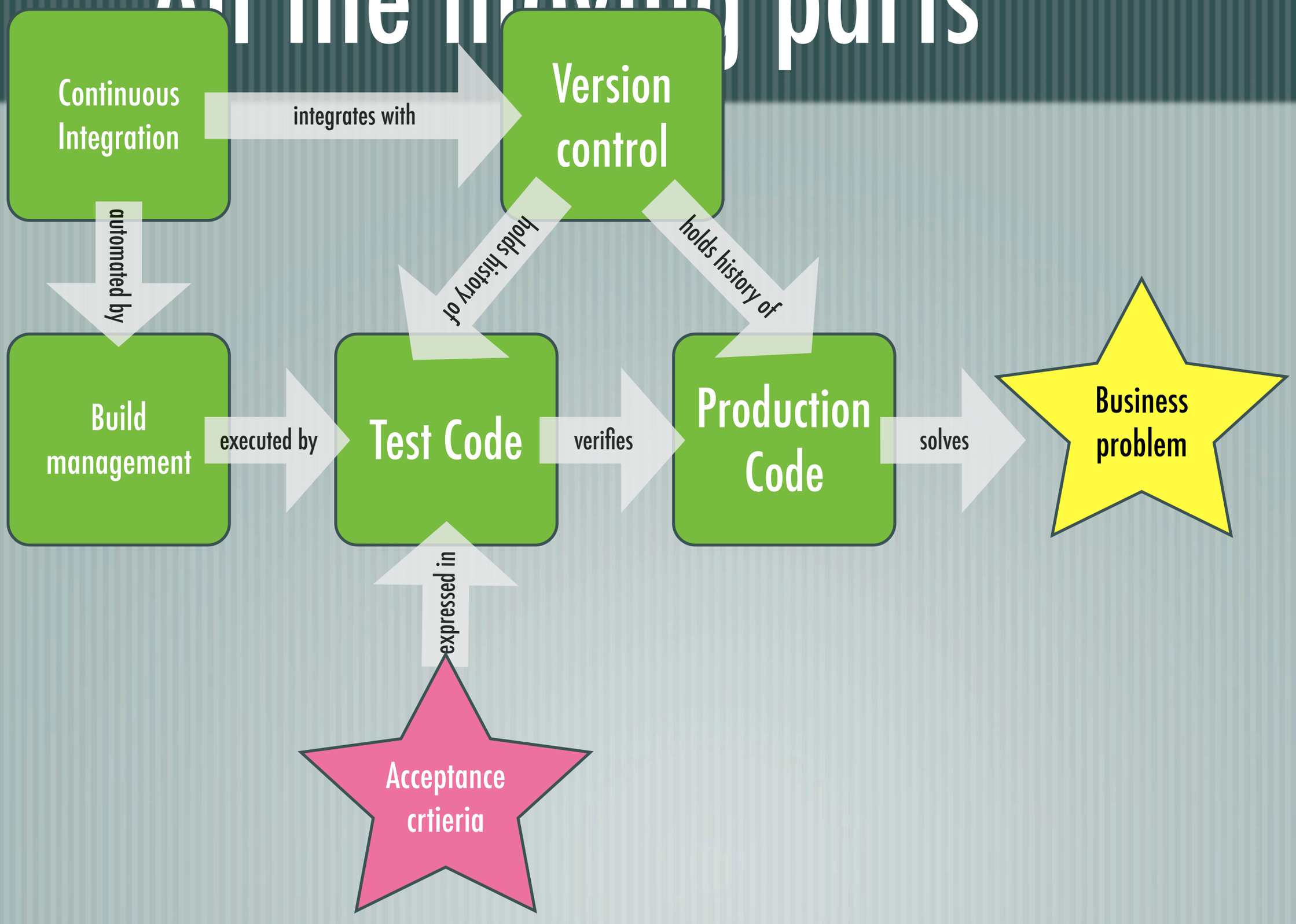
And here's how we'll build out all those elements.

All the moving parts



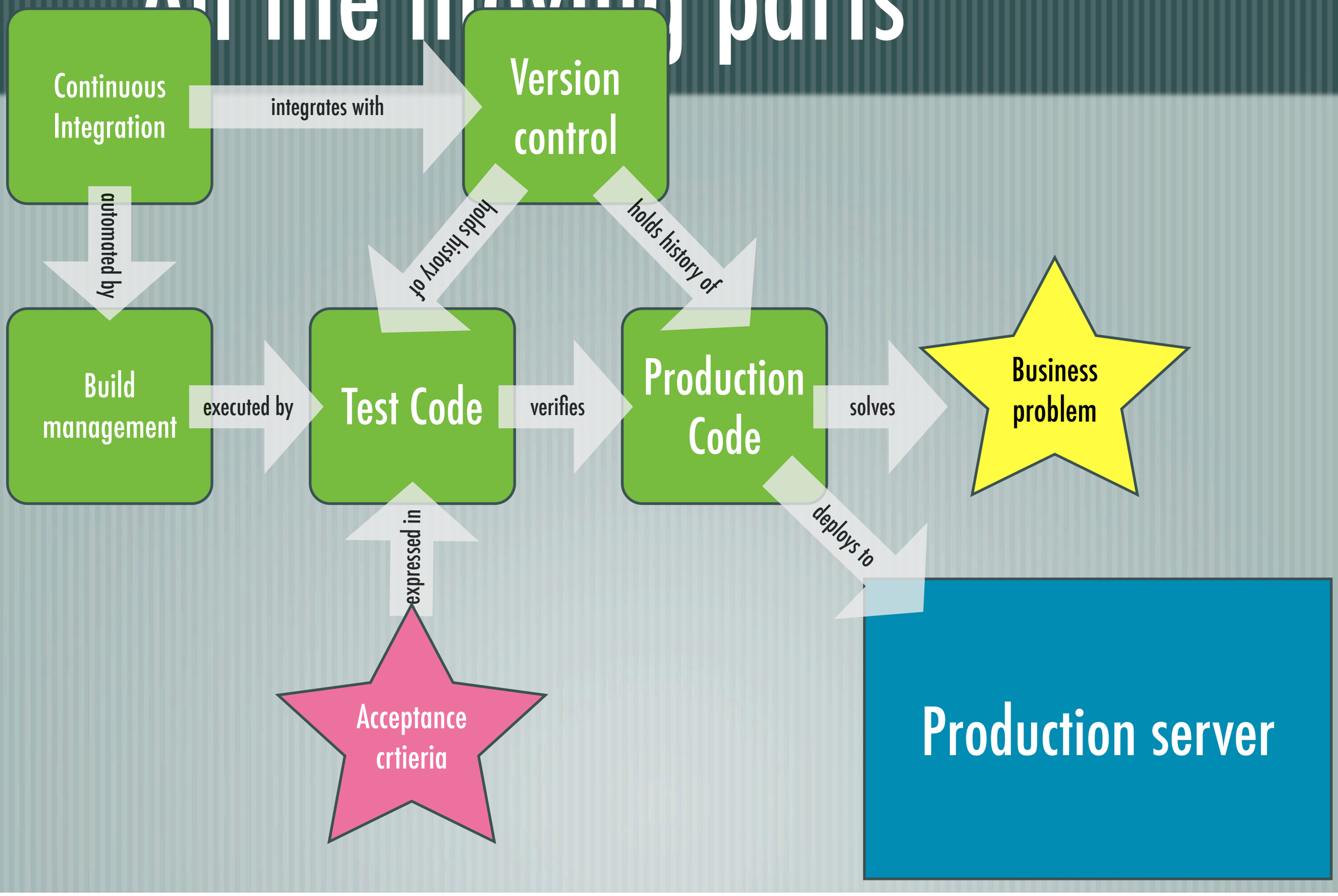
And here's how we'll build out all those elements.

All the moving parts



And here's how we'll build out all those elements.

All the moving parts



And here's how we'll build out all those elements.

What's in our toolbox?

And these are the main tools we're going to use on the path to enlightenment about development in the small medium and large.

Ruby is our programming language [poll on usage] and we'll get a bit of time to write a bit of Ruby code to both solve a small business problem and also test that our solution works as expected. Ruby is a good choice of language because:

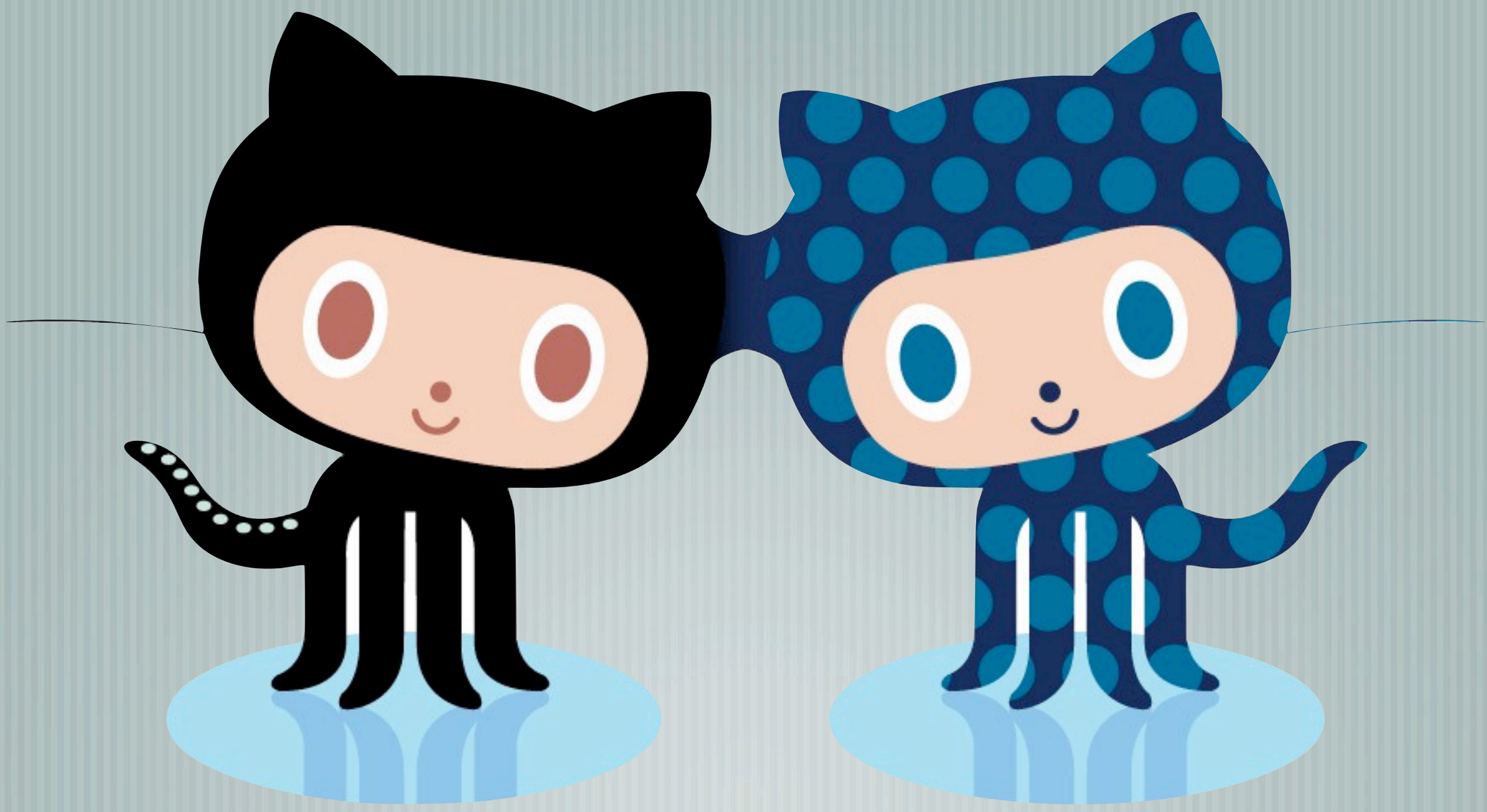
- the number of steps you need to get a ruby program running is very low,
- it can be written in a way which is generally quite easy to read, even if you are unfamiliar with the language,
- it integrates well with some of our other tools, in particular...

Travis-CI, which is a hosted solution that we'll use to make sure our Ruby code is working properly. We'll talk more about Travis-CI later and how it works.

GitHub, which I mentioned in my keynote, will be used to hold the code we write and as a very basic development environment which means I don't have to worry about everyone setting up their own environment.

Heroku and AWS will be used to host our solution on the Internet, which will be our last big activity of the day.

Mode of working



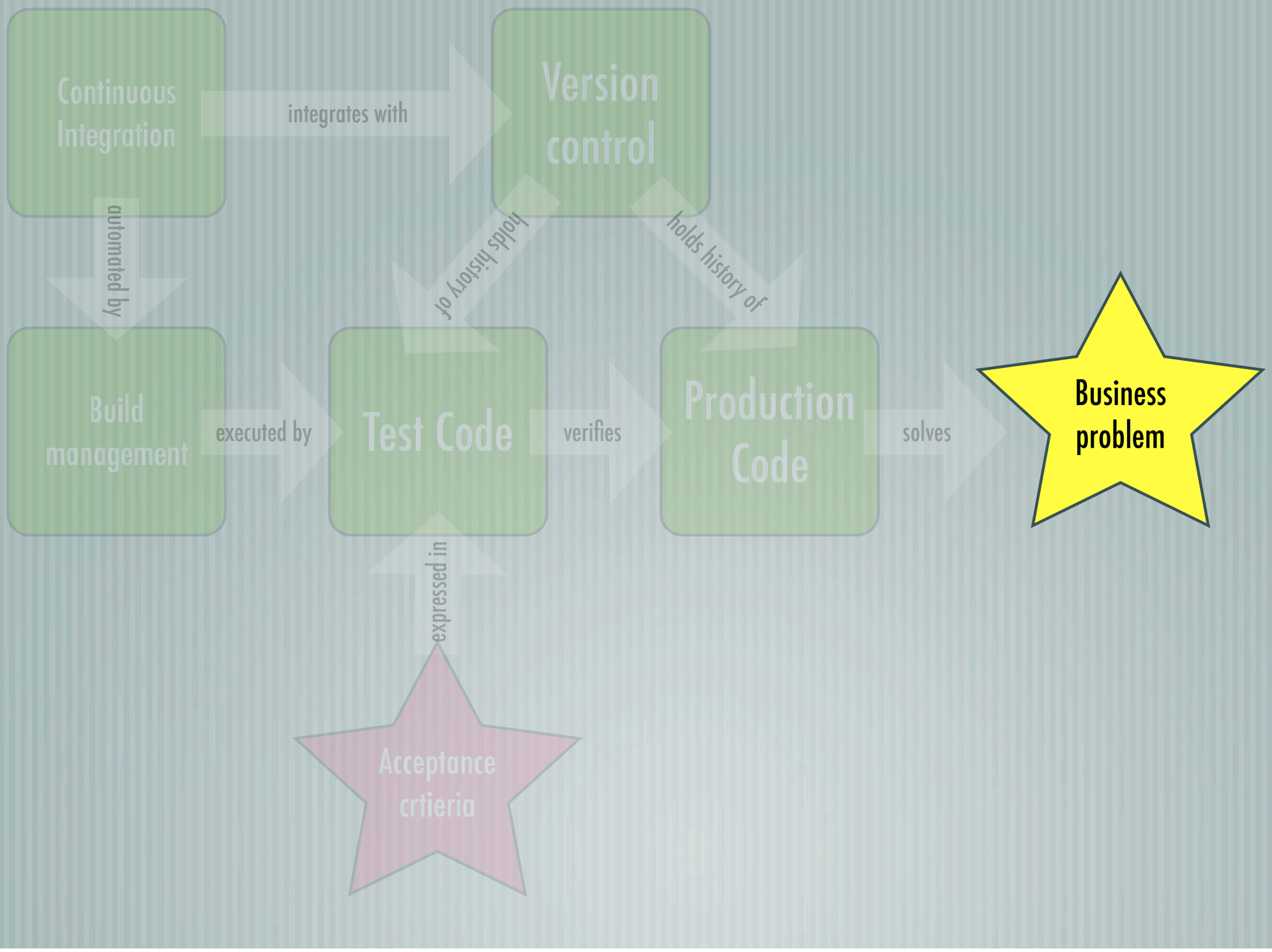
Pairing, pairing, pairing.

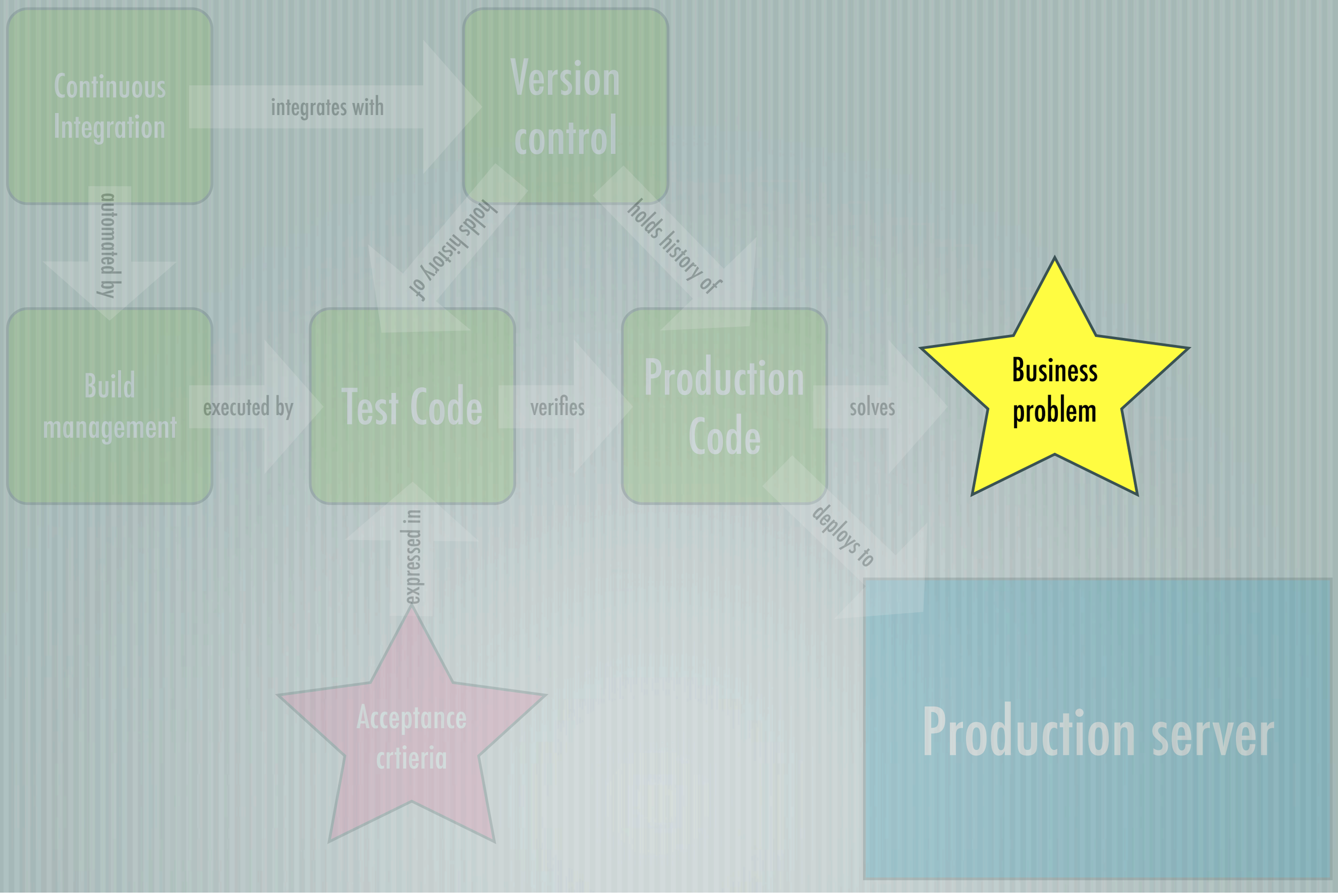
The Business Problem

- Given the time (in Singapore) is currently X, what is the difference to the time in country Y?
- For example: how far away (in hours) is New York from Singapore?

And speaking of the business problem – here it is…

Now I did warn you that it's trivial didn't I :–)


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
Acceptance crtieria


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
deploys to
Acceptance crtieria
Production server

1. Specify how it should behave

it should…

it should…

it should…

etc.

Can we take 5-10 minutes or so to think of a bunch of descriptions for how we think this clock should work. This is crucial part of building any piece of software however big or small - doing a little bit of thinking about how we think this clock should work so that:

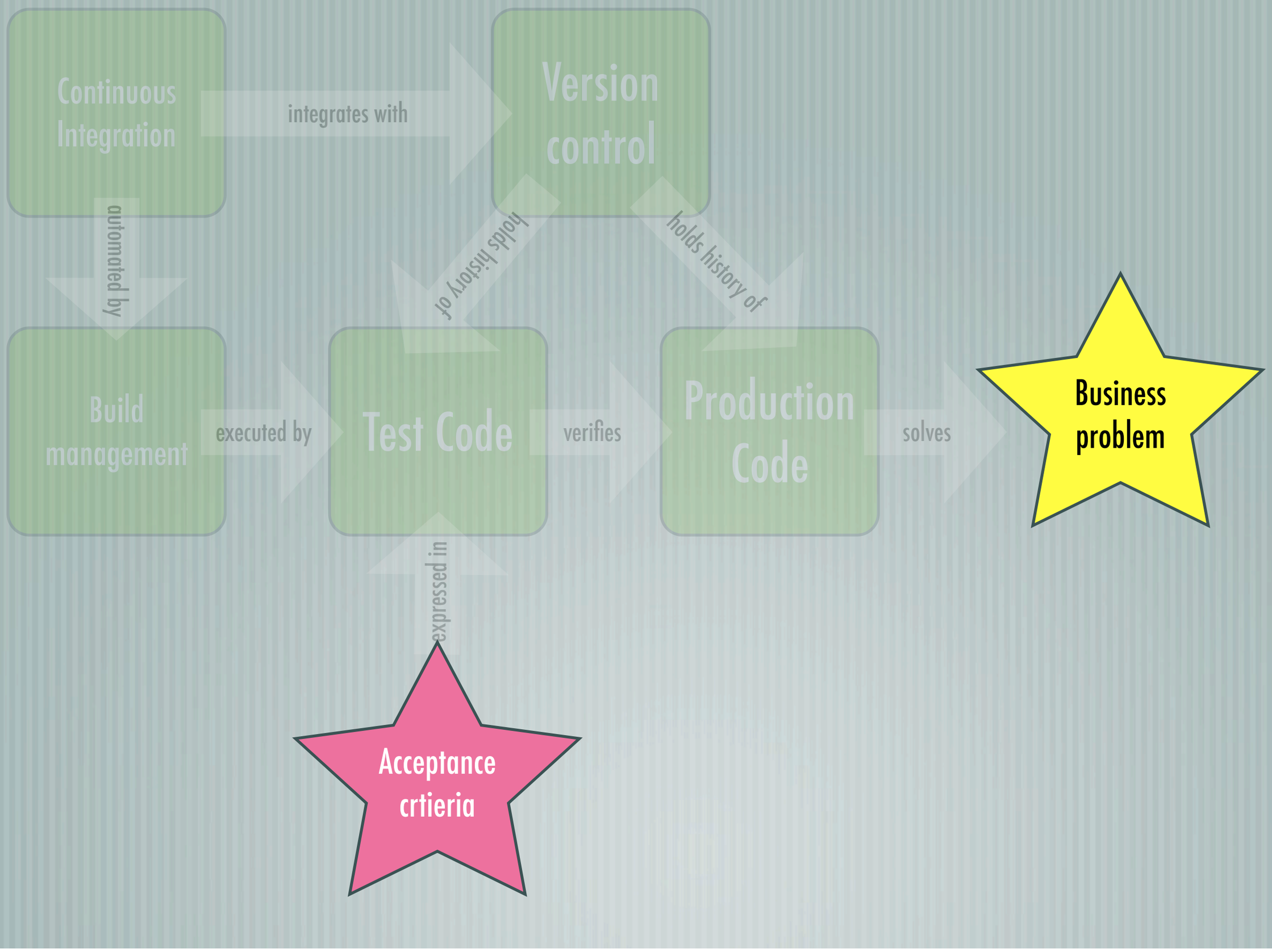
- we know when it is working,
- when it's not working,
- when we still have work to do, and
- when we're finished

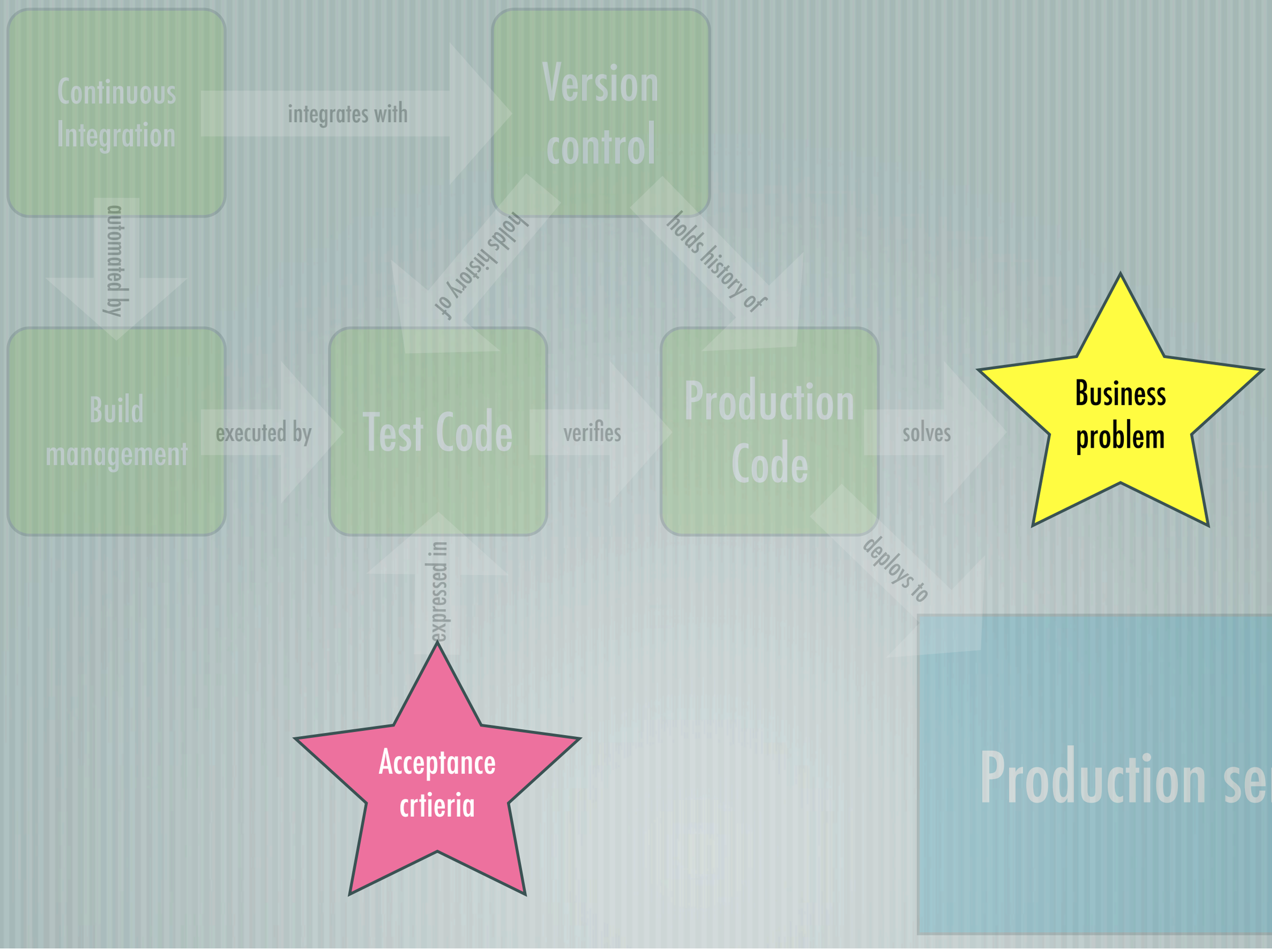
And this is an aspect of software development that still requires significant human input.

For example...

- it should give no time difference to a city in the same timezone as Singapore
- it should give -2 hours difference to Sydney
- it should tell you if it doesn't recognise the city

You'll notice one of these examples is general in nature and one is quite specific – it's good to think along both axis as part of this activity. Two of the examples are positive in nature (i.e., testing expected paths with valid data) and the other is testing a negative path


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
Acceptance crtieria

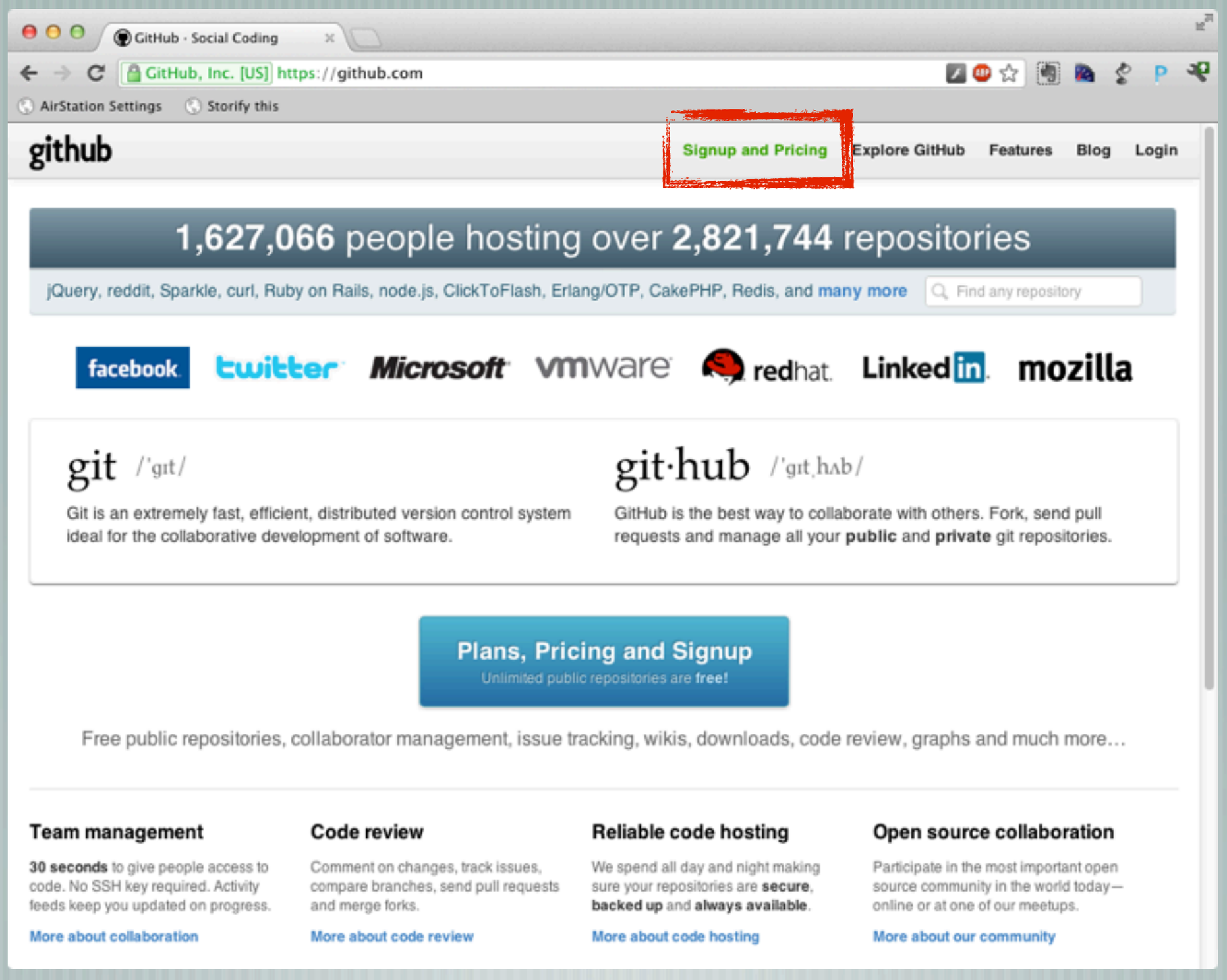

Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
deploys to
expressed in
Acceptance crtieria
Production server

2. Create your GitHub account

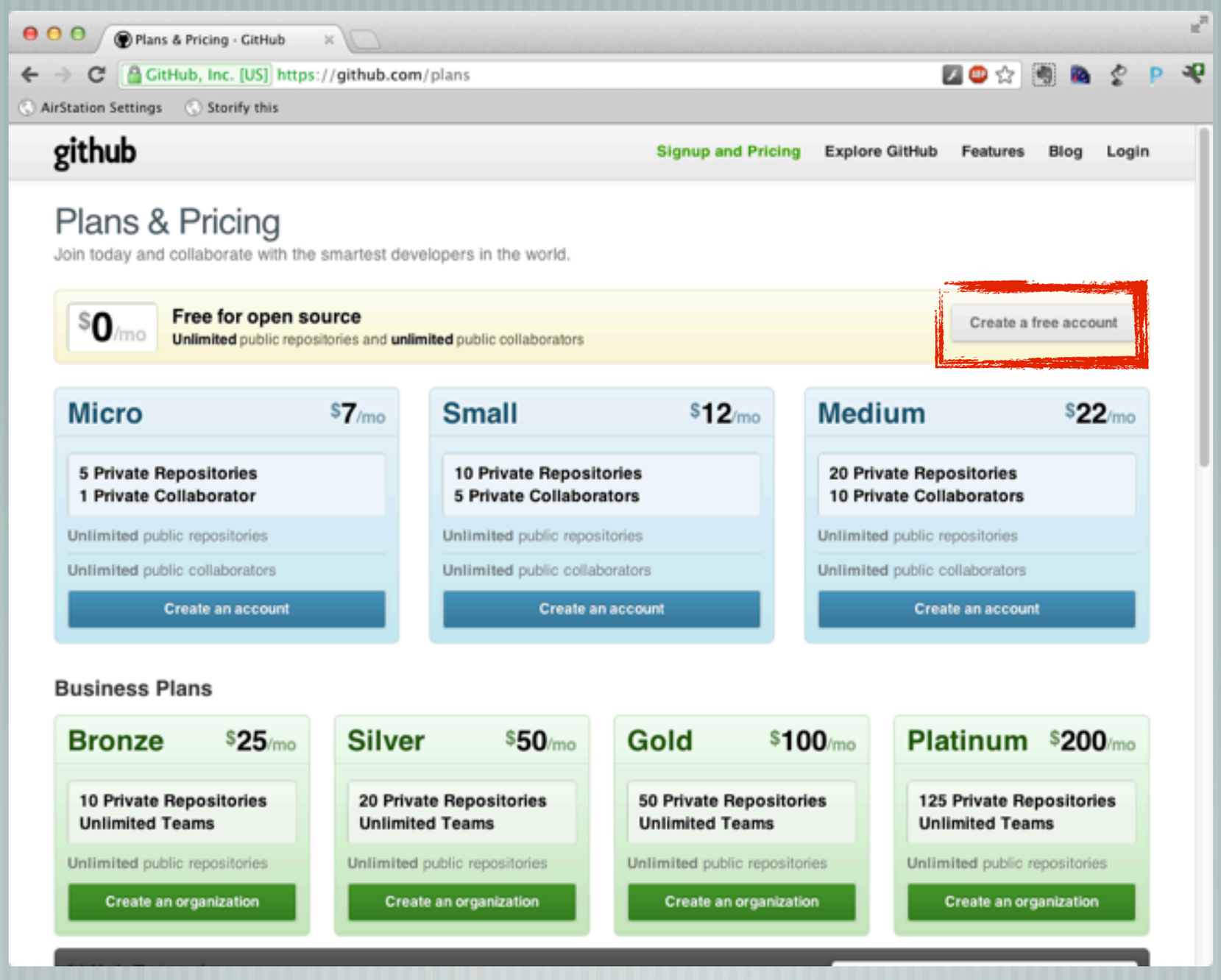
www.github.com

So the first thing we need to do is for each of you to create a GitHub account. This is a relatively smooth and quick process, doesn't require you to provide any credit card information and can be the basis for a living resume of your coding prowess over time.

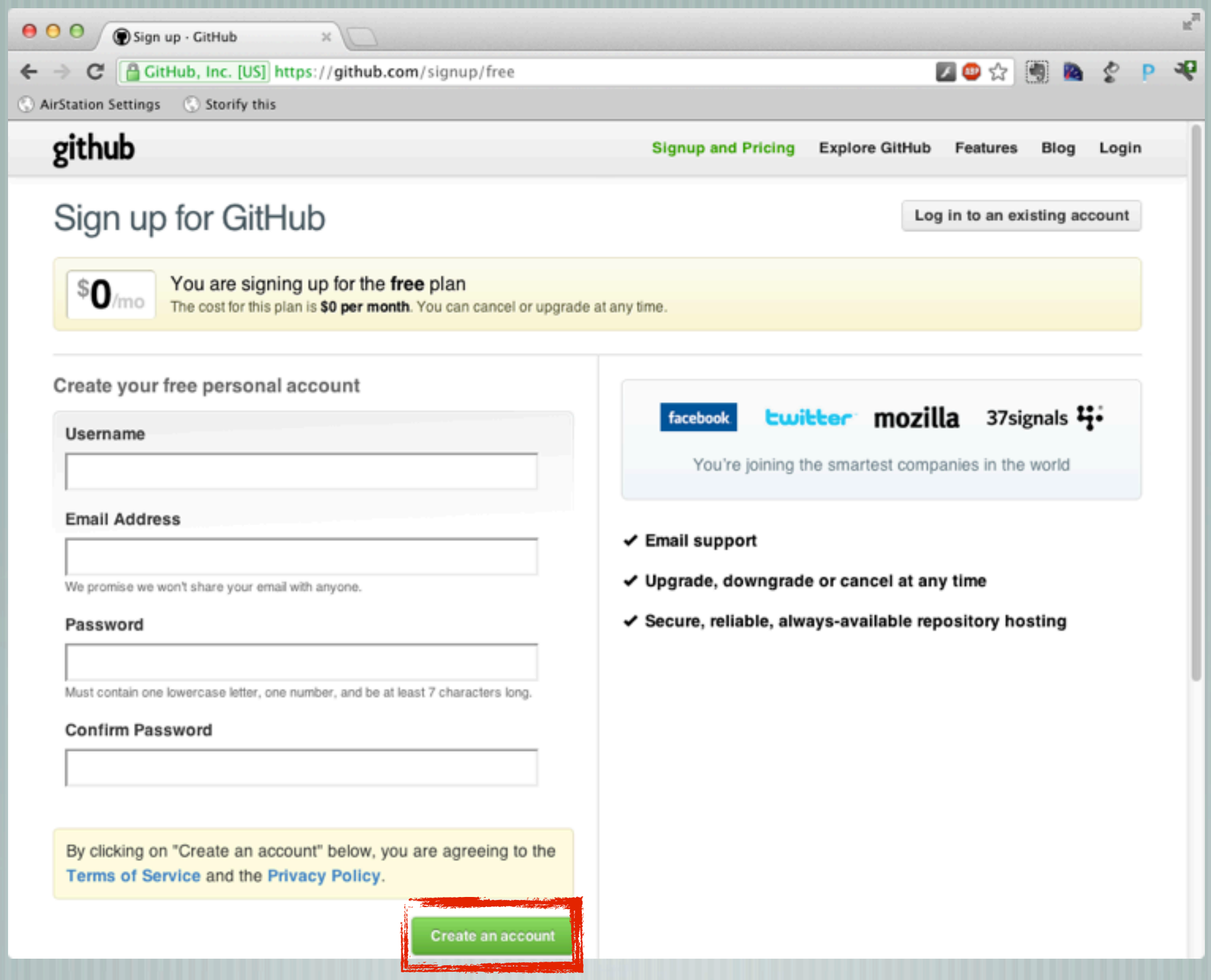
Obviously, if you already have a GitHub account feel free to re-use that one.



So start at github.com and click the Signup and Pricing link on the top nav.

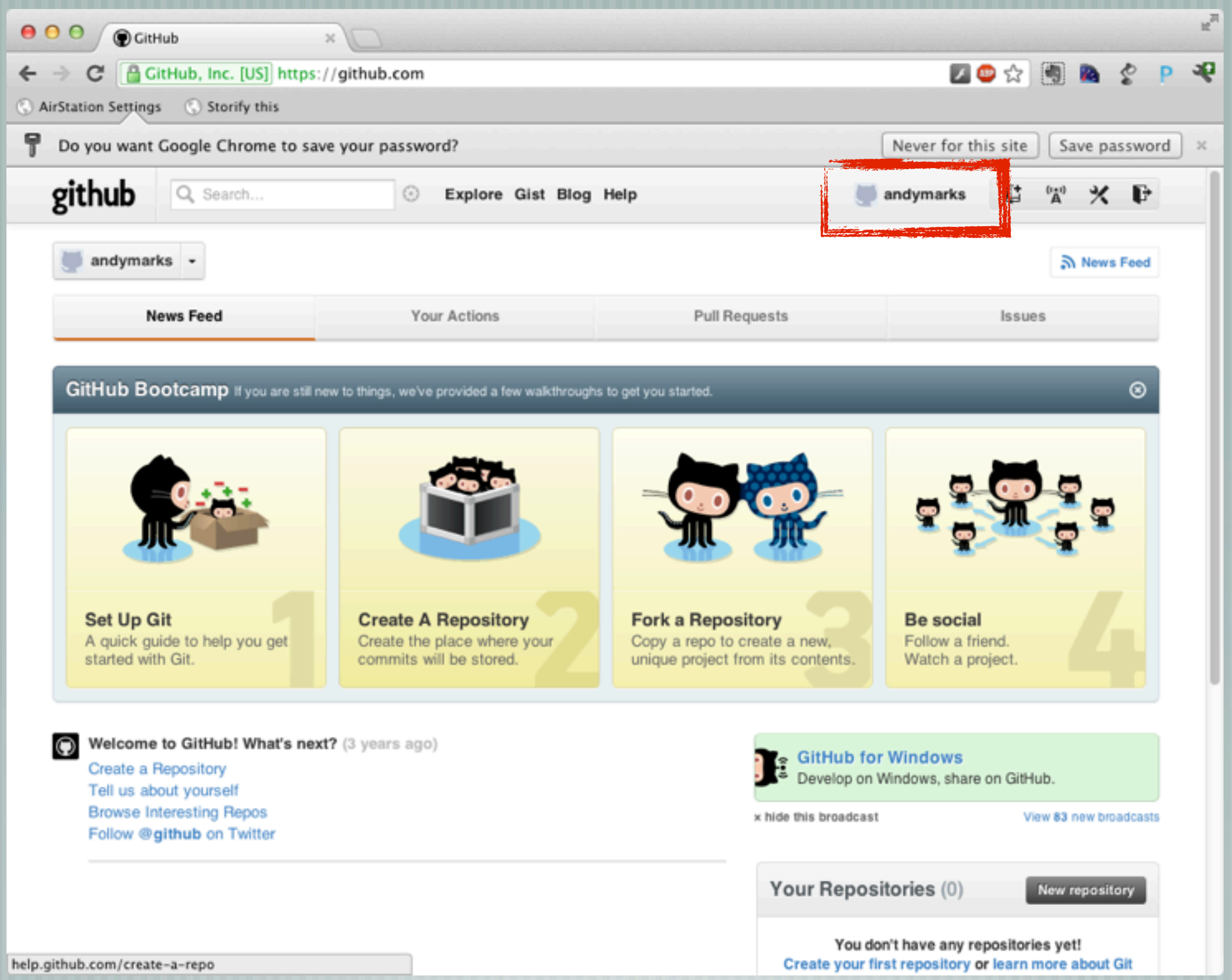


Which will take you to this page at which point you're completely free to choose any of the plans they advertise... but you may as well just choose Create a free account.



Fill in the details. It doesn't matter too much at this phase, but your GitHub account name can become an important credential if you plan on doing a lot of open source development, so sometimes it's worth taking the effort to chose one that is either a good representation of yourself or catch or hopefully both.

When you're happy with all of your details - click Create an account.



And now you should be signed in... at which point let me say a little more about GitHub...

What is GitHub?

GitHub is an organisation that runs a website for hosting source code using the Git version control system.

Version control systems are software tools that allow developers to keep a running history of all the changes they make to the code they are running on - giving us a nice big bouncy safety net to use in case we do something wrong... not that that ever happens of course :-)

Git is a relatively modern and highly successful form of version control system and GitHub is basically a massive installation of Git with a web front-end.

GitHub also has lots of cutesy drawings of Octocat (Octopus + Cat), which is it's mascot.

In Git and GitHub terminology, each set of source code hosted on it is called a repository. You can have many repositories for different projects. We'll only be using one today.

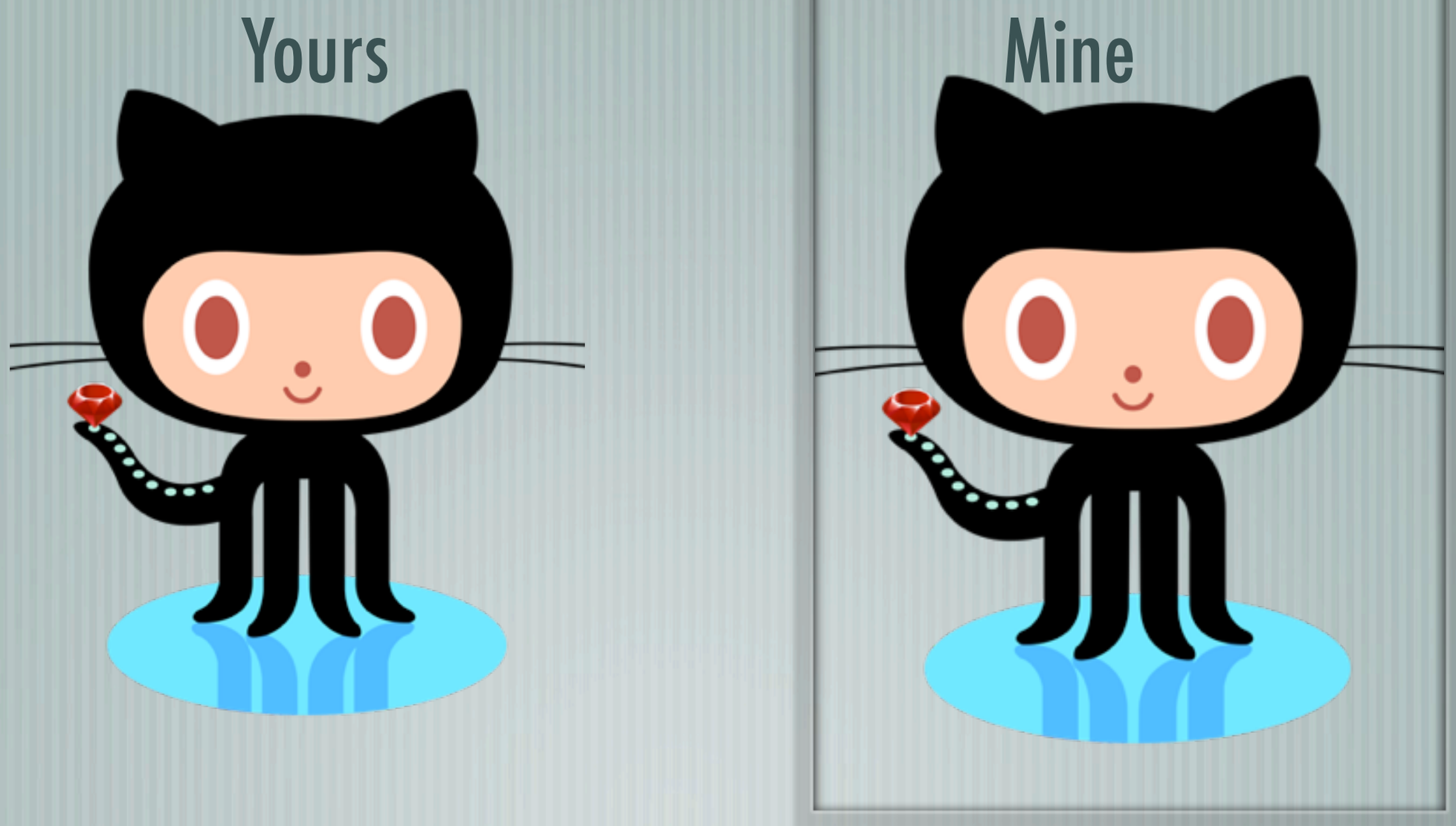
3. Copy the starter code

Now I mentioned before that I'll be providing a lot of the basic bits and pieces of what we'll be working on today and I'll be doing that by sharing a code repository that I have created with you.

In fact, it sounds like I've explicitly allowed this to happen but if you have a free account on GitHub then everything you host there is automatically "gettable" by anyone else. If it's code privacy you're seeking, you either need to use a paid GitHub account or run your own version of Git or some other version control system.

But by cloning this repository, you'll get your very own copy of the code under your newly created GitHub account and you'll be able change that code completely independently of any of my changes.

3. Copy the starter code

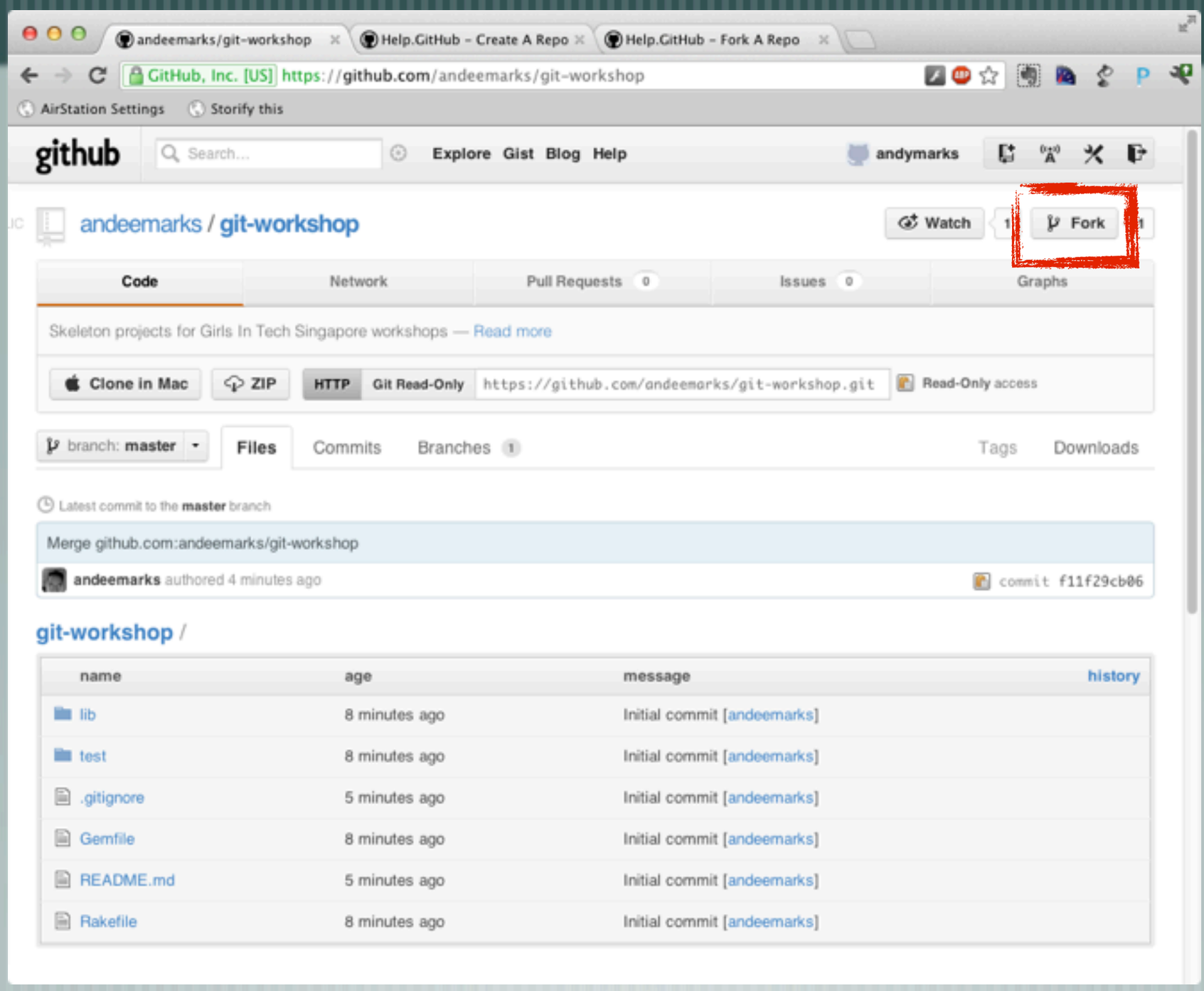


Now I mentioned before that I'll be providing a lot of the basic bits and pieces of what we'll be working on today and I'll be doing that by sharing a code repository that I have created with you.

In fact, it sounds like I've explicitly allowed this to happen but if you have a free account on GitHub then everything you host there is automatically "gettable" by anyone else. If it's code privacy you're seeking, you either need to use a paid GitHub account or run your own version of Git or some other version control system.

But by cloning this repository, you'll get your very own copy of the code under your newly created GitHub account and you'll be able change that code completely independently of any of my changes.

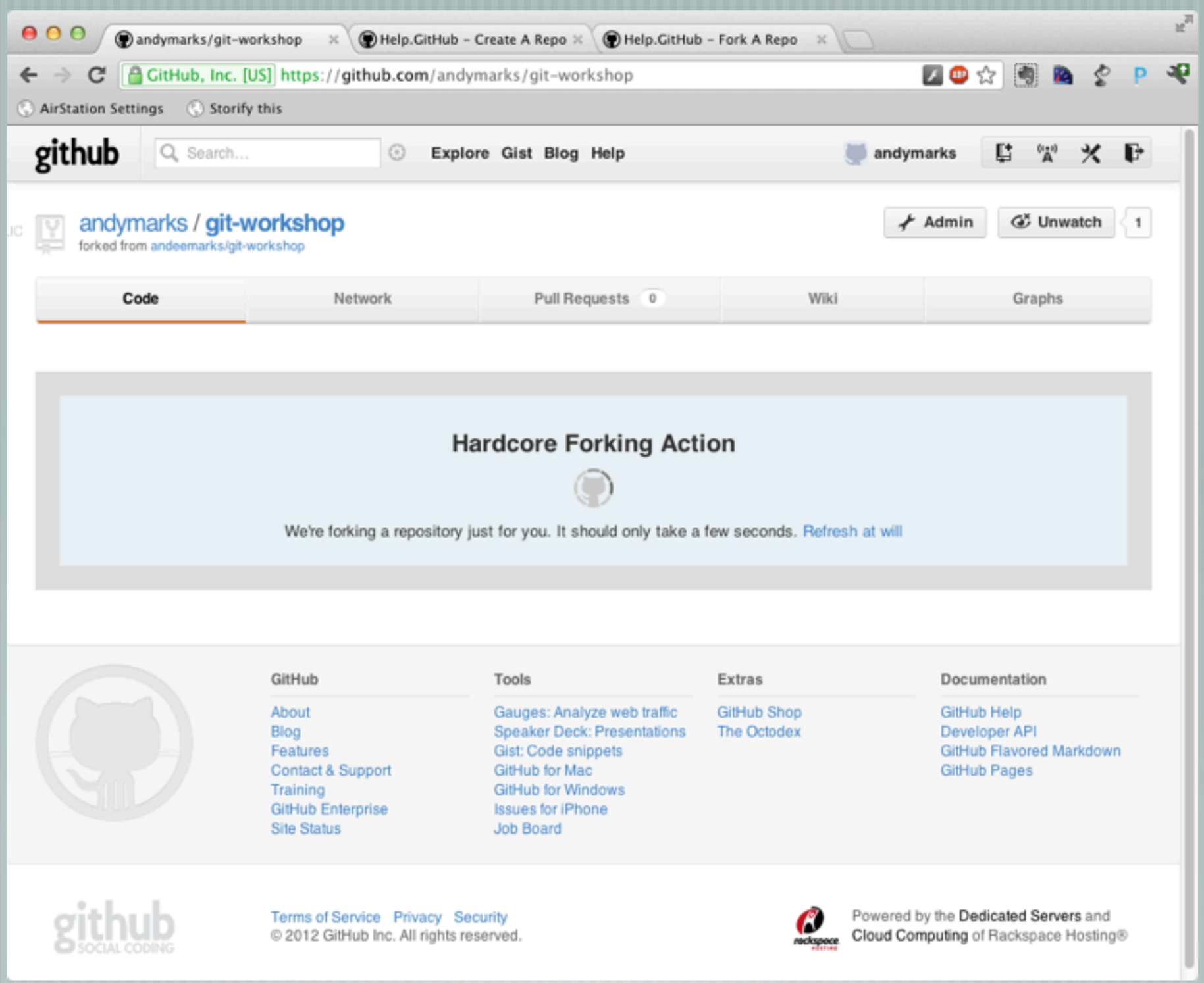
https://github.com/andeemarks/git-workshop



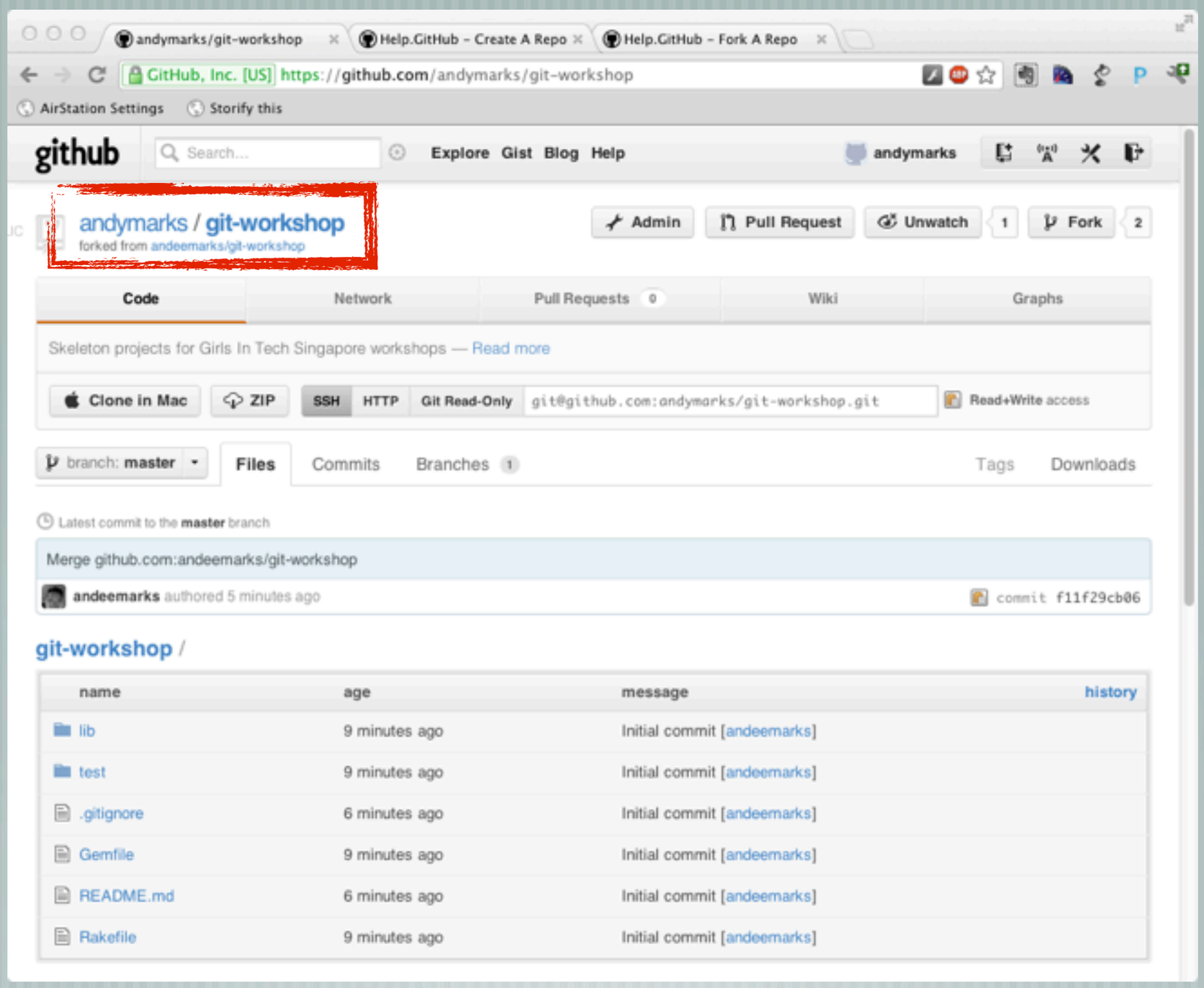
To start this activity, browse to my GitHub account and the repository I setup for this event. The URL is at the bottom of the slide.

BTW, you might notice my user name changing around a little in these slides: don't worry about it. I was switching around between a couple of my GitHub accounts when I was taking screenshots and wasn't always consistent.

Once you're on this page, click the Fork button to start the process of creating a copy of this repository under your GitHub account.

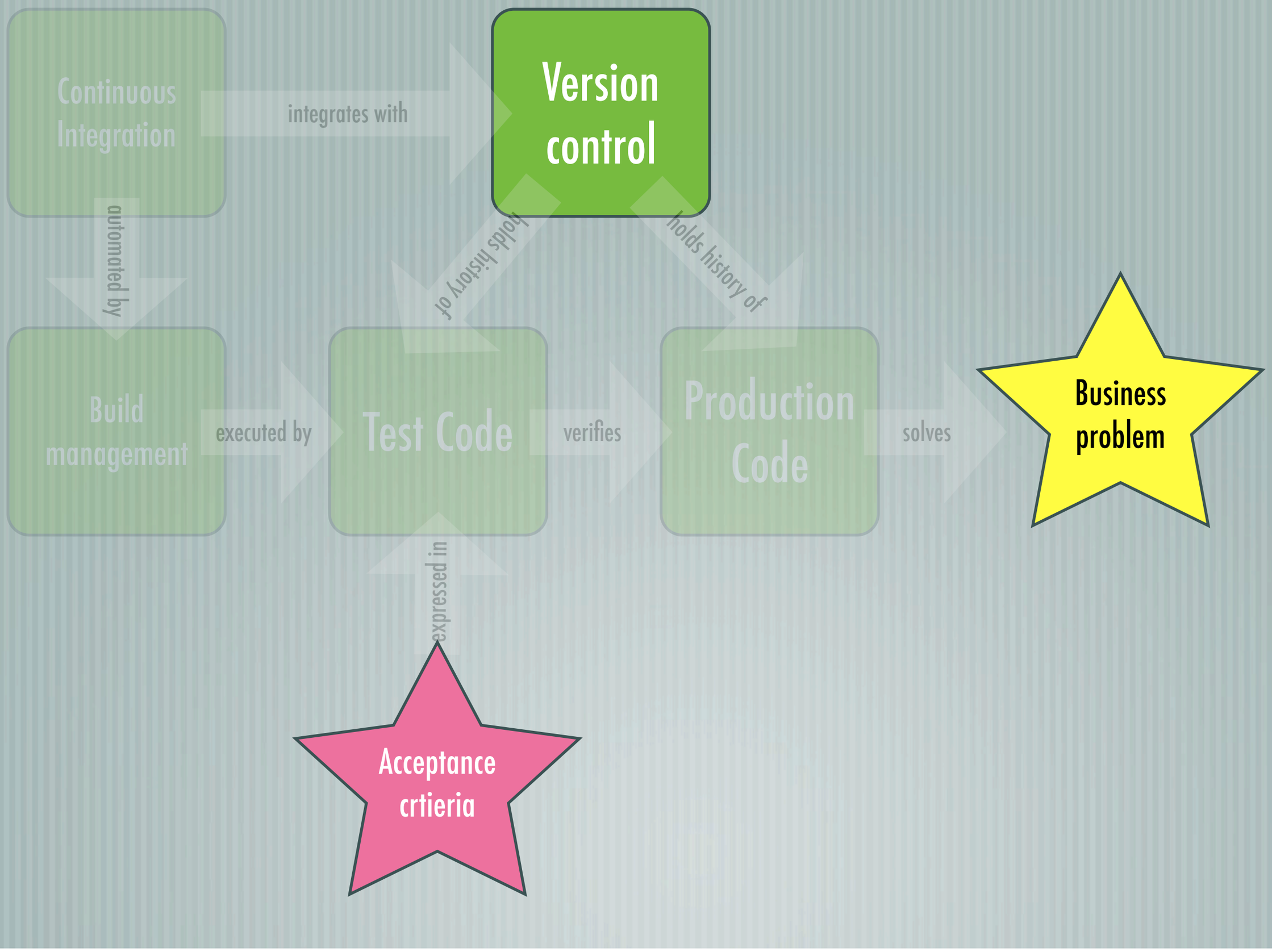


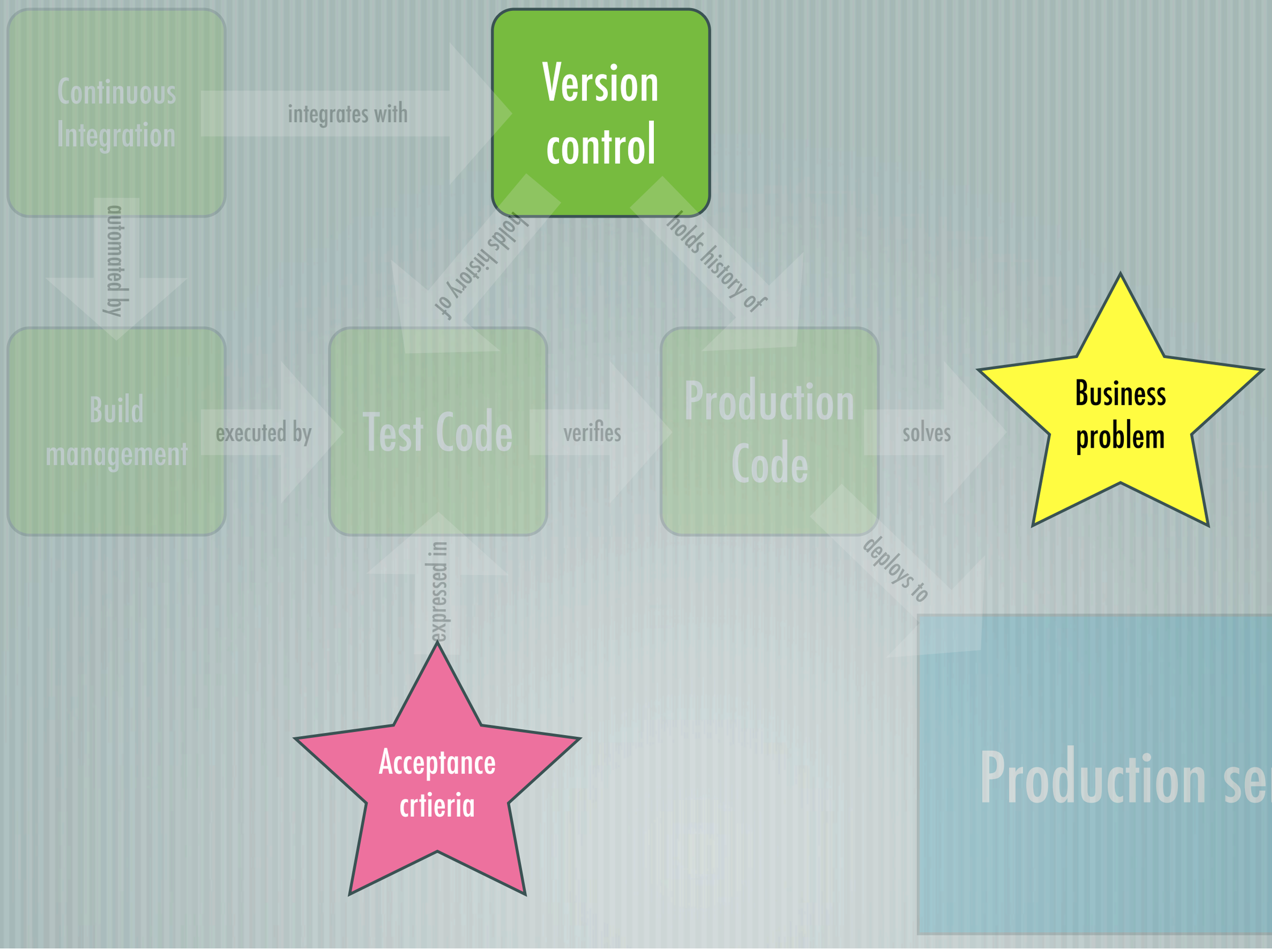
You'll get this interstitial page for a while as the code is copied in the background.



And then you should find yourself on your own version of this repository. Make sure the username is your own.

It will even tell you which repository this one was forked from.


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
Acceptance crtieria

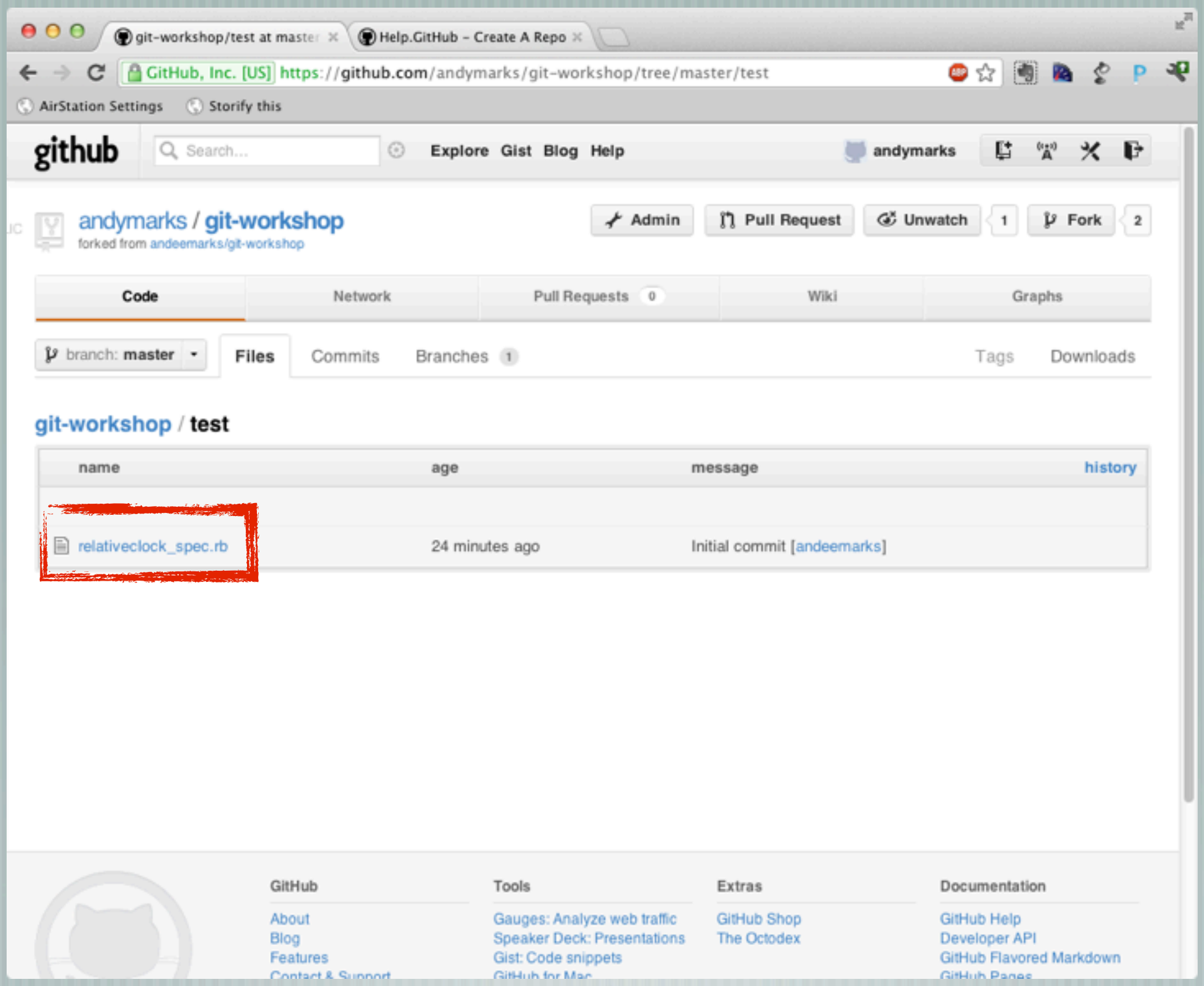

Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
deploys to
Acceptance crtieria
Production server

4. Write your tests

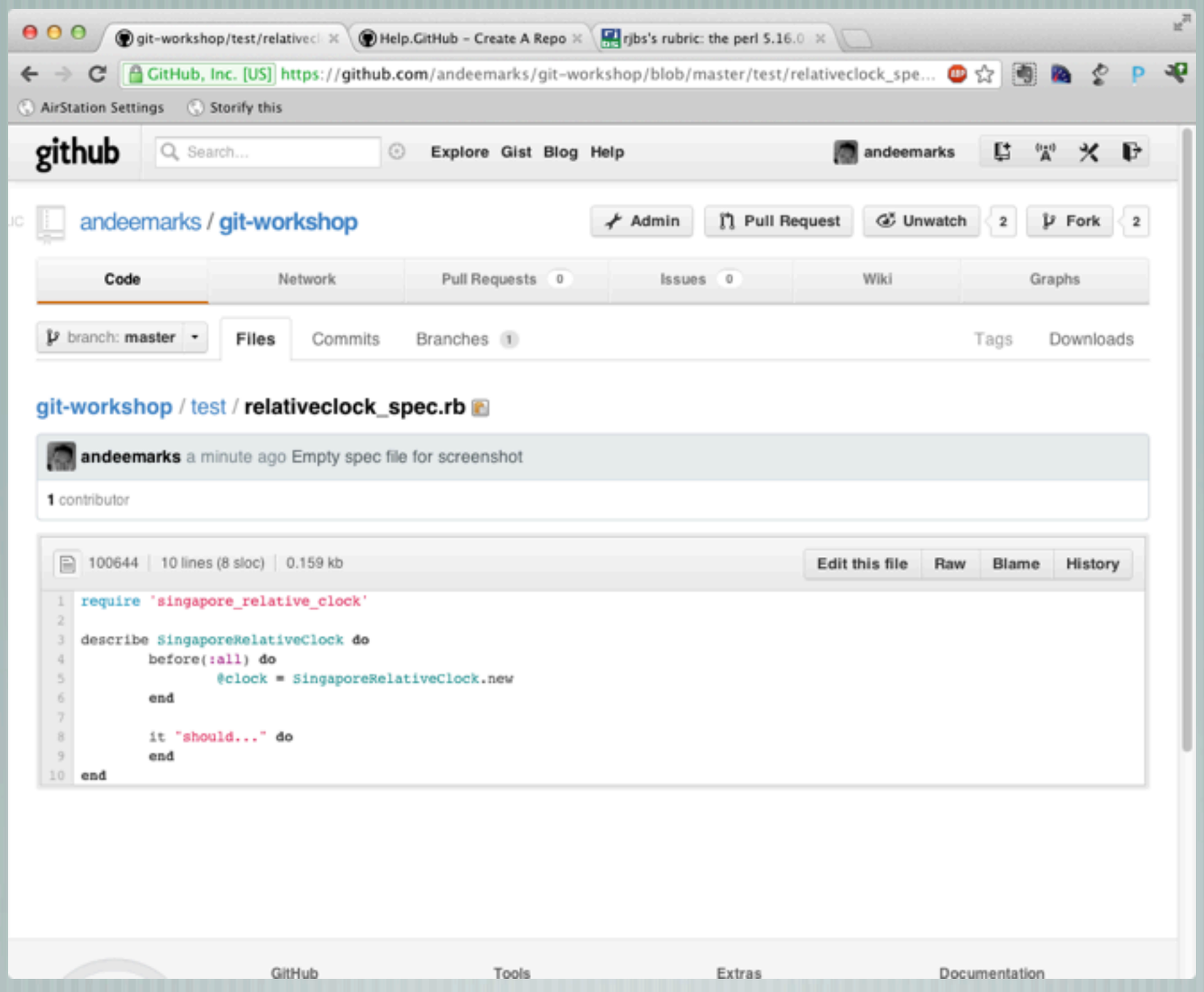
Now I mentioned before that I'll be providing a lot of the basic bits and pieces of what we'll be working on today and I'll be doing that by sharing a code repository that I have created with you.

In fact, it sounds like I've explicitly allowed this to happen but if you have a free account on GitHub then everything you host there is automatically "gettable" by anyone else. If it's code privacy you're seeking, you either need to use a paid GitHub account or run your own version of Git or some other version control system.

But by cloning this repository, you'll get your very own copy of the code under your newly created GitHub account and you'll be able change that code completely independently of any of my changes.



So now you've got the skeleton codebase, have a look at one of the files contained within. Click on the test folder and you'll see it's contents and then click on the relativeclock_spec.rb file to see what is contained within the file.



And you should see something like this, which for some of you will be your first ever look at Ruby code, so we should probably take a bit of time to go over this code at a high level.

```ruby
require 'singapore_relative_clock'

describe SingaporeRelativeClock do
        before(:all) do
                @clock = SingaporeRelativeClock.new
        end

        it "should..." do
        end
end
```


Talk about coding... text files, dynamic languages, interpreted languages

Also:
- automated testing (code verifying other code)
- rspec
- require statements
- blocks
- before/test setup
-

What's in an RSpec test?

- Test methods
- Descriptions
- Setup
- Execution
- Assertion(s)

RSpec is a commonly used framework for writing automated tests in Ruby. It's one of many, but it's well know n and a good place to start with automated testing.

Writing tests in RSpec can produce code that is generally quite easy to read, even for an inexperienced Ruby coder. And like all the code we write, we need to constantly struggle to maintain the simplicity of our code at all times.

```ruby
require 'gmt_offset_finder'

describe GMTOffsetFinder do
        before(:all) do
                @finder = GMTOffsetFinder.new
        end

        it "should find the GMT offset for a known city" do
                @finder.offsetFor("Melbourne").should_not be_nil
        end

        it "should find the GMT offset for a known city with embedded spaces" do
                @finder.offsetFor("New York").should_not be_nil
        end

        it "should find the GMT offset for a known city with surrounding spaces" do
                @finder.offsetFor("Rome  ").should_not be_nil
                @finder.offsetFor("  Rome  ").should_not be_nil
        end

        it "should treat cities in a case insensitive manner" do
                @finder.offsetFor("melbourne").should_not be_nil
                @finder.offsetFor("MELBOURNE").should_not be_nil
                @finder.offsetFor("MeLBOurNE").should_not be_nil
        end

        it "should return numeric offset for a known city" do
                @finder.offsetFor("Melbourne").should be_a_kind_of Numeric
        end

        it "should return correct offset for a known city" do
                @finder.offsetFor("Melbourne").should === 10.0
        end

        it "should error on an unknown city" do
                lambda{@finder.offsetFor("Atlantis")}.should raise_error
        end
```


Test methods.

Test methods are discrete blocks of code that represent a single test you want to execute. Each method should describe the type of test it is responsible for. The method is starts with the keyword “it” and ends with “end”. There should be one of these for each of the acceptance criteria you identified earlier.

```ruby
require 'gmt_offset_finder'

describe GMTOffsetFinder do
        before(:all) do
                @finder = GMTOffsetFinder.new
        end

        it "should find the GMT offset for a known city" do
                @finder.offsetFor("Melbourne").should_not be_nil
        end

        it "should find the GMT offset for a known city with embedded spaces" do
                @finder.offsetFor("New York").should_not be_nil
        end

        it "should find the GMT offset for a known city with surrounding spaces" do
                @finder.offsetFor("Rome  ").should_not be_nil
                @finder.offsetFor("  Rome  ").should_not be_nil
        end

        it "should treat cities in a case insensitive manner" do
                @finder.offsetFor("melbourne").should_not be_nil
                @finder.offsetFor("MELBOURNE").should_not be_nil
                @finder.offsetFor("MeLBOurNE").should_not be_nil
        end

        it "should return numeric offset for a known city" do
                @finder.offsetFor("Melbourne").should be_a_kind_of Numeric
        end

        it "should return correct offset for a known city" do
                @finder.offsetFor("Melbourne").should === 10.0
        end

        it "should error on an unknown city" do
                lambda{@finder.offsetFor("Atlantis")}.should raise_error
        end
```


Test descriptions. Descriptions can be written anyway you like, but convention says they should be phrased using should, to describe the expected behaviour.

```ruby
require 'gmt_offset_finder'

describe GMTOffsetFinder do
        before(:all) do
                @finder = GMTOffsetFinder.new
        end

        it "should find the GMT offset for a known city" do
                @finder.offsetFor("Melbourne").should_not be_nil
        end

        it "should find the GMT offset for a known city with embedded spaces" do
                @finder.offsetFor("New York").should_not be_nil
        end

        it "should find the GMT offset for a known city with surrounding spaces" do
                @finder.offsetFor("Rome  ").should_not be_nil
                @finder.offsetFor("  Rome  ").should_not be_nil
        end

        it "should treat cities in a case insensitive manner" do
                @finder.offsetFor("melbourne").should_not be_nil
                @finder.offsetFor("MELBOURNE").should_not be_nil
                @finder.offsetFor("MeLBOurNE").should_not be_nil
        end

        it "should return numeric offset for a known city" do
                @finder.offsetFor("Melbourne").should be_a_kind_of Numeric
        end

        it "should return correct offset for a known city" do
                @finder.offsetFor("Melbourne").should === 10.0
        end

        it "should error on an unknown city" do
                lambda{@finder.offsetFor("Atlantis")}.should raise_error
        end
```


Setup methods. If you find yourself repeating a lot of code inside your tests, most frameworks give you a way to isolate that code to remove the duplication. This is one of the ways RSpec does this. The code in this method will be executed once before all of the tests are run. The code itself is creating the thing which contains the behaviour we are testing.

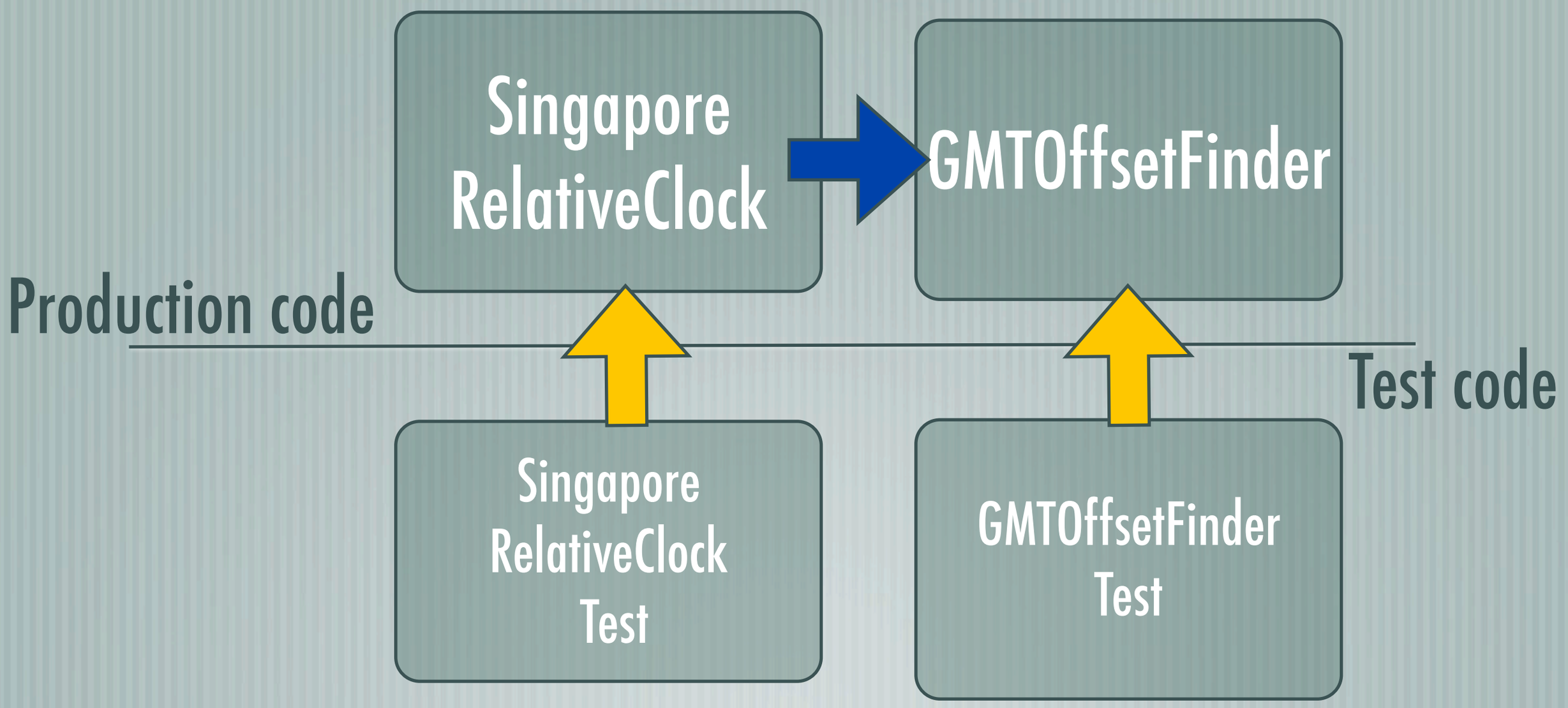
```ruby
require 'gmt_offset_finder'

describe GMTOffsetFinder do
        before(:all) do
                @finder = GMTOffsetFinder.new
        end

        it "should find the GMT offset for a known city" do
                @finder.offsetFor("Melbourne").should_not be_nil
        end

        it "should find the GMT offset for a known city with embedded spaces" do
                @finder.offsetFor("New York").should_not be_nil
        end

        it "should find the GMT offset for a known city with surrounding spaces" do
                @finder.offsetFor("Rome  ").should_not be_nil
                @finder.offsetFor("  Rome  ").should_not be_nil
        end

        it "should treat cities in a case insensitive manner" do
                @finder.offsetFor("melbourne").should_not be_nil
                @finder.offsetFor("MELBOURNE").should_not be_nil
                @finder.offsetFor("MeLBOurNE").should_not be_nil
        end

        it "should return numeric offset for a known city" do
                @finder.offsetFor("Melbourne").should be_a_kind_of Numeric
        end

        it "should return correct offset for a known city" do
                @finder.offsetFor("Melbourne").should === 10.0
        end

        it "should error on an unknown city" do
                lambda{@finder.offsetFor("Atlantis")}.should raise_error
        end
```


Test execution. And each test method should include the code to execute the behaviour you wish to verify. In each of these test methods, it's the same behaviour we're testing (the "offsetFor" method) but passing in different parameters.

```ruby
require 'gmt_offset_finder'

describe GMTOffsetFinder do
        before(:all) do
                @finder = GMTOffsetFinder.new
        end

        it "should find the GMT offset for a known city" do
                @finder.offsetFor("Melbourne").should_not be_nil
        end

        it "should find the GMT offset for a known city with embedded spaces" do
                @finder.offsetFor("New York").should_not be_nil
        end

        it "should find the GMT offset for a known city with surrounding spaces" do
                @finder.offsetFor("Rome  ").should_not be_nil
                @finder.offsetFor("  Rome  ").should_not be_nil
        end

        it "should treat cities in a case insensitive manner" do
                @finder.offsetFor("melbourne").should_not be_nil
                @finder.offsetFor("MELBOURNE").should_not be_nil
                @finder.offsetFor("MeLBOurNE").should_not be_nil
        end

        it "should return numeric offset for a known city" do
                @finder.offsetFor("Melbourne").should be_a_kind_of Numeric
        end

        it "should return correct offset for a known city" do
                @finder.offsetFor("Melbourne").should === 10.0
        end

        it "should error on an unknown city" do
                lambda{@finder.offsetFor("Atlantis")}.should raise_error
        end
```


Test assertions. The most important part of good automated tests is how they verify the behaviour they expect through the use of assertions. Working out how to code assertions in a dynamic language like Ruby can be a bit tricky, so I've included a cheat sheet of RSPec assertions for you guys.

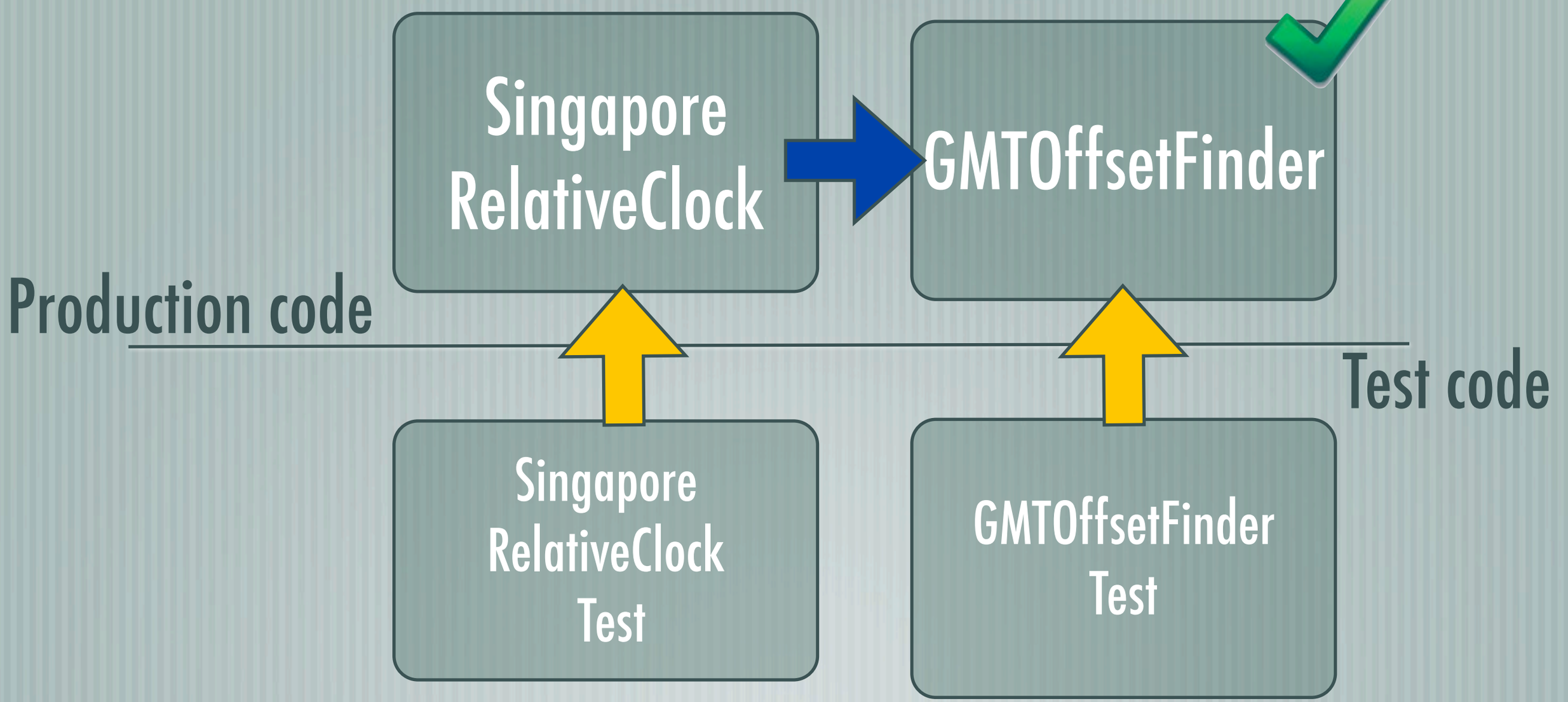
Where does this fit in?



So where do these tests fit into what we're doing. We'll you may have worked out by looking at what behaviour the tests were expected, but the GMTOffsetFinder is a piece of Ruby code that's already in your repository and does some of the heavy lifting for our relative clock.

Knowing how that code works shouldn't change the sort of tests you need to write for the SingaporeRelativeClock, but it will change the way you make those tests pass, which is our next task.

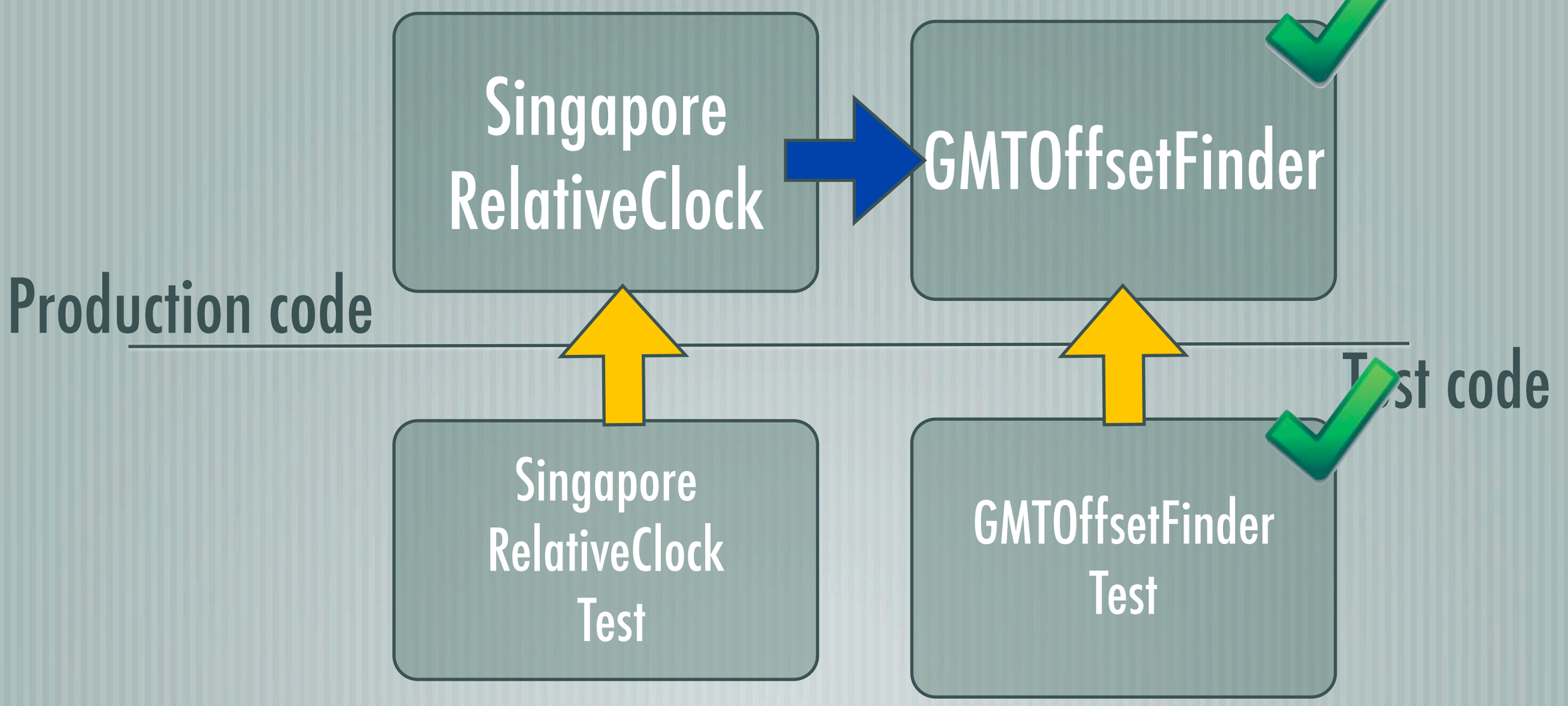
Where does this fit in?



So where do these tests fit into what we're doing. We'll you may have worked out by looking at what behaviour the tests were expected, but the GMTOffsetFinder is a piece of Ruby code that's already in your repository and does some of the heavy lifting for our relative clock.

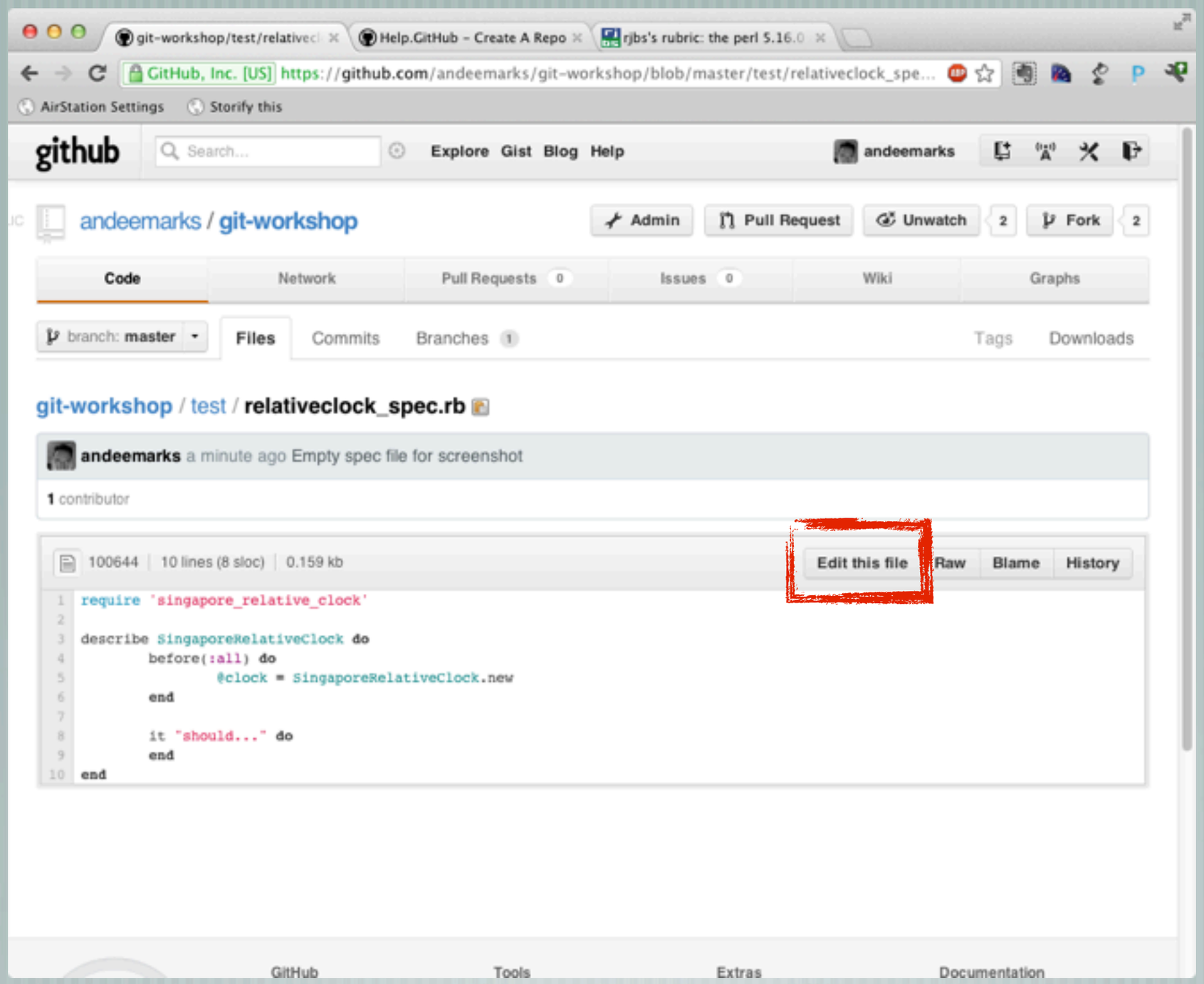
Knowing how that code works shouldn't change the sort of tests you need to write for the SingaporeRelativeClock, but it will change the way you make those tests pass, which is our next task.

Where does this fit in?

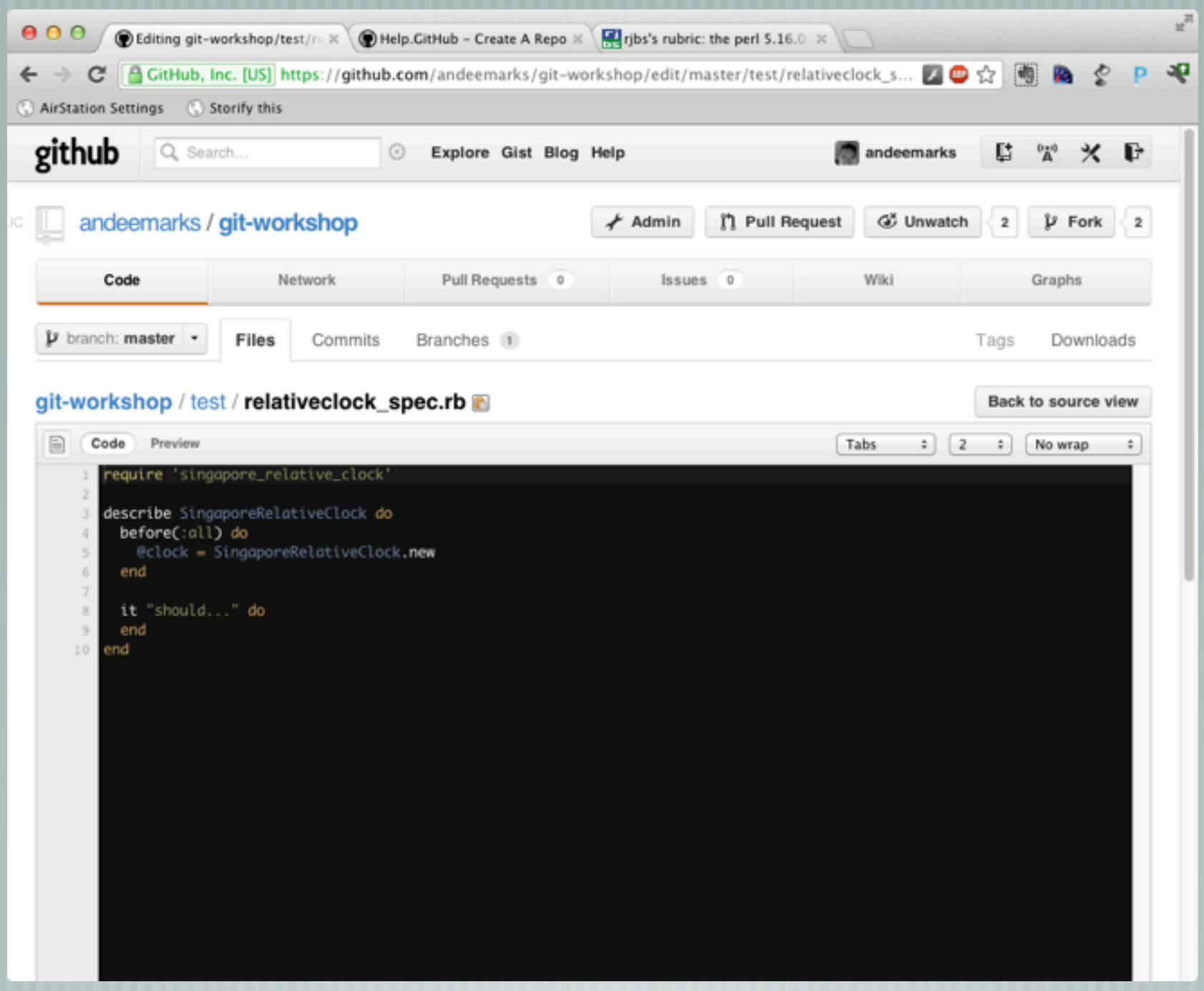


So where do these tests fit into what we're doing. We'll you may have worked out by looking at what behaviour the tests were expected, but the GMTOffsetFinder is a piece of Ruby code that's already in your repository and does some of the heavy lifting for our relative clock.

Knowing how that code works shouldn't change the sort of tests you need to write for the SingaporeRelativeClock, but it will change the way you make those tests pass, which is our next task.

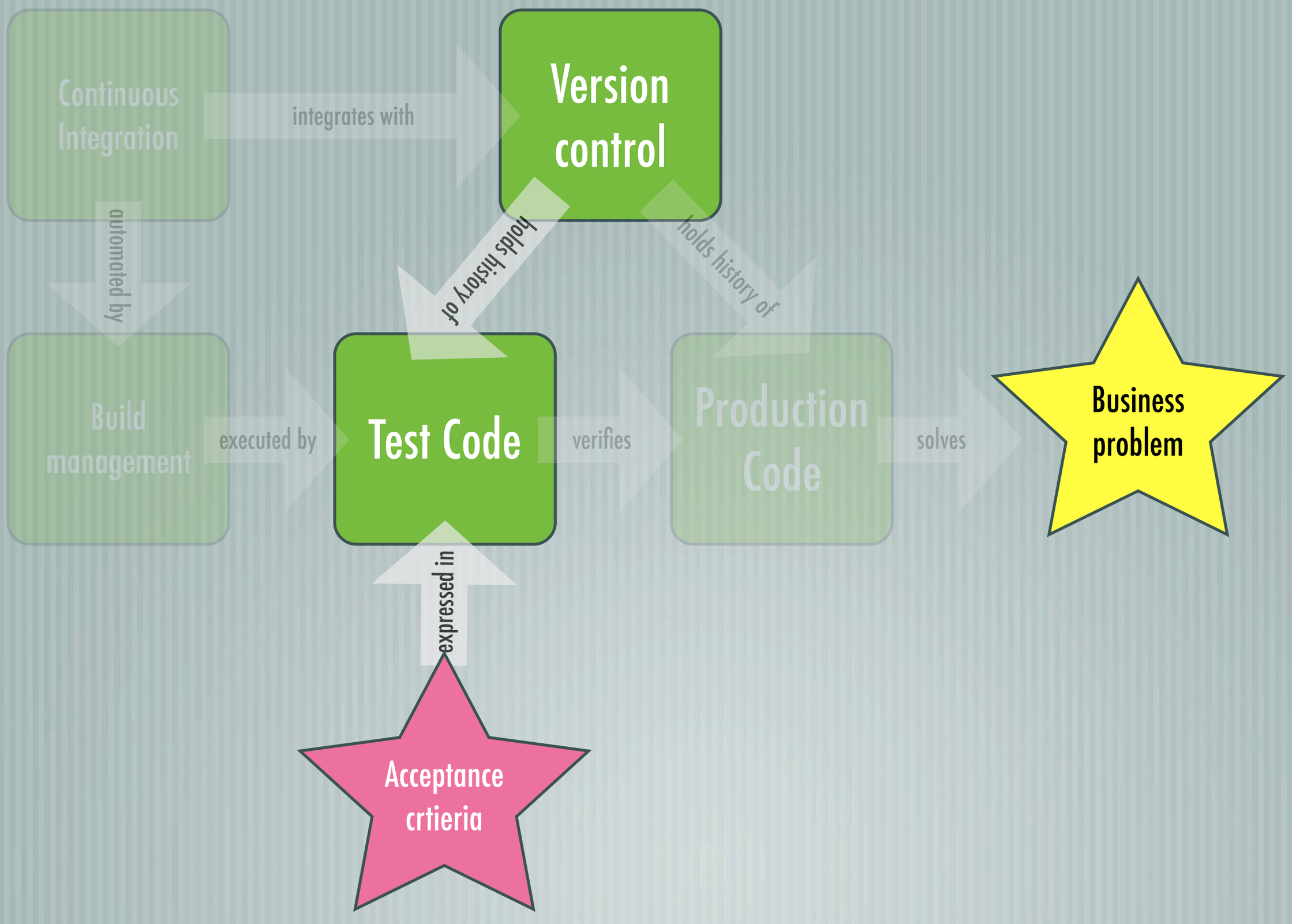


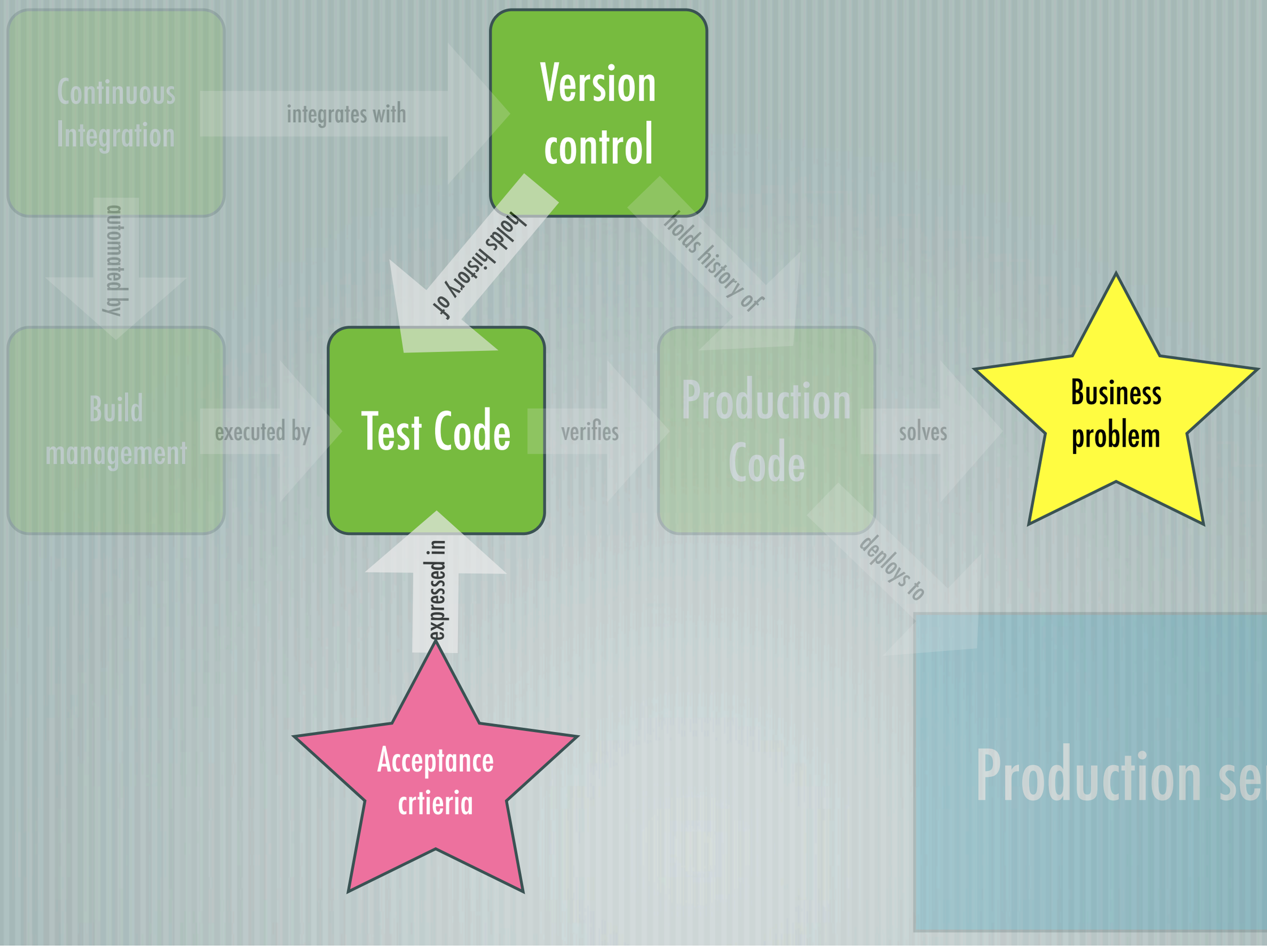
So once you have a good set of ways to test your application, we'll type them in so we don't forget about them. The way to do this is to click on the Edit this file link which will bring up a basic in-browser editor for this file...

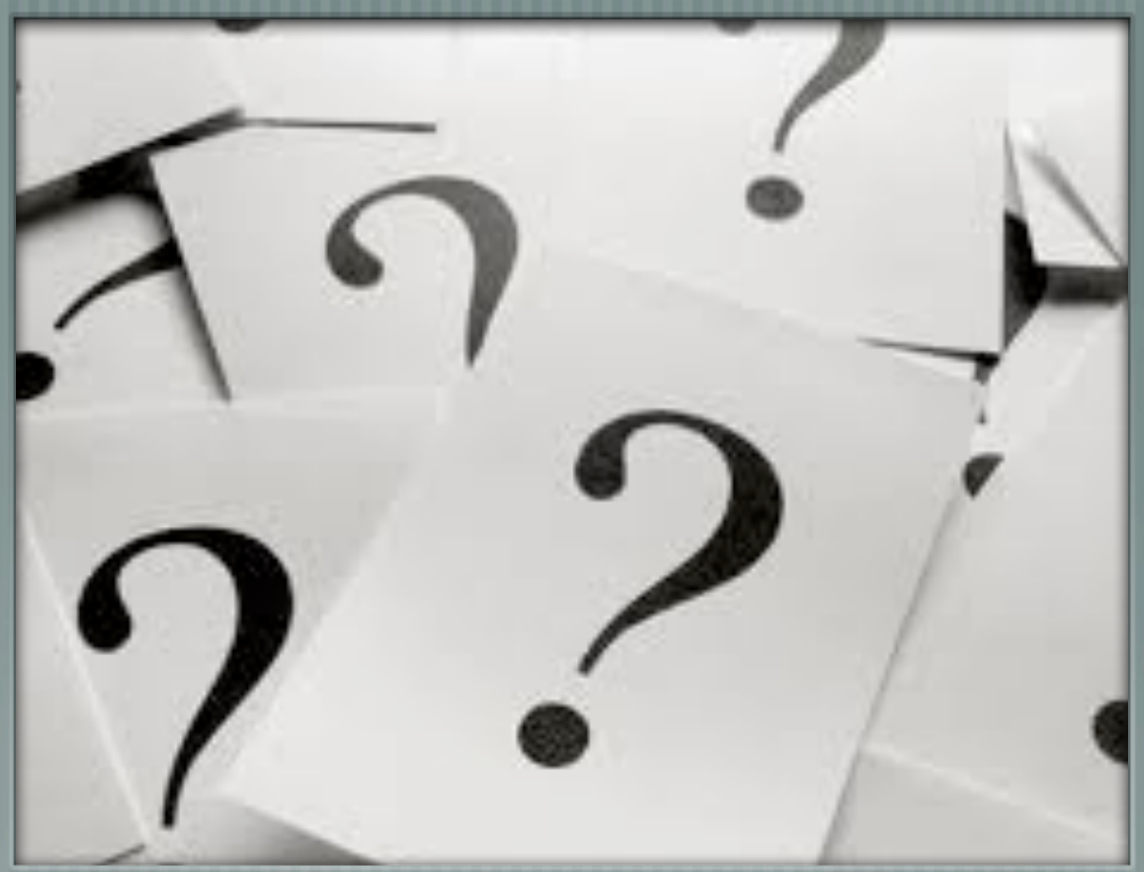


So type in your tests here.

Don't worry too much about what goes in between the begin...end bits, just focus on getting all the information inside the it "should..." bits. When you've finished typing them in, click the Commit Changes button down the bottom.


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
Acceptance crtieria


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
Acceptance crtieria
deploys to
Production server



But does it work?

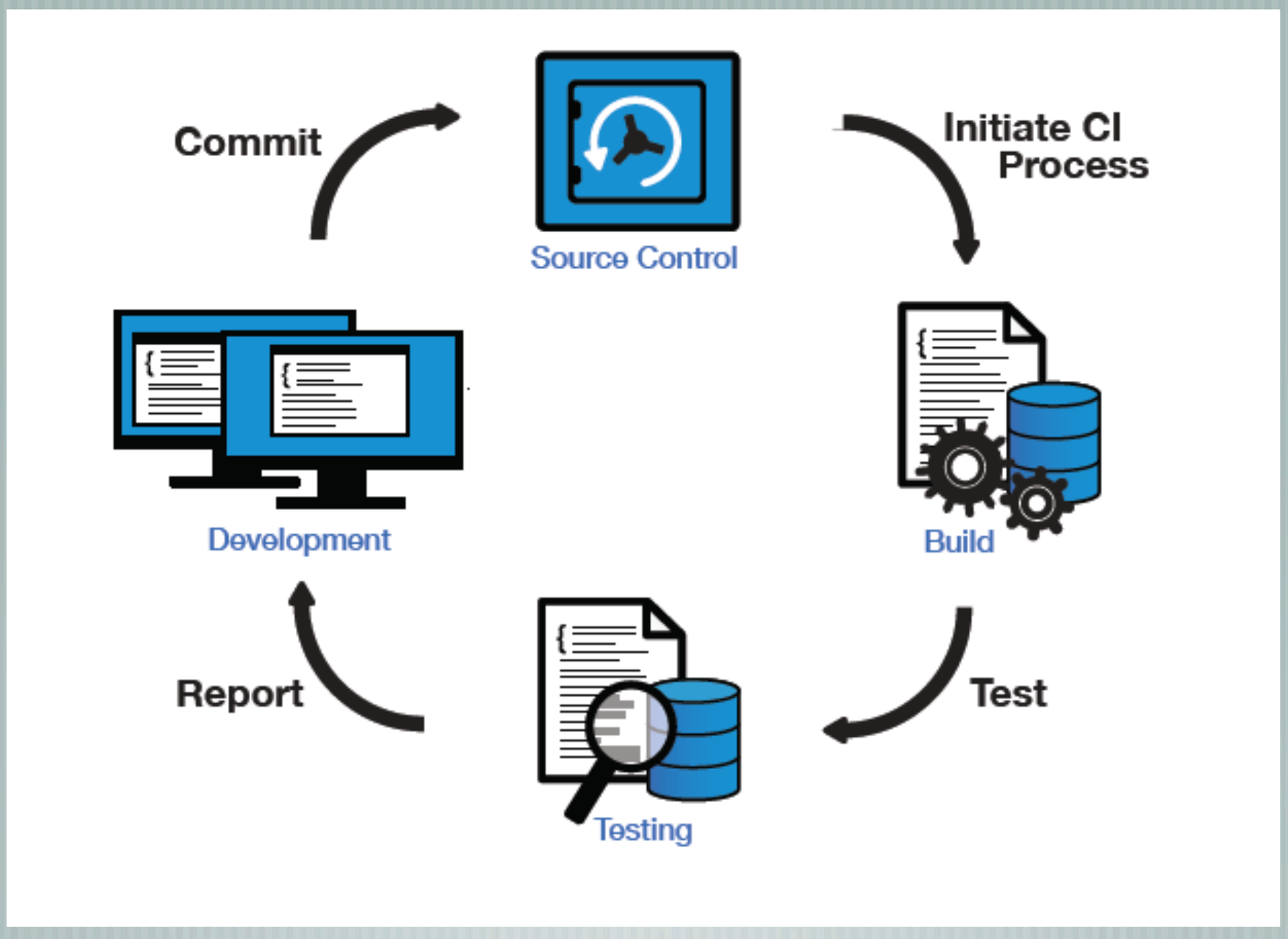
Now one of the downsides with me trying to build a workshop where you don't need a development environment running on your own laptop is that you really don't have a way of finding out whether your tests are working or not... although at this point in time the answer is almost definitely "no" because you haven't written any code to make the tests work... but we'll get to that in due time.

Thankfully, some very nice people have made it quite easy for us to test our code without using our own machines... which is sometimes a very good thing.

5. Run the tests

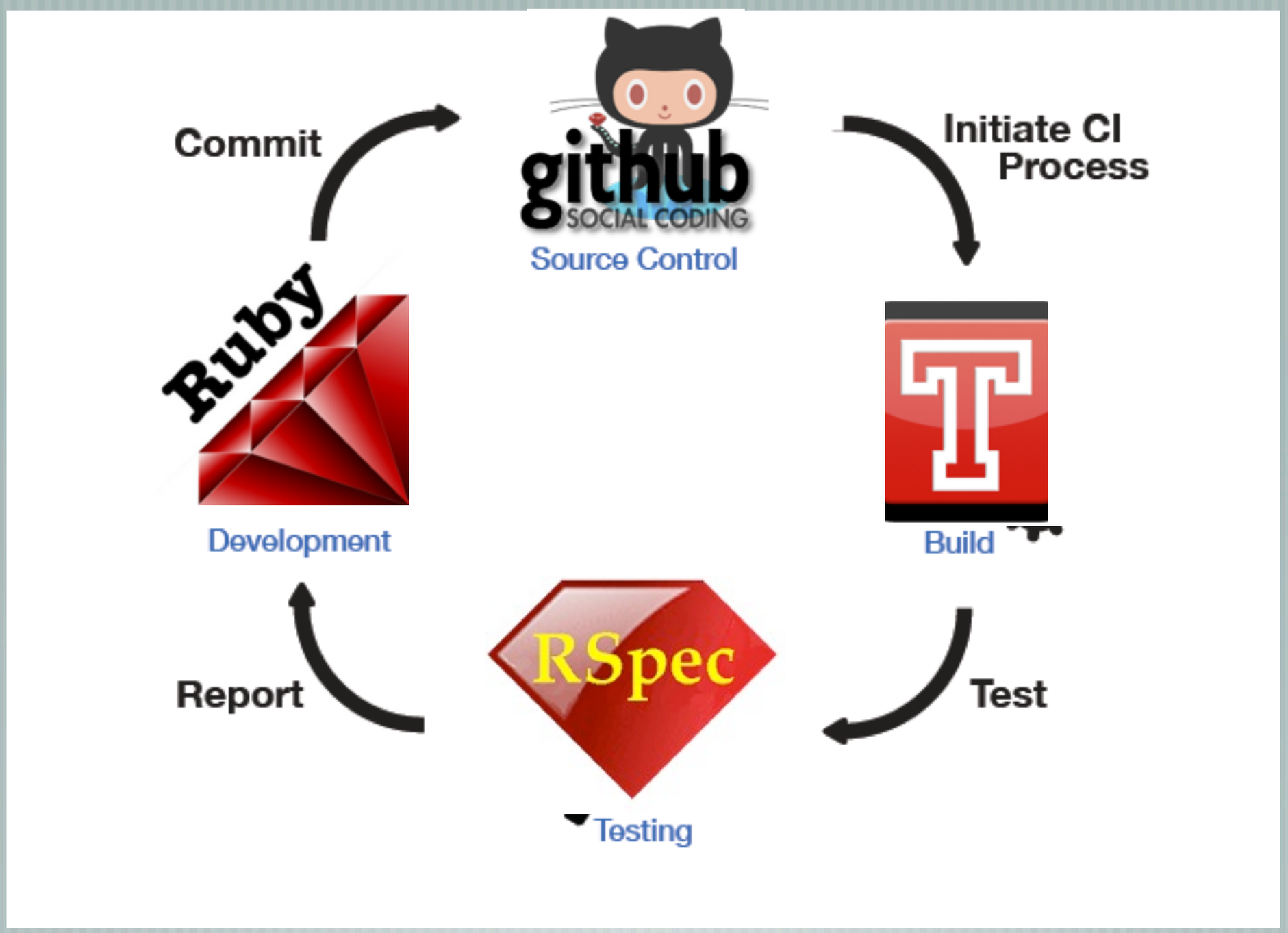
http://travis-ci.org

And one of those companies is called Travis-CI. Travis CI is a web-based service that allows you to automatically run tests against code. It's particularly well disposed towards Ruby (which is good) and looks even more favourably on GitHub (even more useful).



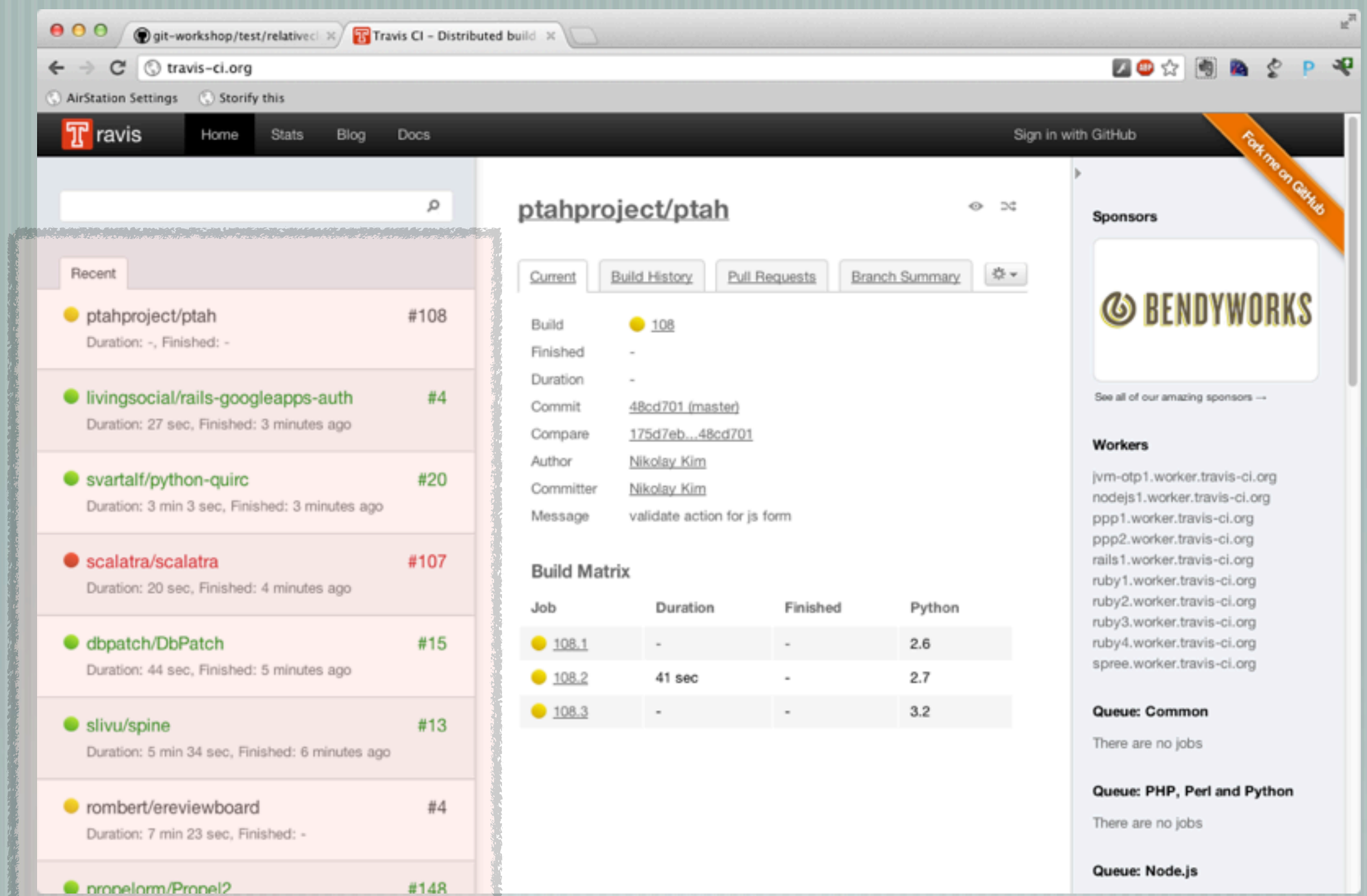
I'm going to take a little diversion from our work here to explain to you how CI, which stands for Continuous Integration, works. There are many CI solutions available on the market (Travis CI is just one), but they all support the same basic process.

All CI tools monitor Source Control systems (GitHub in our case) for changes in code triggered by developers commiting updates, just like we did by editing our test file to add in our new tests. When the CI system detects these changes it will start off a process to build and test the software that has changed and then report the results.



We'll use Travis for exactly this behaviour over the next set of slides.

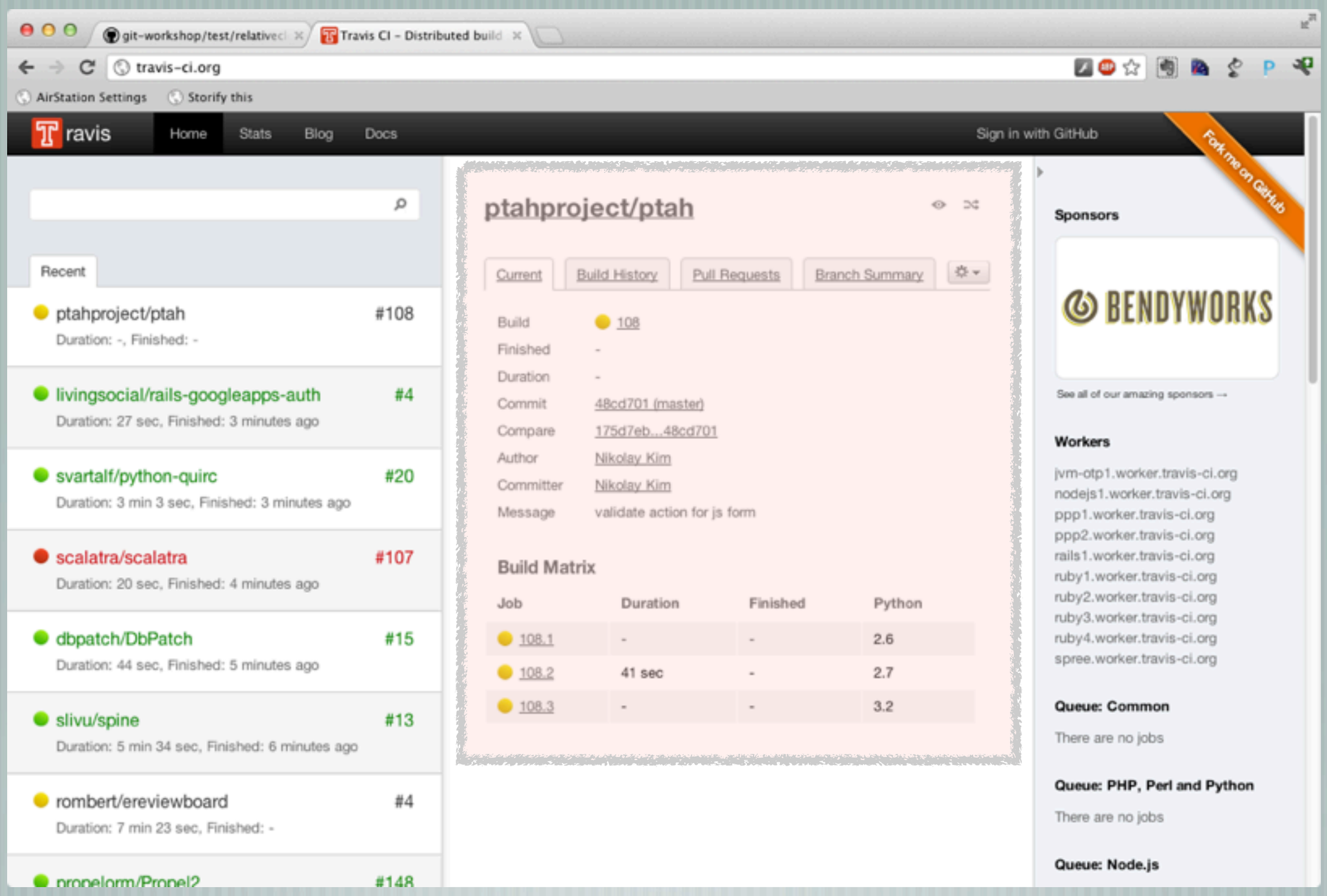
Our Source Control system is GitHub.
Our build system is Travis CI itself.
We've written our tests using a tool called RSpec.
The code we're testing is being done in Ruby.



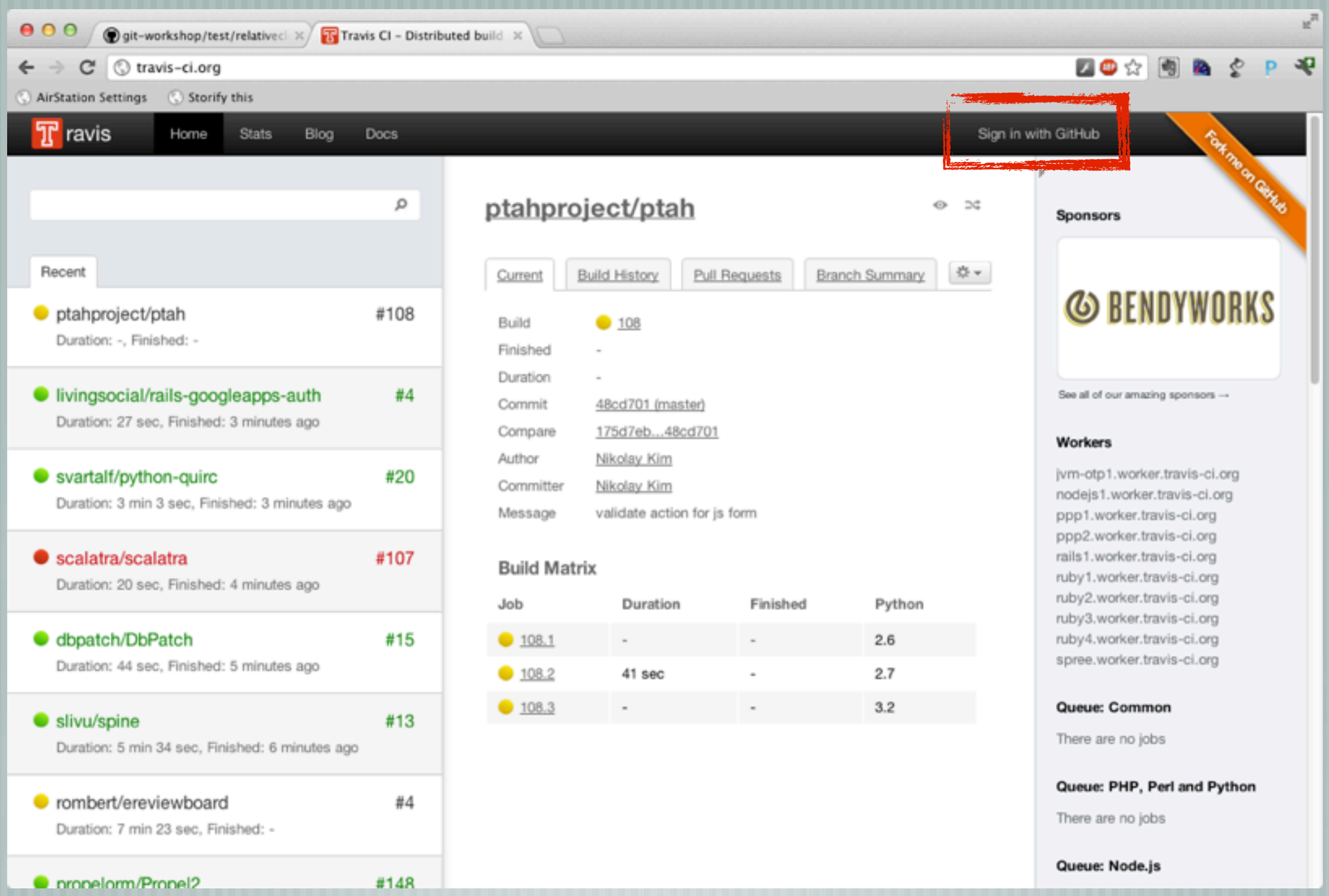
When you first visit the Travis homepage, there's a lot of information visible, so we'll have a look at the most relevant bits just to get familiar with it.

Firstly, the left hand panel shows all the projects that have been built by Travis recently, from most recent to least recent. Colour is important; green means the build/test cycle passed, red means it failed and yellow means the result was unknown. The number on the right of each project was how many times that project had been build by Travis since it was first introduced.

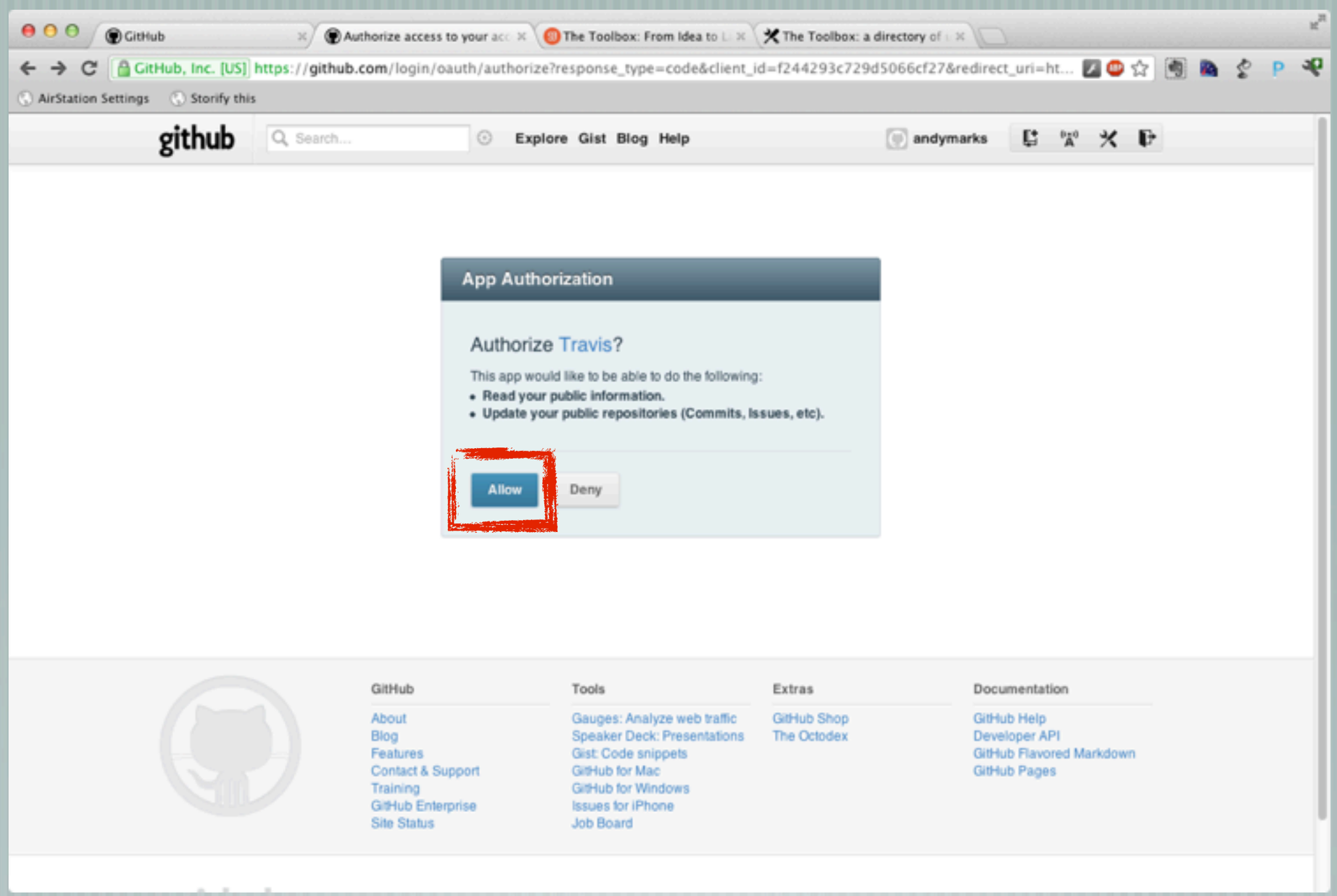
And there's some more metadata about the build for each project in smaller font under the project name.



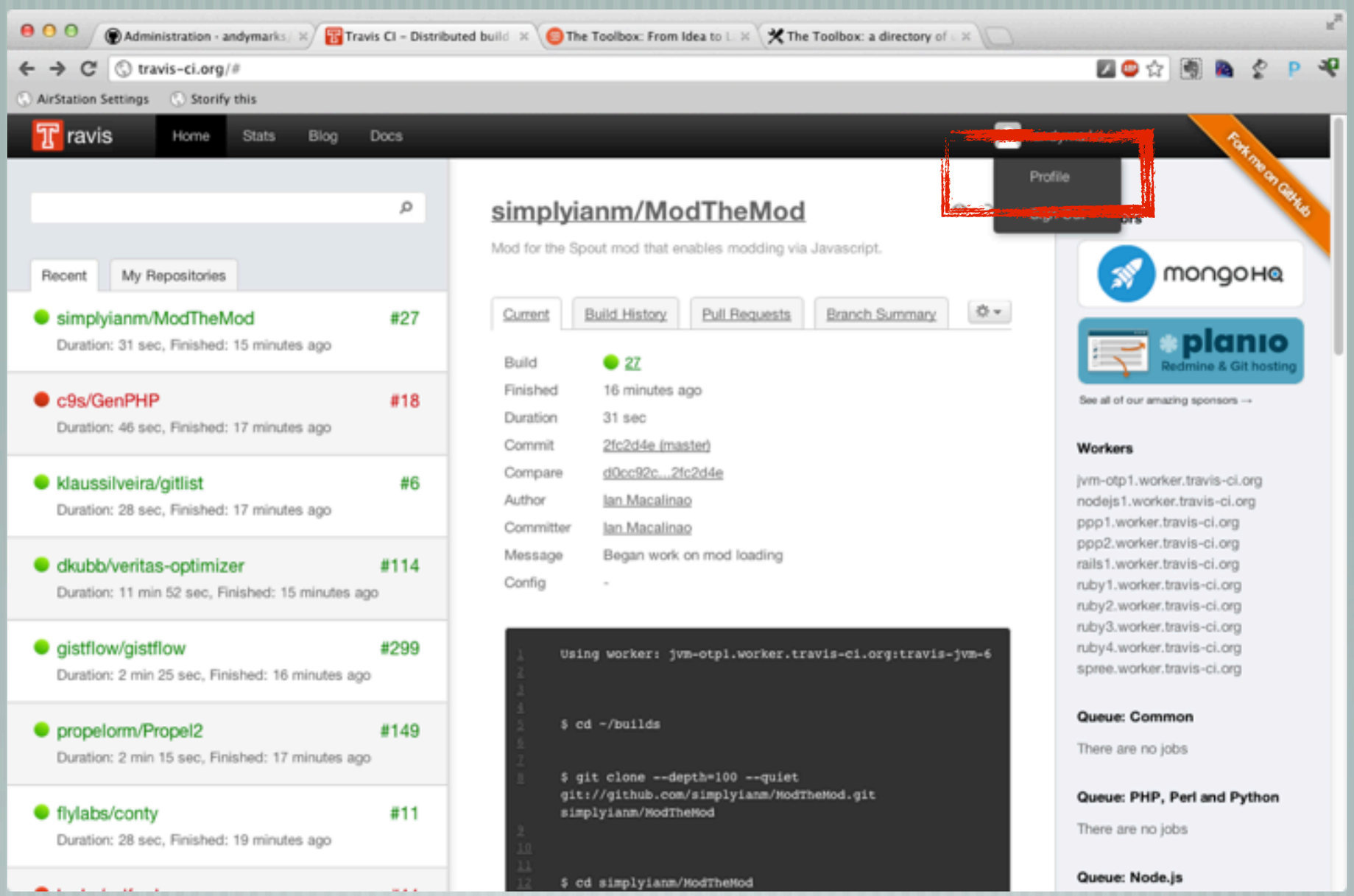
The central panel shows more details for the current or selected project on the right. You can click through the tabs and see more stuff there if you wish, but we'll get a bit closer to Travis in a short while, so there's no great hurry.



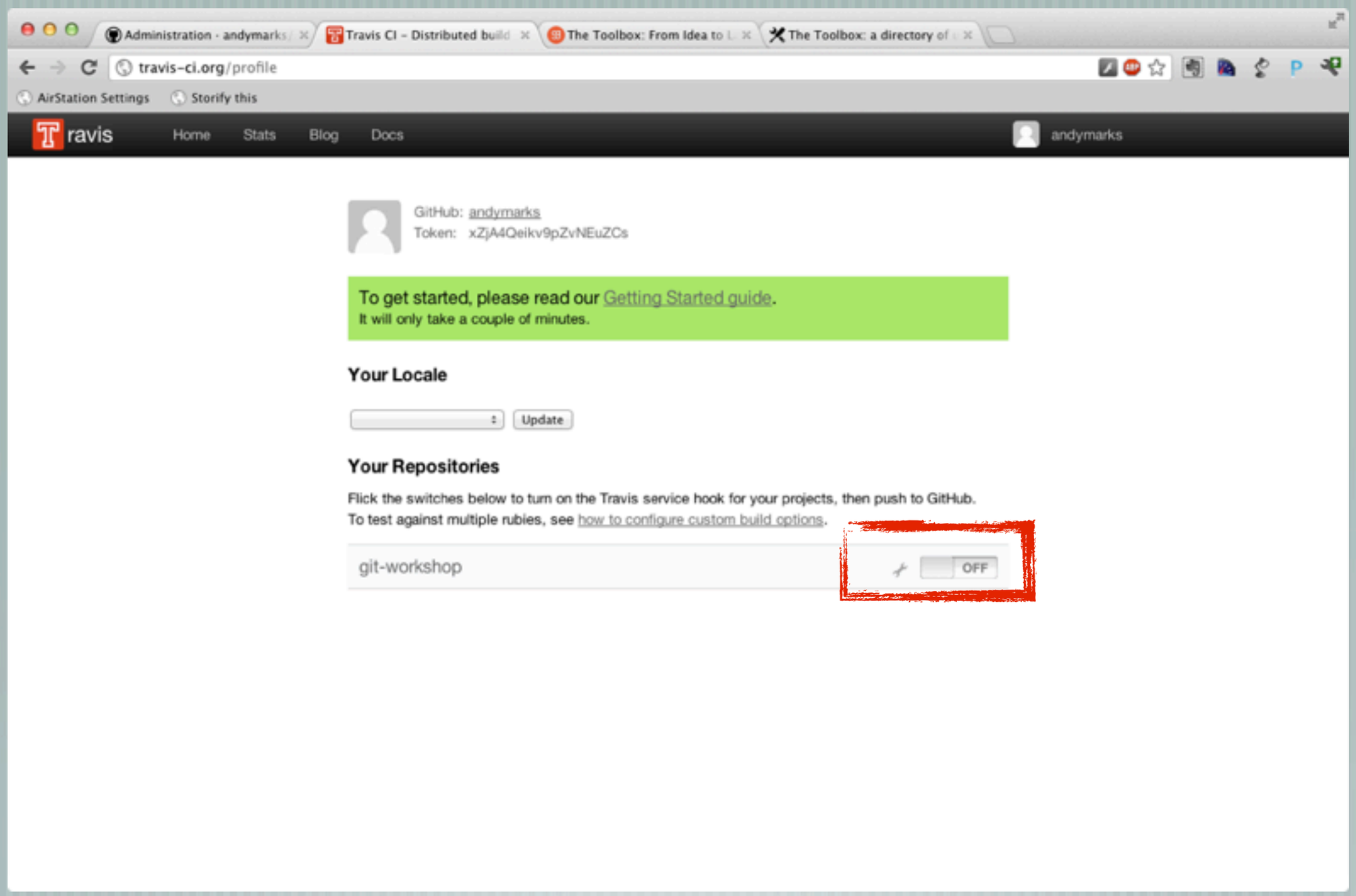
Let's start hooking our code up to Travis but clicking the Sign in with GitHub button. Remember how I told you Travis was very closely integrated with GitHub? Because you've all created your GitHub accounts, you get automatic access to Travis as well through the same credentials.



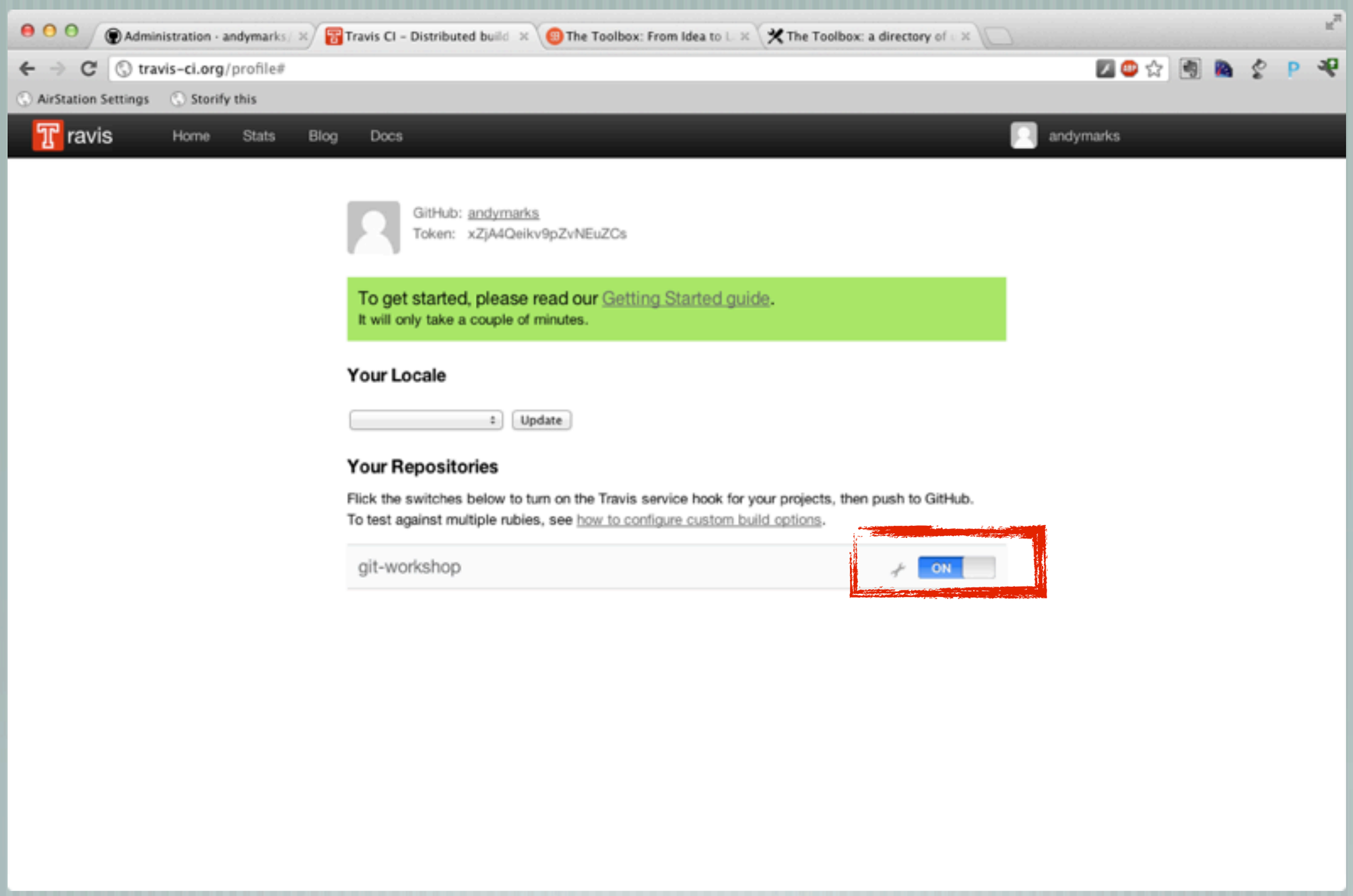
When you click on that button, you should get this warning screen asking you to allow GitHub and Travis to share the information they need to work together seamlessly. Click Allow.



You'll then be brought back to the Travis homepage, but your GitHub username will appear in the top Nav. Click on the username and then the Profile option underneath that menu...



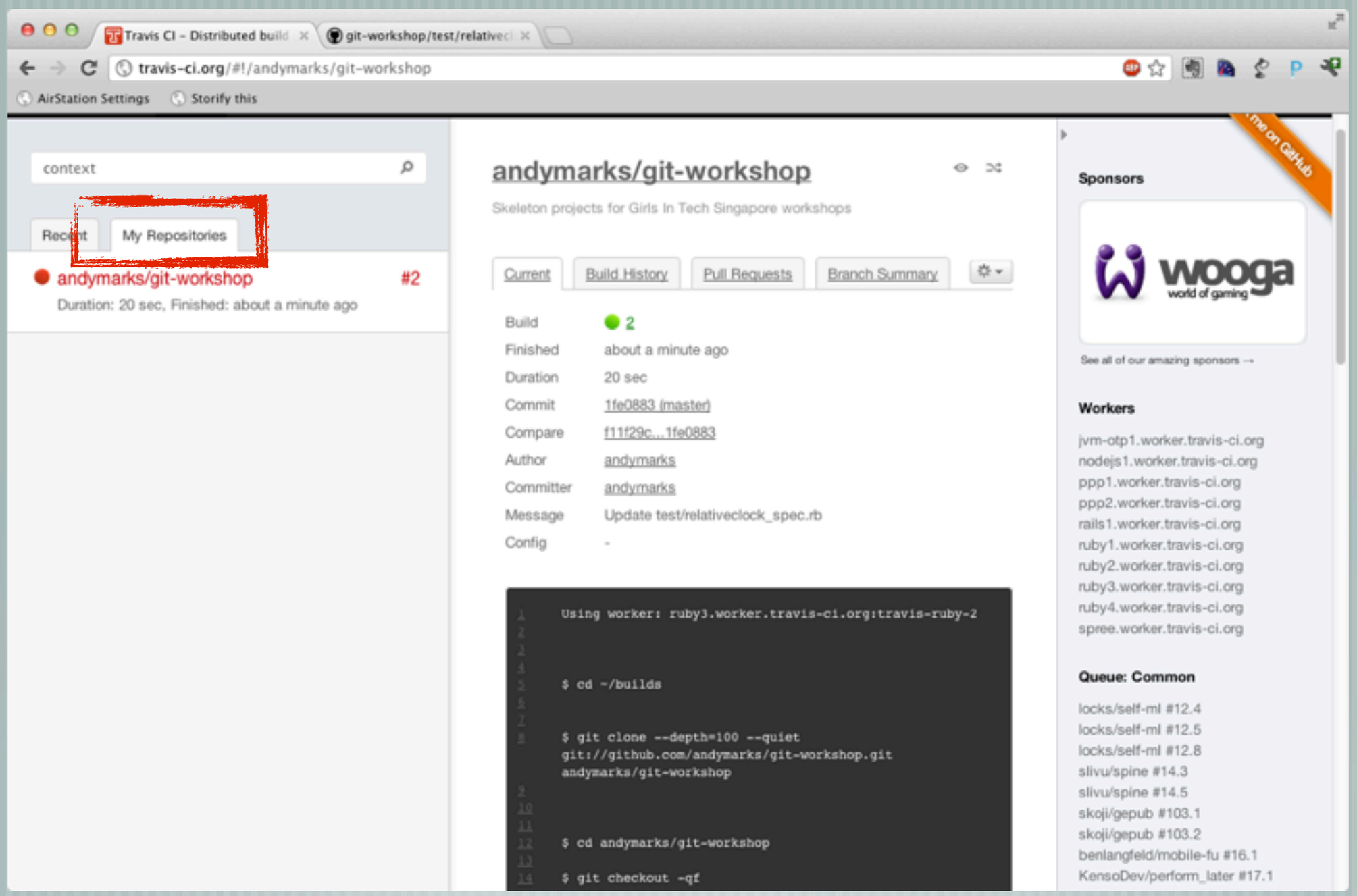
Now you should see a page containing a list of all the code repositories associated with your GitHub account. You will only have 1 at this point.



Toggle the On/Off button, so that it shows On. By doing this, you are telling Travis CI to build this project when the code next changes.

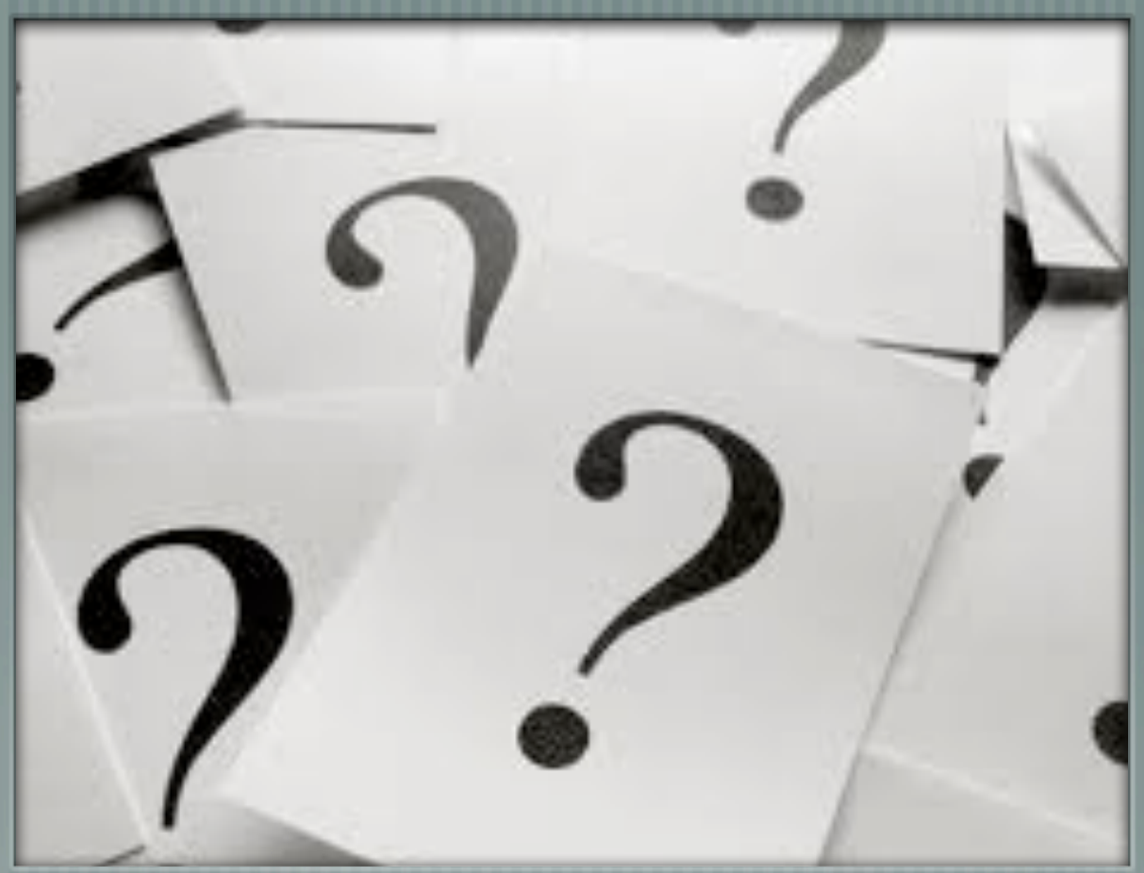
Now change something!

Having configured our project to be built by Travis, we now need to make some changes to the codebase in GitHub to poke Travis CI into action. So edit one of the files we've looked at through GitHub and then come back to Travis after you've finished.



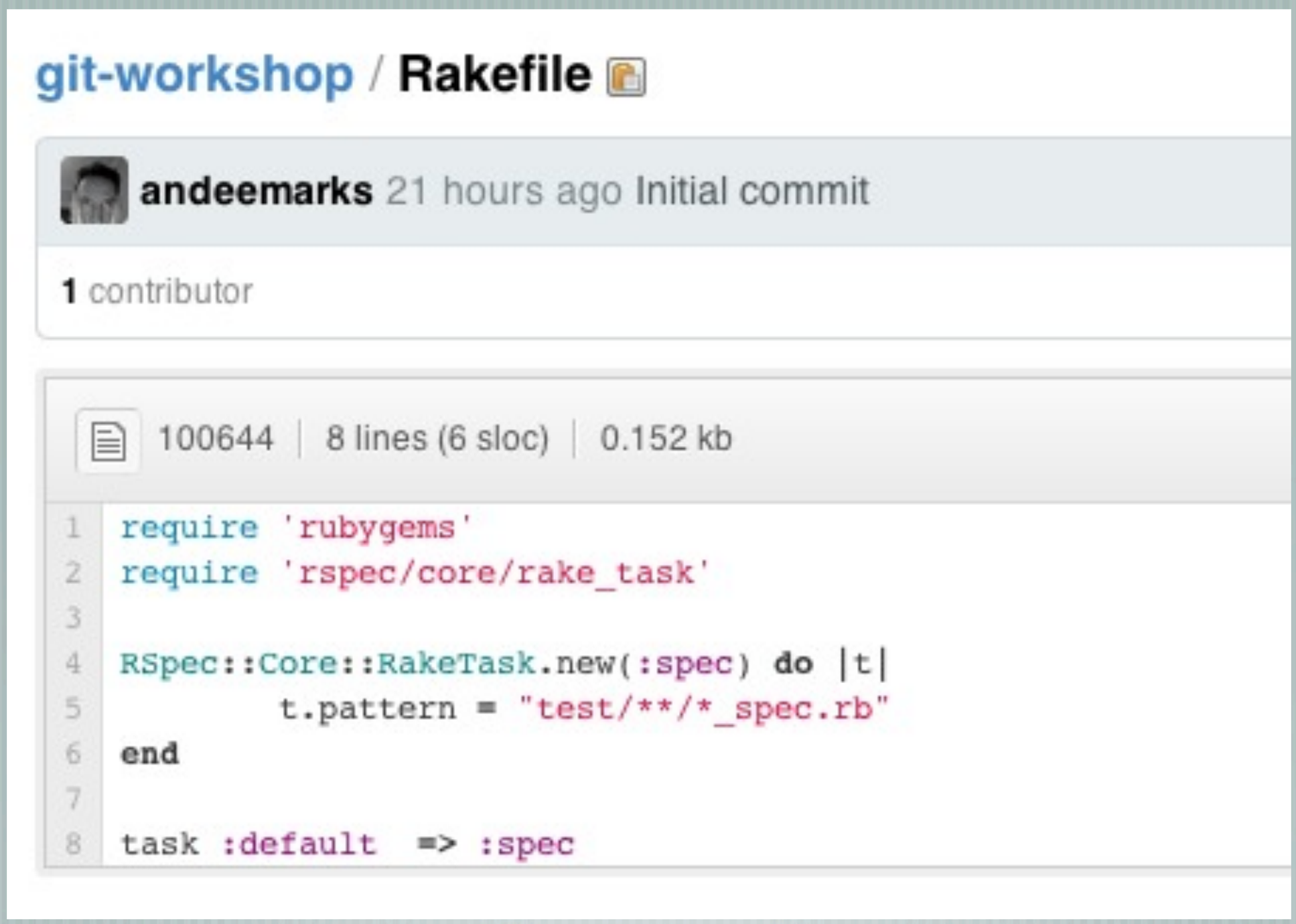
If you now return to the Travis CI homepage and click on the My Repositories tab, you should see the repository you just enabled listed there. You may even see that Travis has build the repository by the colour associated with it.

I should caution everyone at this point that this activity does not happen instantaneously and it sometimes takes a few minutes between enabling a repository and Travis getting around to building it... so be patient.



What just happened?

So if you've reached the point where Travis CI has built (and failed) your project, then it's high time we look at what role programming had in this activity. Yes, it's true that we've written some tests for our code and possibly even some code to make the tests pass, but there's another piece of code I've hidden from you now that was just executed by Travis.



Go back to your GitHub page and have a look at the file called Rakefile.

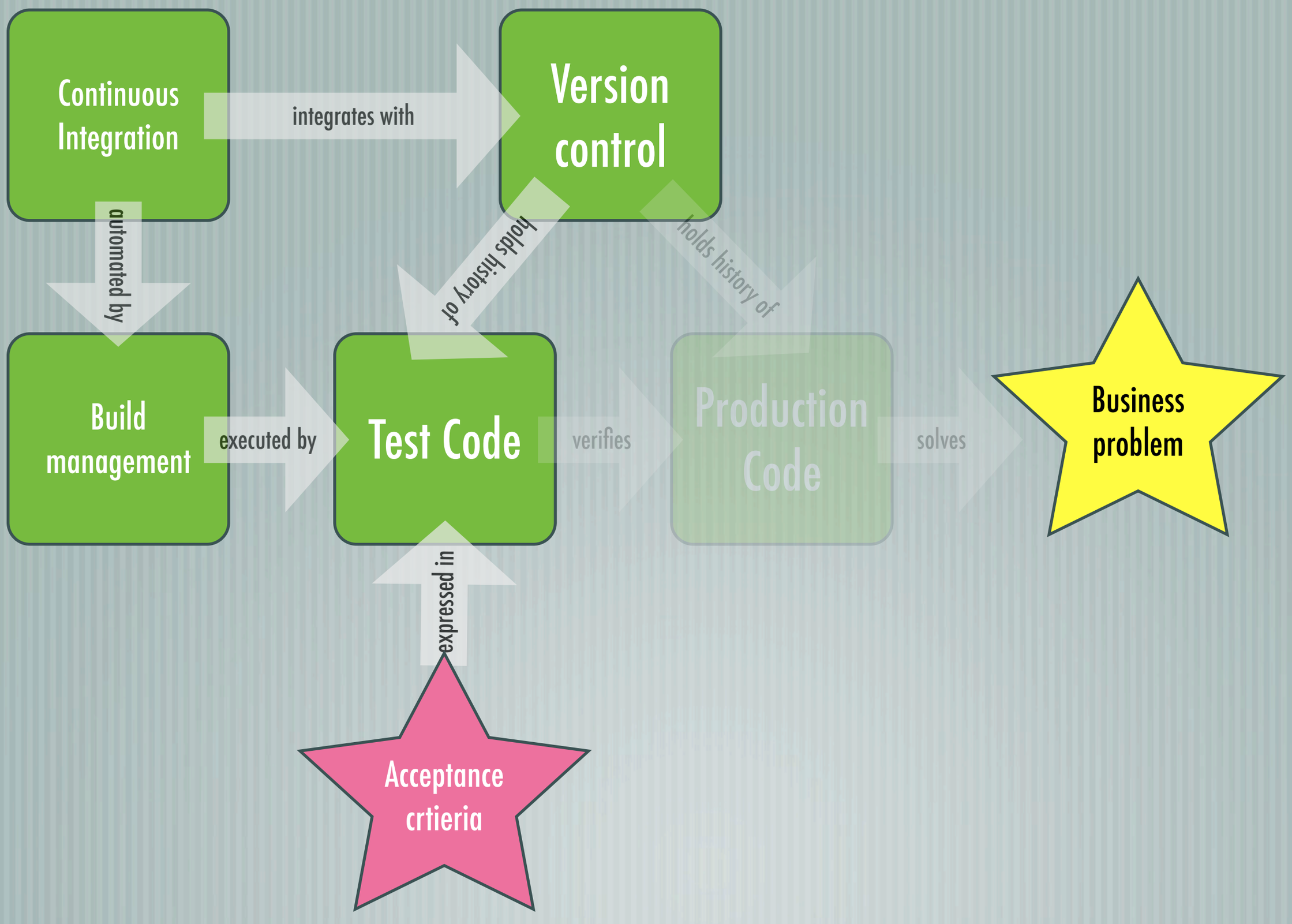
This is another piece of ruby code, but it's not for any of the purposes we've looked at to date.

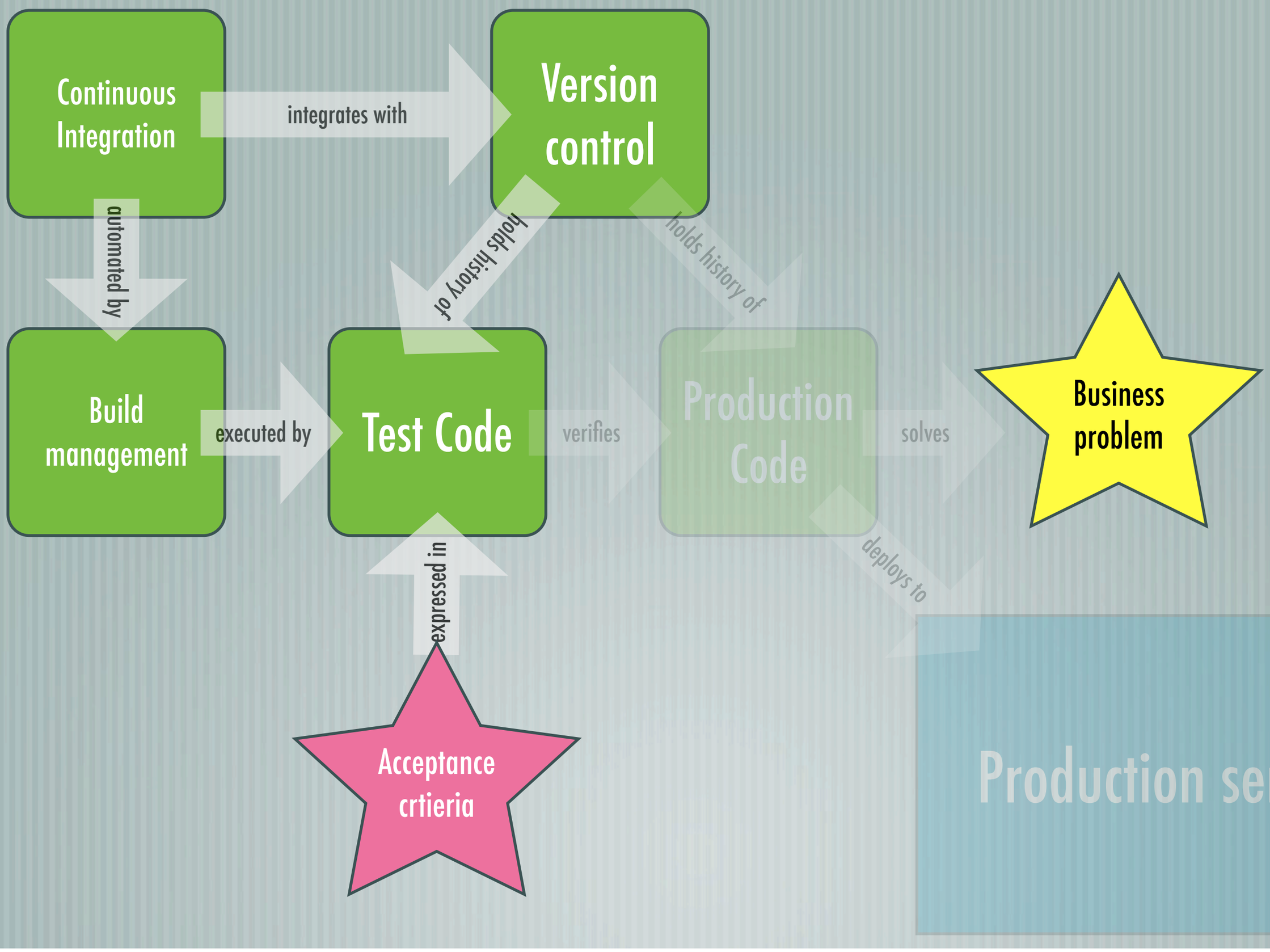
- It's not a test file assuring our production code is working
- It's not production code we're using to solve our business problem.
- **It's support code to help us run the test code against the production code.**

Rake is a special type of Ruby code that allows you to specify how to do things with the source code of your application, in this case lines 4-6 specify that we have a set of test files that live below the test directory and are all named with a similar pattern. These files are all grouped together into a category of RSpec tests that the Rake system knows about.

Long story short: if you type "rake spec" (or even just "rake") at the command line, it will run all the tests you have written.

There is a lot more stuff that you would put into a typical Rakefile for a decent sized Ruby project, but we don't need to worry about that now. The important take away is that this is another example of how we've used development to automated manual tasks.


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
Acceptance crtieria


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
deploys to
Acceptance crtieria
Production server

```
{15:42}~/Code/ruby/git-workshop:master ✓ ➪ rake spec
/Users/amarks/.rvm/rubies/ruby-1.9.3-p0/bin/ruby -S rspec test/gmt_offset_finder
_spec.rb test/relativeclock_spec.rb

GMTOffsetFinder
  should find the GMT offset for a known city
  should find the GMT offset for a known city with embedded spaces
  should find the GMT offset for a known city with surrounding spaces
  should treat cities in a case insensitive manner
  should return numeric offset for a known city
  should return correct offset for a known city
  should error on an unknown city

SingaporeRelativeClock
  should return 0 if the city is in the same timezone as Singapore
  should return a negative number if the city is ahead of Singapore's timezone
  should return the difference in hours
  should return a positive number if the city is behind of Singapore's timezone
  should return an error if the city is unrecognized
  should return an error if the city is missing

Finished in 0.03324 seconds
13 examples, 0 failures
{15:42}~/Code/ruby/git-workshop:master ✓ ➪
```


Here's what running "rake spec" looks like on my machine. You can see a bunch of stuff here...

- Green text. Descriptions of passing test (sorry, no failures here)
- White headings. The names of the files containing the two sets of tests.
- Some summary data about the test run

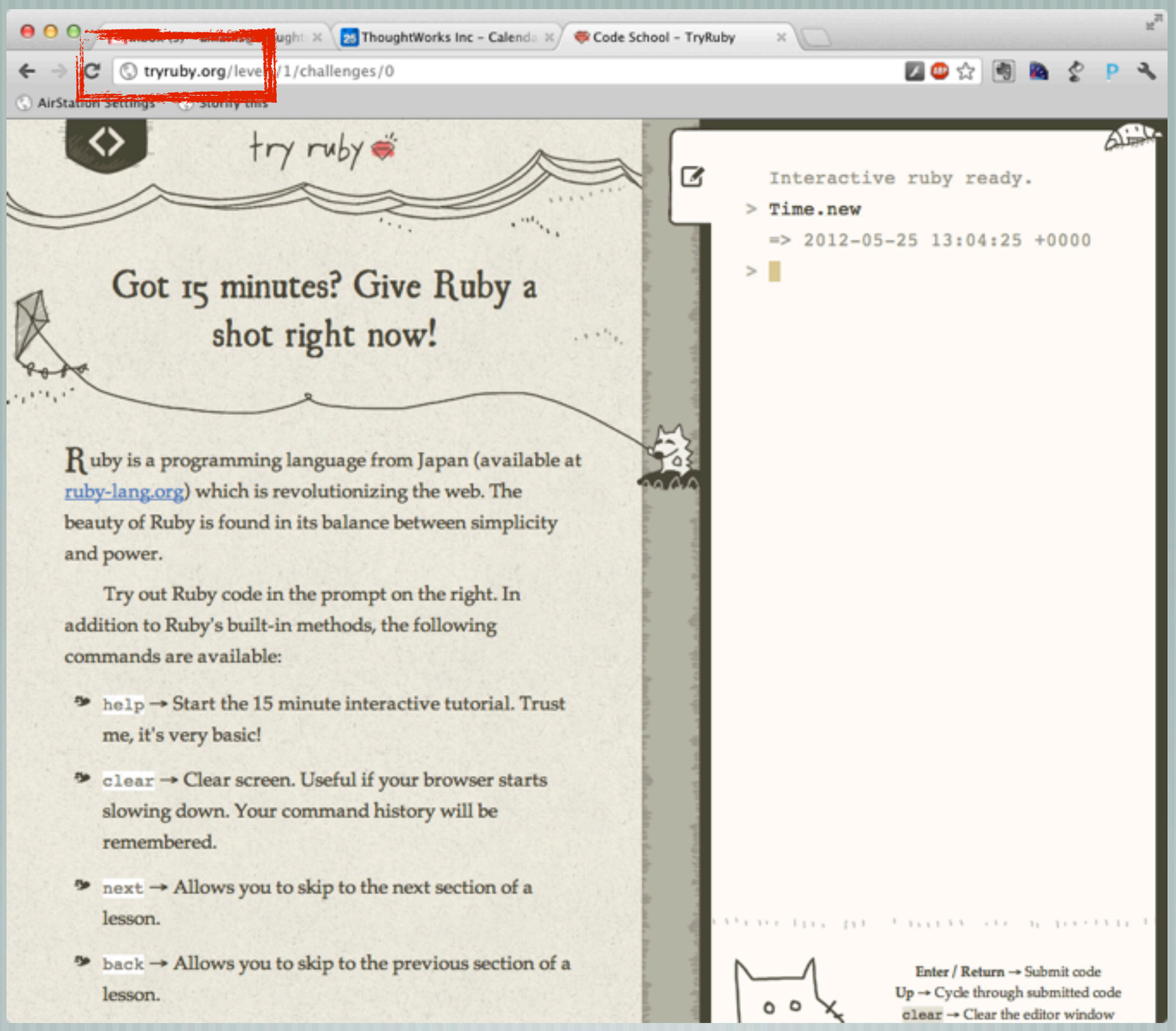
```
12  $ cd andeemarks/git-workshop
13
14  $ git checkout -qf 6b41281e235a6f5bb44989ec6b6ead8abf35b029
15
16
17  $ export TRAVIS_RUBY_VERSION=1.9.3
18
19  $ rvm use 1.9.3
20  Using /home/vagrant/.rvm/gems/ruby-1.9.3-p125
21
22  $ ruby --version
23  ruby 1.9.3p125 (2012-02-16 revision 34643) [i686-linux]
24
25  $ gem --version
26  1.8.17
27
28
29  $ export BUNDLE_GEMFILE=/home/vagrant/builds/andeemarks/git-workshop/Gemfile
30
31  $ bundle install
44  Your bundle is complete! Use `bundle show [gemname]` to see where a bundled gem is
    installed.
45
46  $ bundle exec rake
47  /home/vagrant/.rvm/rubies/ruby-1.9.3-p125/bin/ruby -S rspec test/relativeclock_spec.rb
48  .............
49
50  Finished in 0.04702 seconds
51  13 examples, 0 failures
```


And this is what it looks like within Travis. There's a bunch of setup stuff happening to get the project to the point where it can build and then the tests are executed at line 46 onwards. Less output that you saw when from the previous screenshot, but that is a difference in configuration more than anything else.

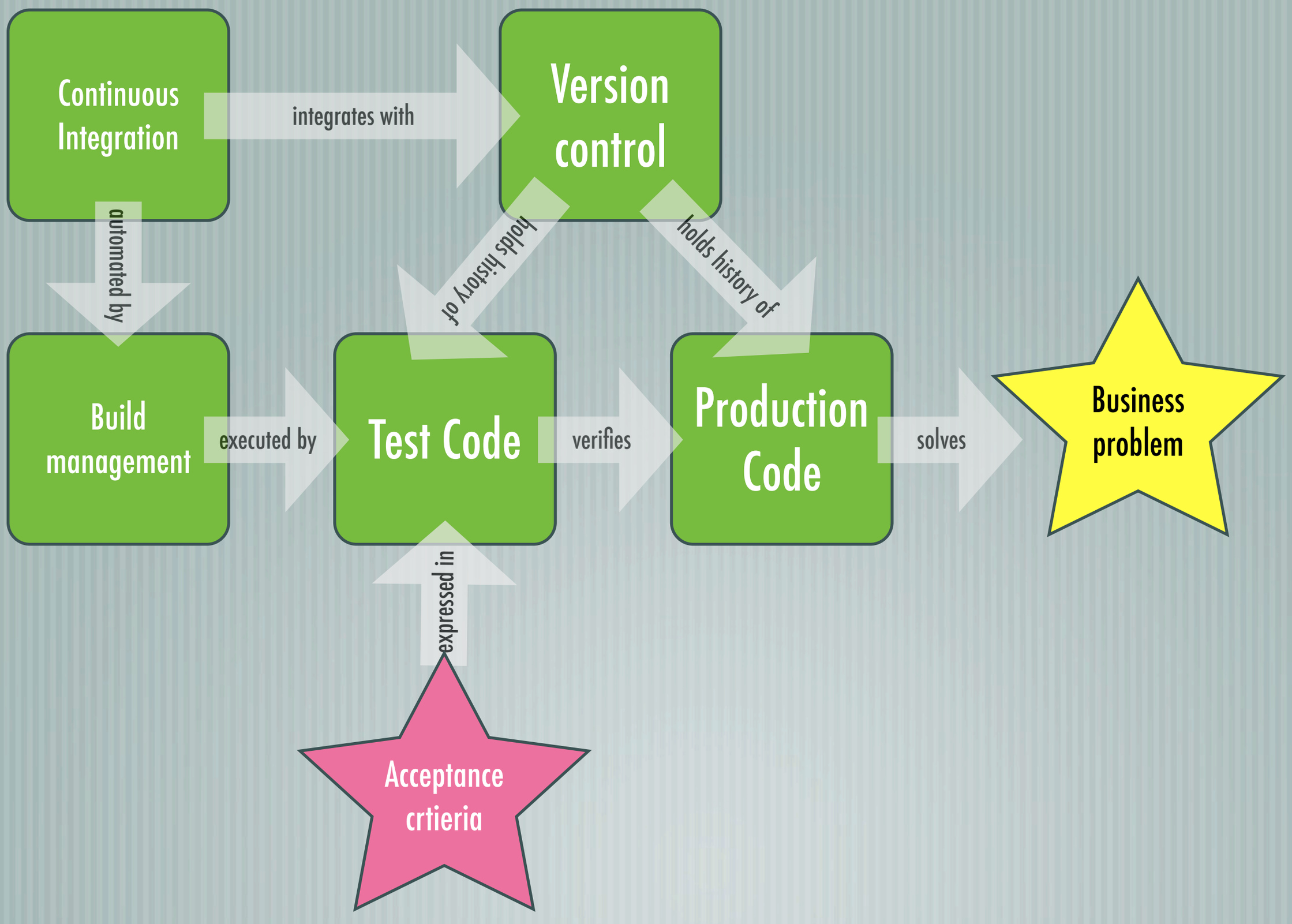
6. Make it work!

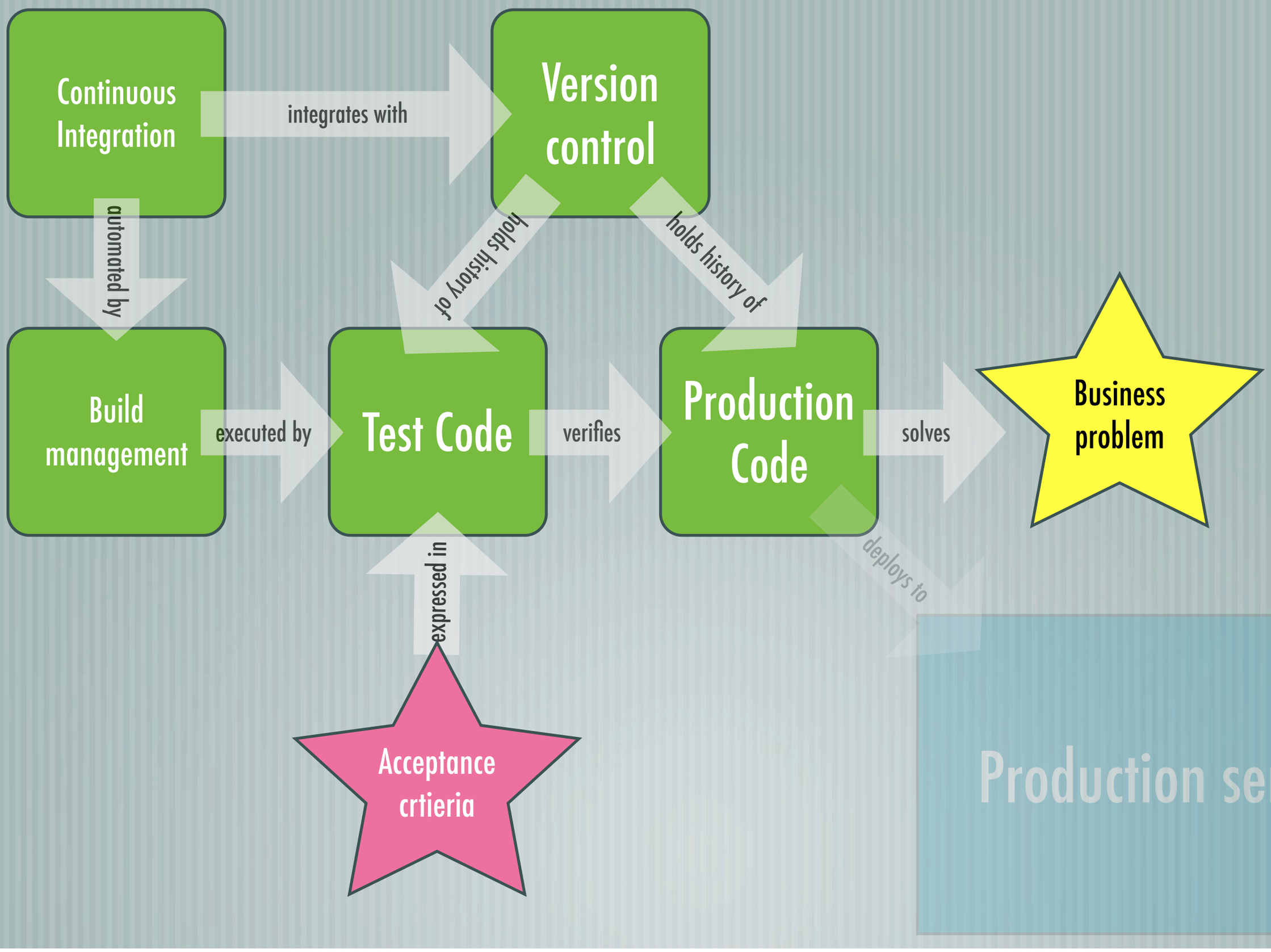
- Update tests in GitHub
- Update code in GitHub
- Test through Travis CI
- Repeat until 100% green

Now we have the minor little activity of making our tests pass. Take you time on this. Be patient if you haven't used Ruby or RSpec before. I'll float around and help where I'm needed.



If you need a bit of help working out how the ruby works in small pieces, head over to TryRuby.org, which gives you an online console that you can type little bits of Ruby into and see what happens. If you couple this tool with the cheat sheet around Ruby dates and times, you should be able to come across a working solution fairly soon.


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
Acceptance crtieria


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
deploys to
Acceptance crtieria
Production server

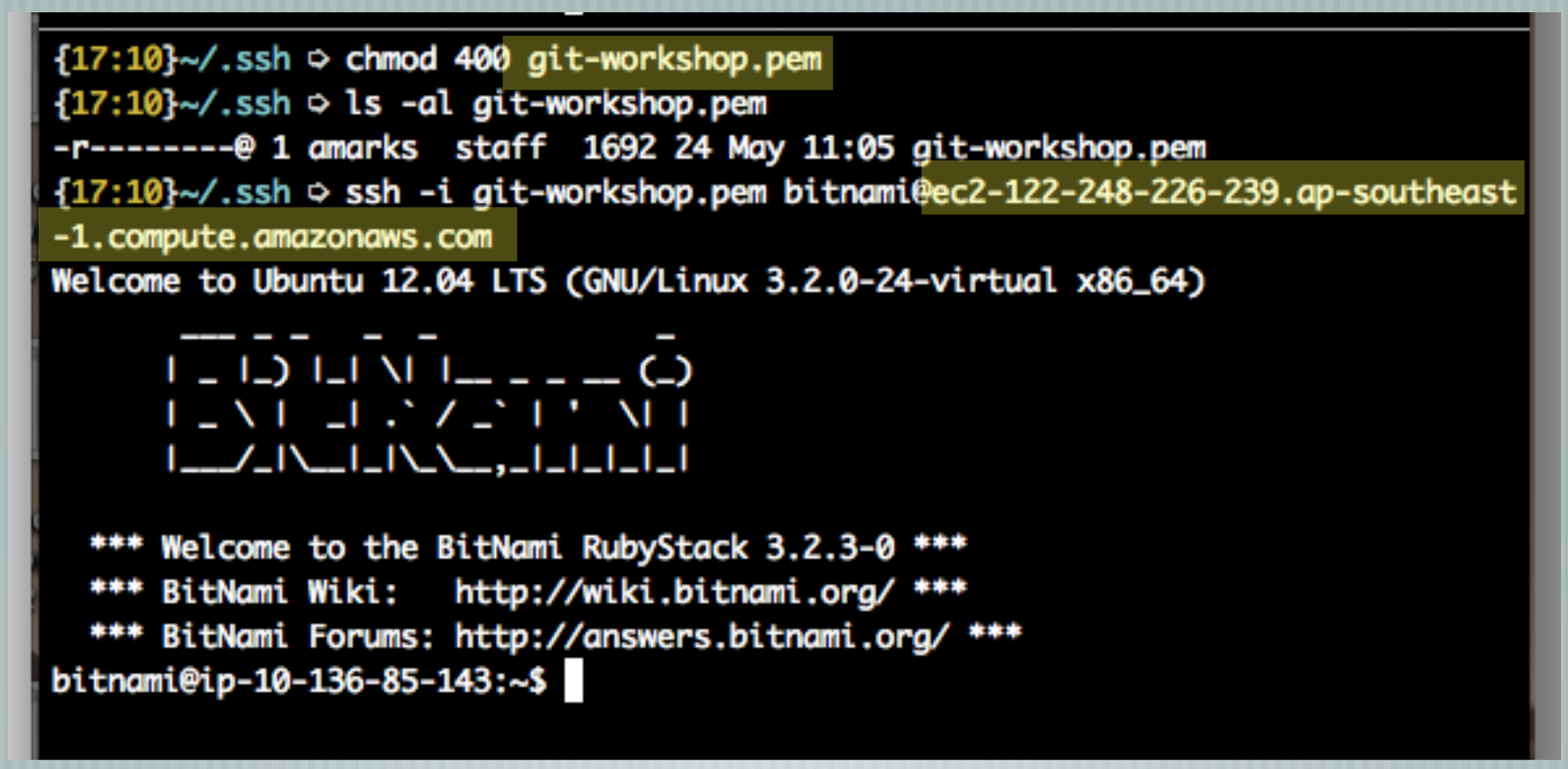
7. Deploy!

Checklist

- ☑ Private SSH key (git-workshop.pem)
- ☑ Public DNS of your EC2 instance

We're onto the last step of our workshops – which is putting our solution somewhere public for people to admire. There are a large number of ways we could have gone about doing this – I decided to use Amazon Web Services to provide virtual servers to run our clock applications, wrapped up in a very simple web application.

But there are a few prerequisites we need before we can start on this last step. I have created a separate virtual server within the AWS infrastructure for each pair here today, but each of you will need to be assigned to one of those servers to avoid clashes. The way we identify these virtual servers is by their public DNS names – each pair will allocate themselves to one of these servers. And to access these servers, each pair will need a copy of a private key file which will allow them to authenticate against the server during login.



Here you can see my terminal window and a couple of commands being run before logging into a virtual server running with AWS EC2.

What are the key points to note here? Well:

- you can see that I have a copy of the private key file: git-workshop.pem
- the weird “chmod 400” thing (if you’re not familiar with Unix file permissions) is changing access control on the file so that only the owner of the file can read it and no-one else has any permissions at all
- the main command is the “ssh” one which is what logs us into the instance.

But now that we’ve logged into our very own personal virtual server, what do we do?

```
bitnami@ip-10-136-85-143:~$ git clone git://github.com/andeemarks/git-workshop.git
Cloning into 'git-workshop'...
remote: Counting objects: 83, done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 83 (delta 34), reused 65 (delta 16)
Receiving objects: 100% (83/83), 13.43 KiB, done.
Resolving deltas: 100% (34/34), done.
bitnami@ip-10-136-85-143:~$ cd git-workshop/
bitnami@ip-10-136-85-143:~/git-workshop$ bundle install --deployment --without development test
Fetching source index for http://rubygems.org/
Installing rack (1.4.1)
Installing rack-protection (1.2.0)
Installing tilt (1.3.3)
Installing sinatra (1.3.2)
Using bundler (1.0.21)
Your bundle is complete! It was installed into ./vendor/bundle
bitnami@ip-10-136-85-143:~/git-workshop$ ruby -rubygems app/web_relative_clock.rb
== Sinatra/1.3.2 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.2.11 codename Bat-Shit Crazy)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:4567, CTRL+C to stop
```


The first thing we need to do is get a copy of the code from our GitHub repository. To do that we'll use the "git clone" command which is effectively doing the same as when we clicked the "fork" button to originally access this code.

In this case however, the code is being copied from the GitHub servers down onto the file system in our AWS server.

```
bitnami@ip-10-136-85-143:~$ git clone git://github.com/andeemarks/git-workshop.git
Cloning into 'git-workshop'...
remote: Counting objects: 83, done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 83 (delta 34), reused 65 (delta 16)
Receiving objects: 100% (83/83), 13.43 KiB, done.
Resolving deltas: 100% (34/34), done.
bitnami@ip-10-136-85-143:~$ cd git-workshop/
bitnami@ip-10-136-85-143:~/git-workshop$ bundle install --deployment --without development test
Fetching source index for http://rubygems.org/
Installing rack (1.4.1)
Installing rack-protection (1.2.0)
Installing tilt (1.3.3)
Installing sinatra (1.3.2)
Using bundler (1.0.21)
Your bundle is complete! It was installed into ./vendor/bundle
bitnami@ip-10-136-85-143:~/git-workshop$ ruby -rubygems app/web_relative_clock.rb
== Sinatra/1.3.2 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.2.11 codename Bat-Shit Crazy)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:4567, CTRL+C to stop
```


Doing the git clone created a “git-workshop” directory so we now need to change into that directory to get the application ready to run.

```
bitnami@ip-10-136-85-143:~$ git clone git://github.com/andeemarks/git-workshop.git
Cloning into 'git-workshop'...
remote: Counting objects: 83, done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 83 (delta 34), reused 65 (delta 16)
Receiving objects: 100% (83/83), 13.43 KiB, done.
Resolving deltas: 100% (34/34), done.
bitnami@ip-10-136-85-143:~$ cd git-workshop/
bitnami@ip-10-136-85-143:~/git-workshop$ bundle install --deployment --without development test
Fetching source index for http://rubygems.org/
Installing rack (1.4.1)
Installing rack-protection (1.2.0)
Installing tilt (1.3.3)
Installing sinatra (1.3.2)
Using bundler (1.0.21)
Your bundle is complete! It was installed into ./vendor/bundle
bitnami@ip-10-136-85-143:~/git-workshop$ ruby -rubygems app/web_relative_clock.rb
== Sinatra/1.3.2 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.2.11 codename Bat-Shit Crazy)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:4567, CTRL+C to stop
```


One of the other things that Travis CI was doing that we glossed over last time was also retrieving a bunch of extra bits of Ruby code that we need to run our application.

The retrieval is being done via another Ruby program called Bundler, which looks inside a file called Gemfile to find the names of these extra bits of code and then downloads them so our application can use them.

Let’s look and see what’s in our Gemfile.

git-workshop / Gemfile

andeemarks 10 hours ago Added grouping to Gemfile for quicker deployment

1 contributor

100644 | 12 lines (8 sloc) | 0.136 kb

```
1  source "http://rubygems.org"
2
3  group :test do
4          gem "rspec", :require => "spec"
5  end
6
7  group :development do
8          gem "rake"
9  end
10
11 gem "sinatra"
```


There are only really two elements worth looking at in this file; gems and groups of gems.

Gems are the name given to those extra pieces of Ruby code I mentioned before. Each gem will have a name and you can see we need three gems for this application; rspec (for testing), rake (for building) and sinatra (for running, although we haven't seen what that means yet).

Gems can also be grouped according to how they're used. In this case I've identified the gems I only need for development and testing - all the other ones are production gems.

```
bitnami@ip-10-136-85-143:~$ git clone git://github.com/andeemarks/git-workshop.git
Cloning into 'git-workshop'...
remote: Counting objects: 83, done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 83 (delta 34), reused 65 (delta 16)
Receiving objects: 100% (83/83), 13.43 KiB, done.
Resolving deltas: 100% (34/34), done.
bitnami@ip-10-136-85-143:~$ cd git-workshop/
bitnami@ip-10-136-85-143:~/git-workshop$ bundle install --deployment --without development test
Fetching source index for http://rubygems.org/
Installing rack (1.4.1)
Installing rack-protection (1.2.0)
Installing tilt (1.3.3)
Installing sinatra (1.3.2)
Using bundler (1.0.21)
Your bundle is complete! It was installed into ./vendor/bundle
bitnami@ip-10-136-85-143:~/git-workshop$ ruby -rubygems app/web_relative_clock.rb
== Sinatra/1.3.2 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.2.11 codename Bat-Shit Crazy)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:4567, CTRL+C to stop
```


Now we understand a bit more about Gemfiles, we can make more sense of this Bundler command.

What it's saying is "install all the gems in my Gemfile, make them ready for deployment, but only those that aren't marked as for development or testing".

In theory this should only leave the Sinatra gem, but it has it's own dependencies that it insists on downloading, which is why tilt and the other ones are there as well.

```
bitnami@ip-10-136-85-143:~$ git clone git://github.com/andeemarks/git-workshop.git
Cloning into 'git-workshop'...
remote: Counting objects: 83, done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 83 (delta 34), reused 65 (delta 16)
Receiving objects: 100% (83/83), 13.43 KiB, done.
Resolving deltas: 100% (34/34), done.
bitnami@ip-10-136-85-143:~$ cd git-workshop/
bitnami@ip-10-136-85-143:~/git-workshop$ bundle install --deployment --without development test
Fetching source index for http://rubygems.org/
Installing rack (1.4.1)
Installing rack-protection (1.2.0)
Installing tilt (1.3.3)
Installing sinatra (1.3.2)
Using bundler (1.0.21)
Your bundle is complete! It was installed into ./vendor/bundle
bitnami@ip-10-136-85-143:~/git-workshop$ ruby -rubygems app/web_relative_clock.rb
== Sinatra/1.3.2 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.2.11 codename Bat-Shit Crazy)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:4567, CTRL+C to stop
```


Next step is to run the application. And now we need to know a bit about Sinatra.

Sinatra is a Ruby framework for creating very simple web applications, which is good because we'll be creating exactly this to run our clock code.

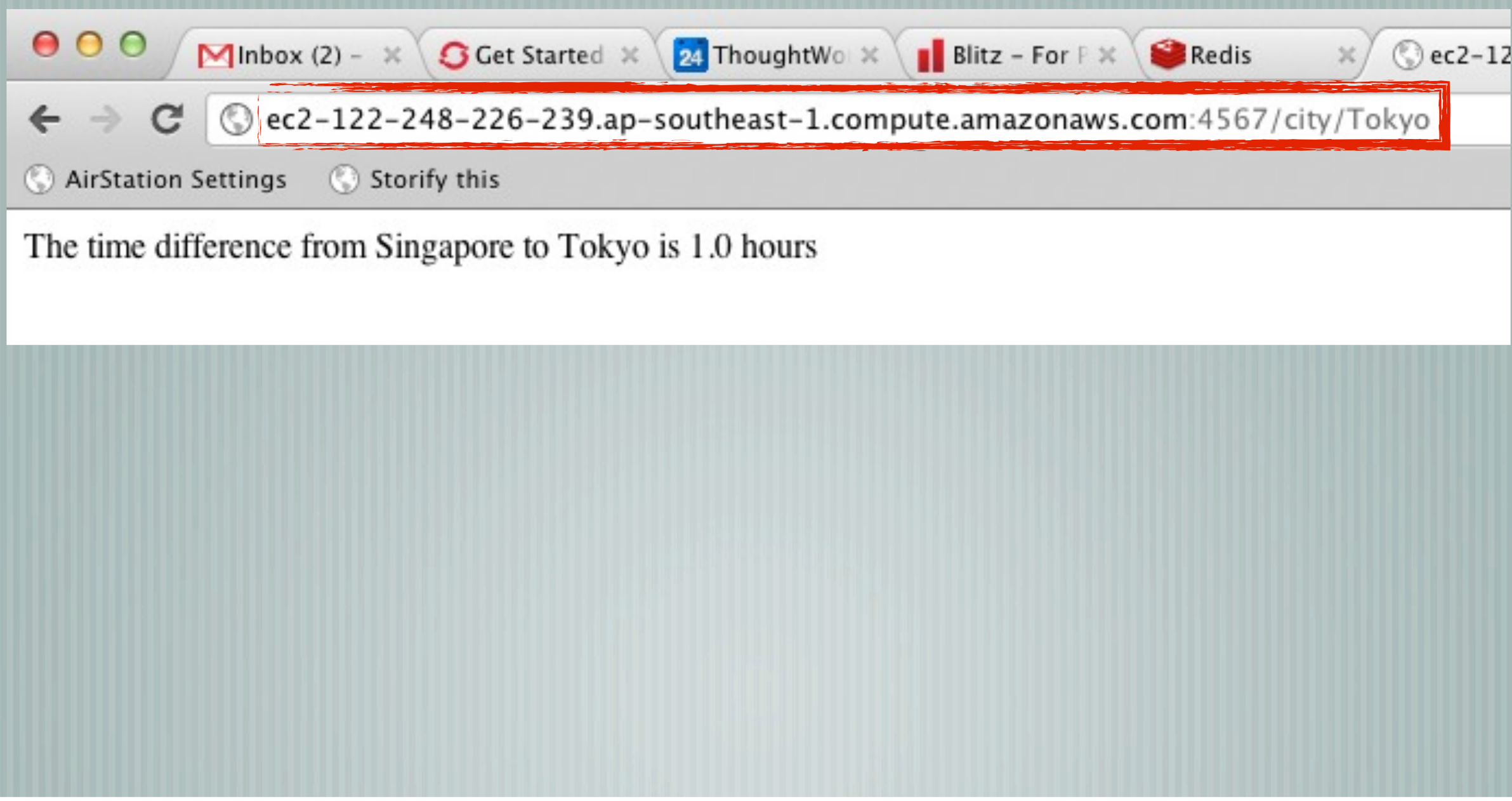
A Sinatra application is a ruby application that is capable of handling requests sent to it from the browser.

This command starts one of the files in our repository which contains the Sinatra part of the application. We'll have a look at what's in this file soon.

```
bitnami@ip-10-136-85-143:~$ git clone git://github.com/andeemarks/git-workshop.git
Cloning into 'git-workshop'...
remote: Counting objects: 83, done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 83 (delta 34), reused 65 (delta 16)
Receiving objects: 100% (83/83), 13.43 KiB, done.
Resolving deltas: 100% (34/34), done.
bitnami@ip-10-136-85-143:~$ cd git-workshop/
bitnami@ip-10-136-85-143:~/git-workshop$ bundle install --deployment --without development test
Fetching source index for http://rubygems.org/
Installing rack (1.4.1)
Installing rack-protection (1.2.0)
Installing tilt (1.3.3)
Installing sinatra (1.3.2)
Using bundler (1.0.21)
Your bundle is complete! It was installed into ./vendor/bundle
bitnami@ip-10-136-85-143:~/git-workshop$ ruby -rubygems app/web_relative_clock.rb
== Sinatra/1.3.2 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.2.11 codename Bat-Shit Crazy)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:4567, CTRL+C to stop
```

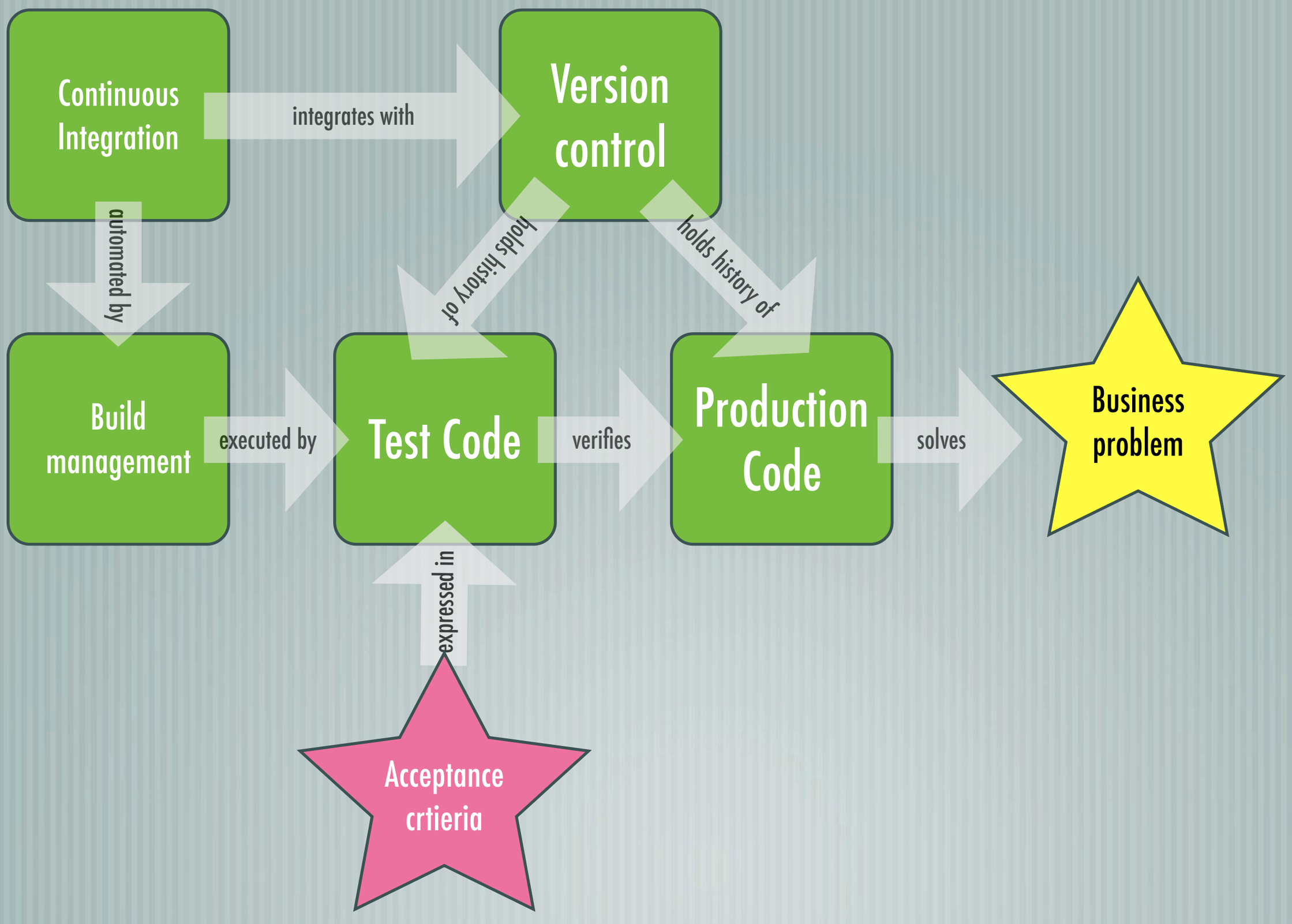

The other interesting thing to note is that by default this application is looking for request on port 4567, instead of the standard port 80 that we're used to for most web servers.

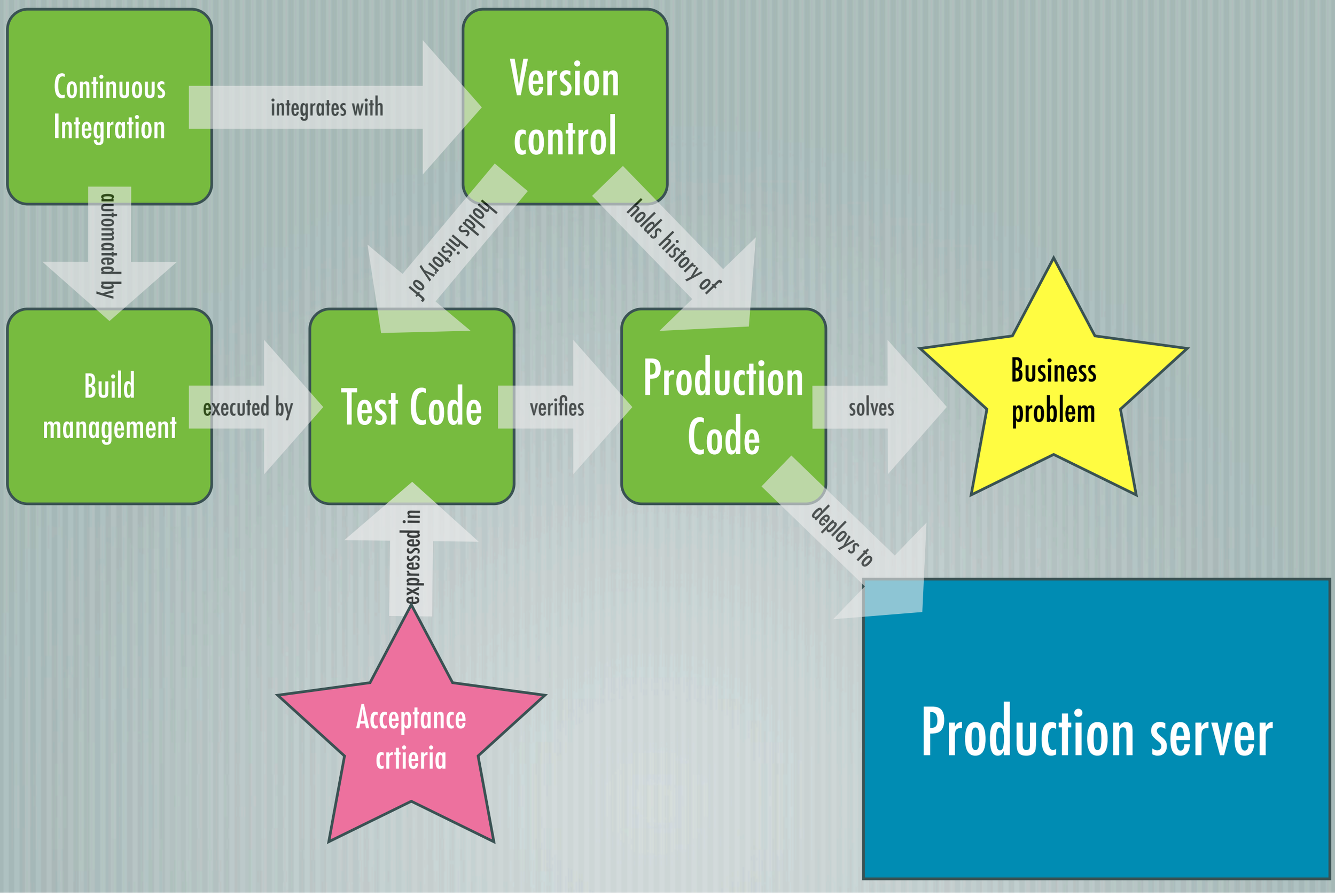
So the only other thing we need to look at our application in action is the Public DNS of our AWS virtual server again.



If you can put those two pieces of data together, you'll pretty much have a URL (albeit an ugly one) that you can use to access our application.

Note that we've added a bit of extra stuff on the end of the URL to be able to specify a city


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
Acceptance crtieria


Continuous Integration
integrates with
Version control
automated by
holds history of
holds history of
Build management
executed by
Test Code
verifies
Production Code
solves
Business problem
expressed in
deploys to
Acceptance crtieria
Production server

git-workshop / app / **web_relative_clock.rb**

andeemarks 8 hours ago Caught unknown city error and wrapped in nicer message

1 contributor

100644 | 22 lines (16 sloc) | 0.444 kb

Edit this file | Raw | Bla

```
1  require 'sinatra'
2
3  require_relative '../lib/singapore_relative_clock.rb'
4
5  get '/' do
6          "Try using me like... #{request.url}city/Singapore"
7  end
8
9  get '/city/:city' do
10         @clock = SingaporeRelativeClock.new
11
12         city = params[:city]
13         begin
14                 "The time difference from Singapore to #{city} is #{@clock.timeDifferenceTo(city)} hours"
15         rescue RuntimeError
16                 "Sorry - could not find a city called #{city}.  Try again."
17         end
18 end
19
```


And here's all the code we need to serve up that plain old vanilla web page.

Talk about:

- routes
- parameters
- string interpolation
- exception handling
- require statements

Too much typing!

Hang on a second - there's got to be an easier way to get our application ready for production though hasn't there? We typed a lot of stuff to get all those commands onto the screen.

And as you may have guessed - there is.

git-workshop / **start-server.sh**

andeemarks 6 hours ago Added startup script for Bitnami RubyStack AMI ami-74094f

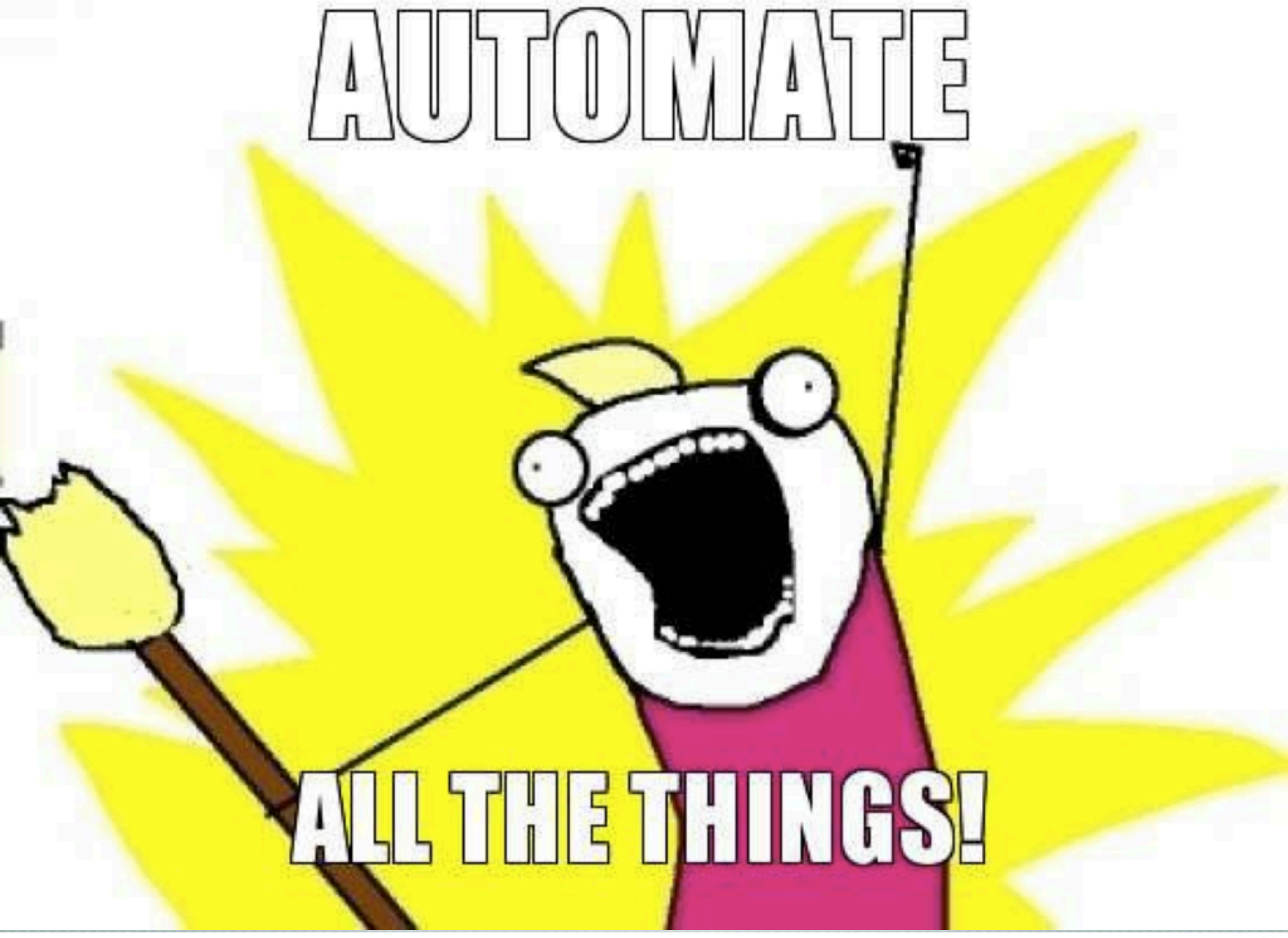
1 contributor

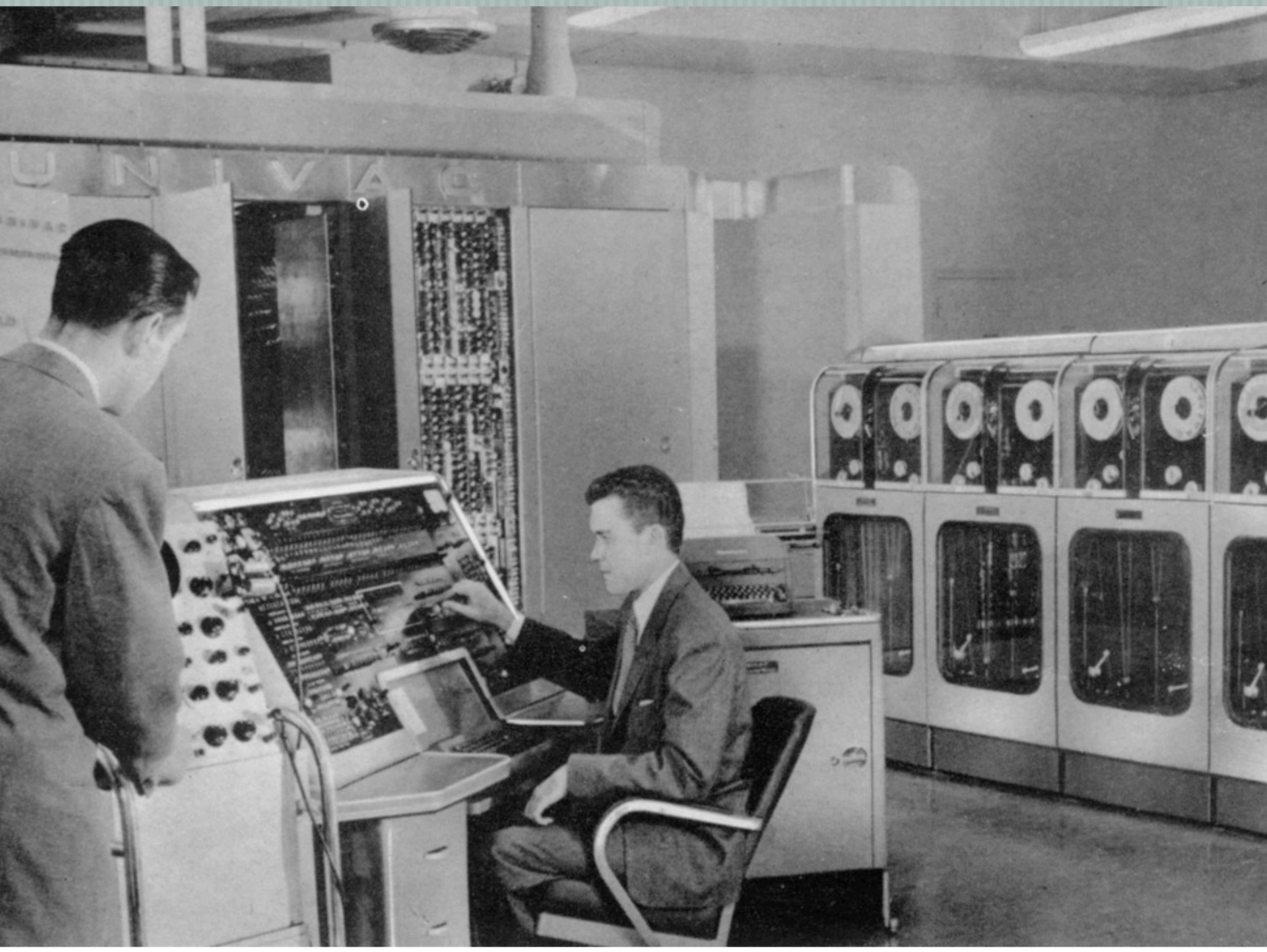
100644 | 11 lines (11 sloc) | 0.393 kb

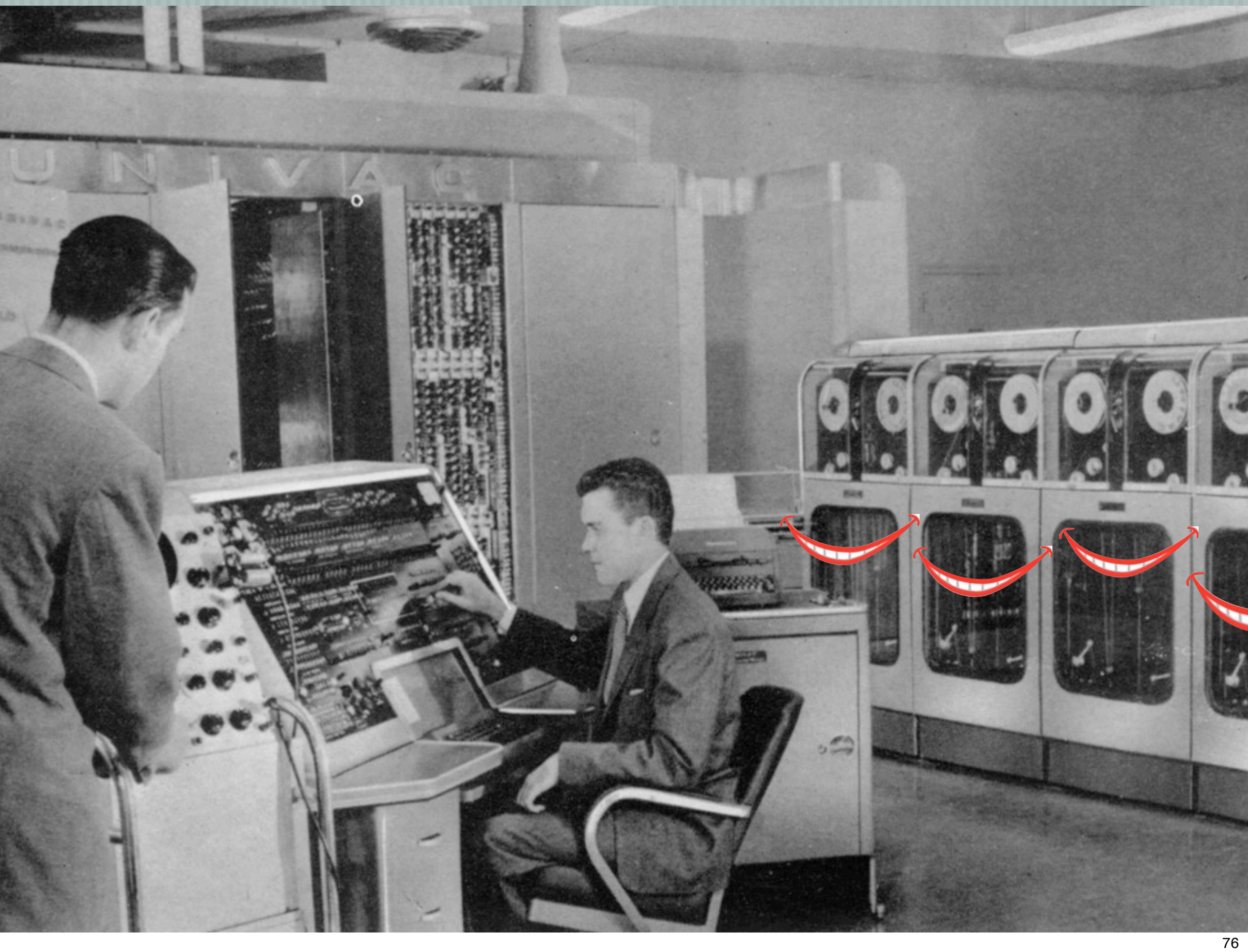
```
1  #!/bin/bash
2  cd ~
3  rm -rf git-workshop
4  kill `lsof | grep 4567 | awk '{print $2}'`
5  git clone git://github.com/andeemarks/git-workshop.git
6  cd git-workshop
7  bundle install --deployment --without development test
8  ruby -rubygems app/web_relative_clock.rb &
9  HOST_NAME=`ec2metadata | grep public-hostname | awk '{print $2}'`
10 sleep 2
11 echo "**** Now running - check http://$HOST_NAME:4567/city/<cityname>"
```

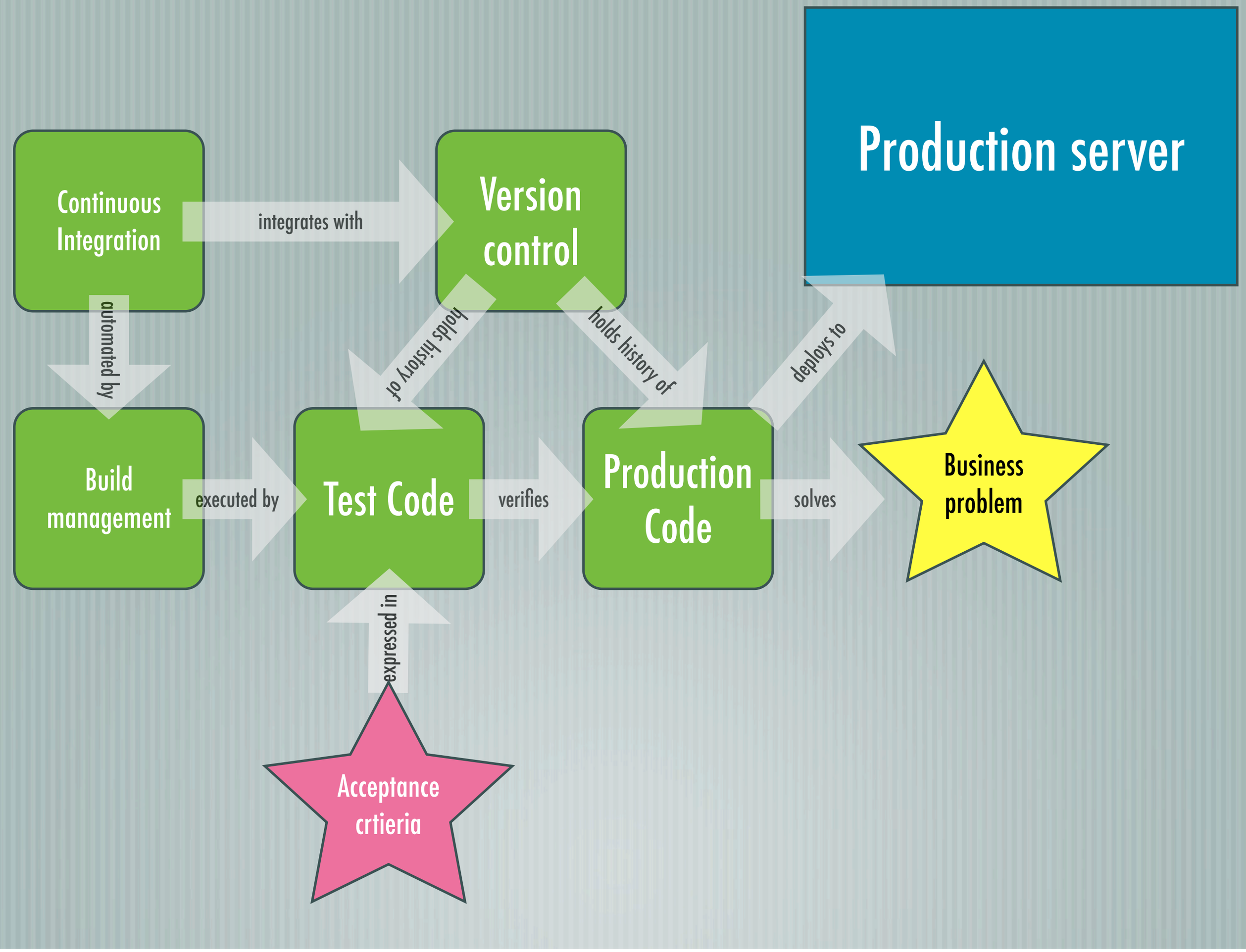

So what does this script do when it's run?

Well pretty much all the stuff we saw before. In addition to all those familiar commands we looked at a couple of slides back, there's a bit of extra stuff (lines 3, 4, 9, 10, 11) to make the whole operation repeatable and a little more communicative.


AUTOMATE
ALL THE THINGS!

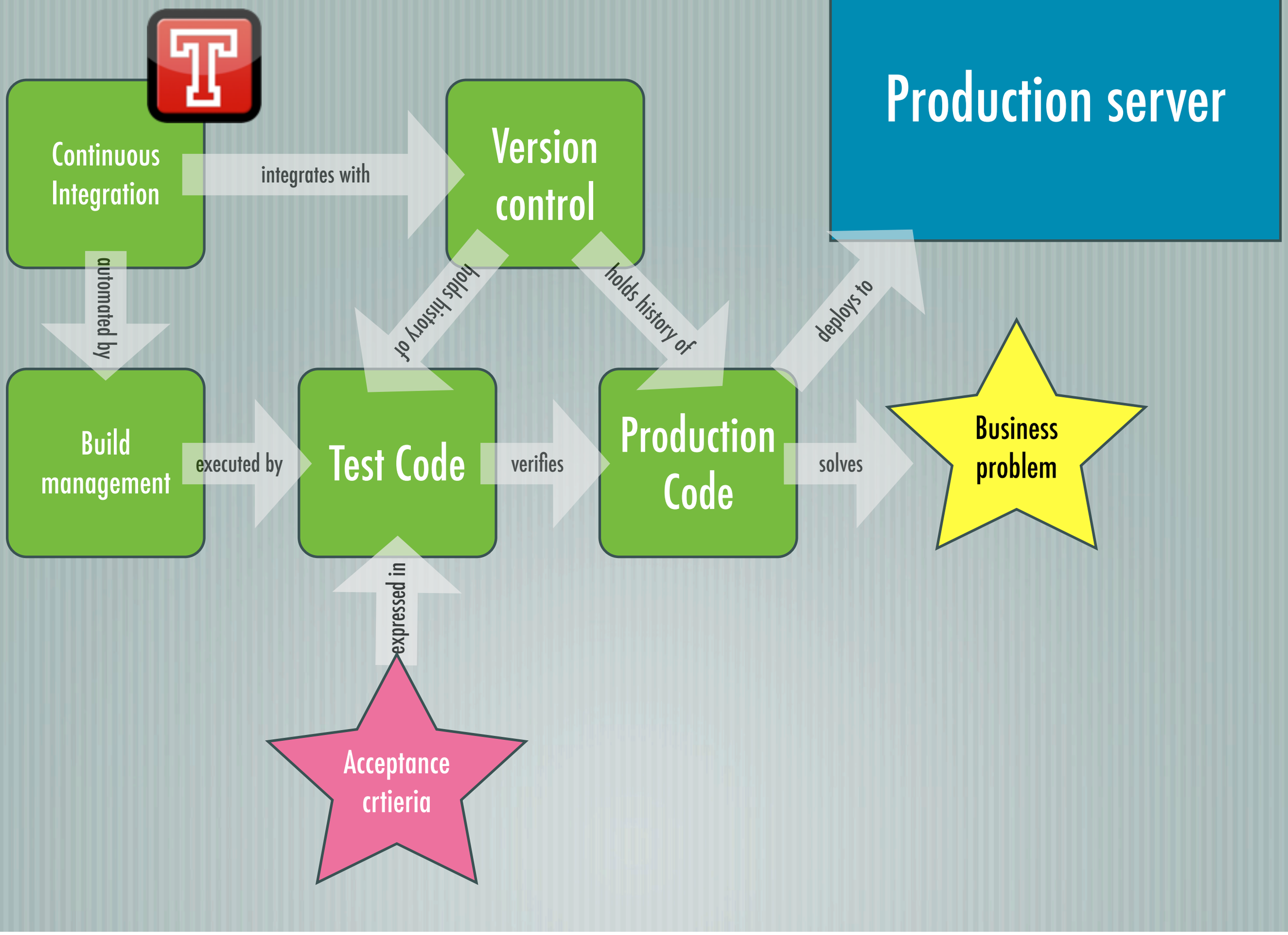



UNIVAC



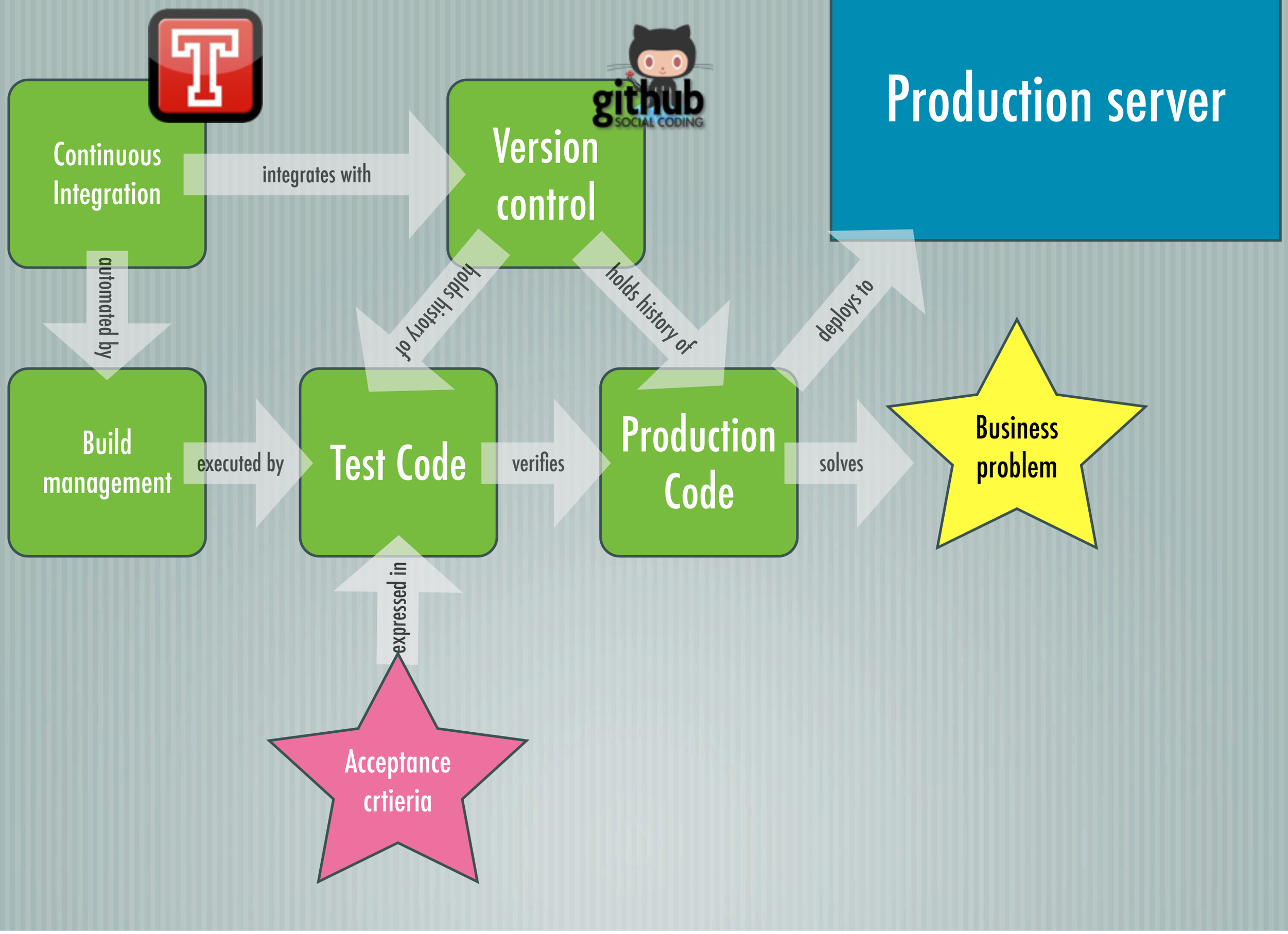
So let's revisit where we started with these workshops and where we've finished.

As I mentioned at the start, we've covered a lot of territory in a short space of time, so we've definitely emphasised breadth over depth. How much breadth? Well lets re-examine the main technical topics I mentioned earlier.



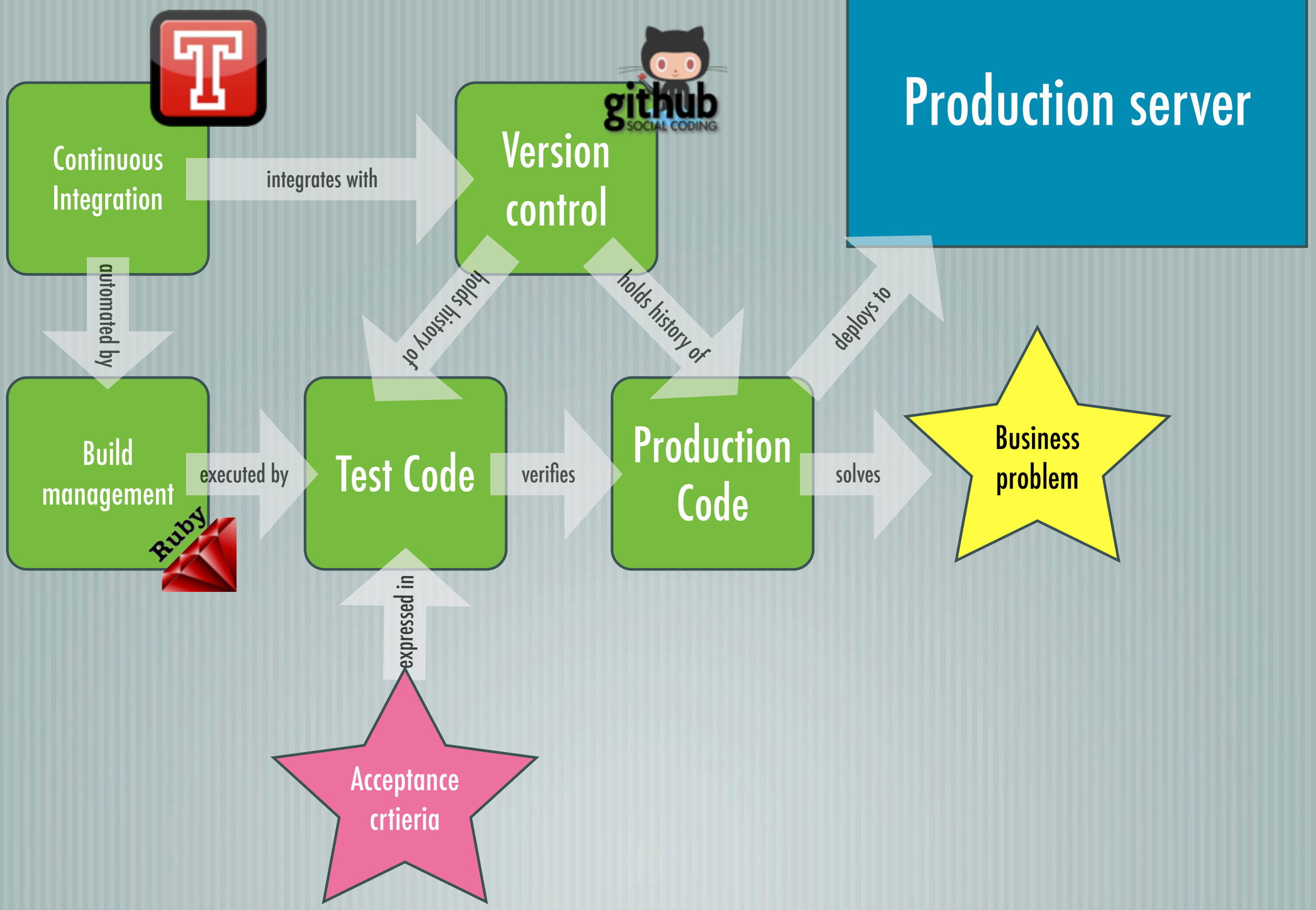
So let's revisit where we started with these workshops and where we've finished.

As I mentioned at the start, we've covered a lot of territory in a short space of time, so we've definitely emphasised breadth over depth. How much breadth? Well lets re-examine the main technical topics I mentioned earlier.



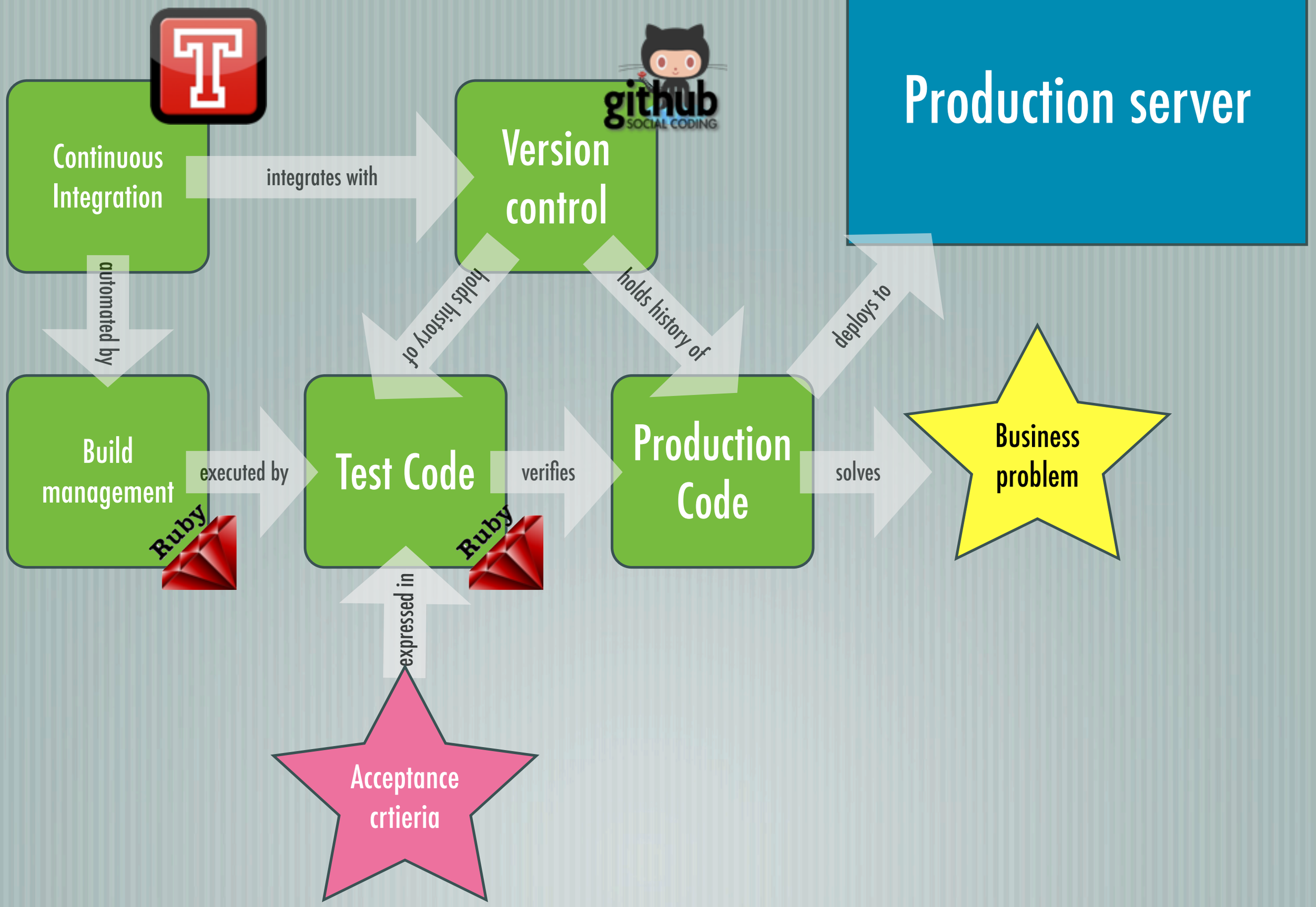
So let's revisit where we started with these workshops and where we've finished.

As I mentioned at the start, we've covered a lot of territory in a short space of time, so we've definitely emphasised breadth over depth. How much breadth? Well lets re-examine the main technical topics I mentioned earlier.



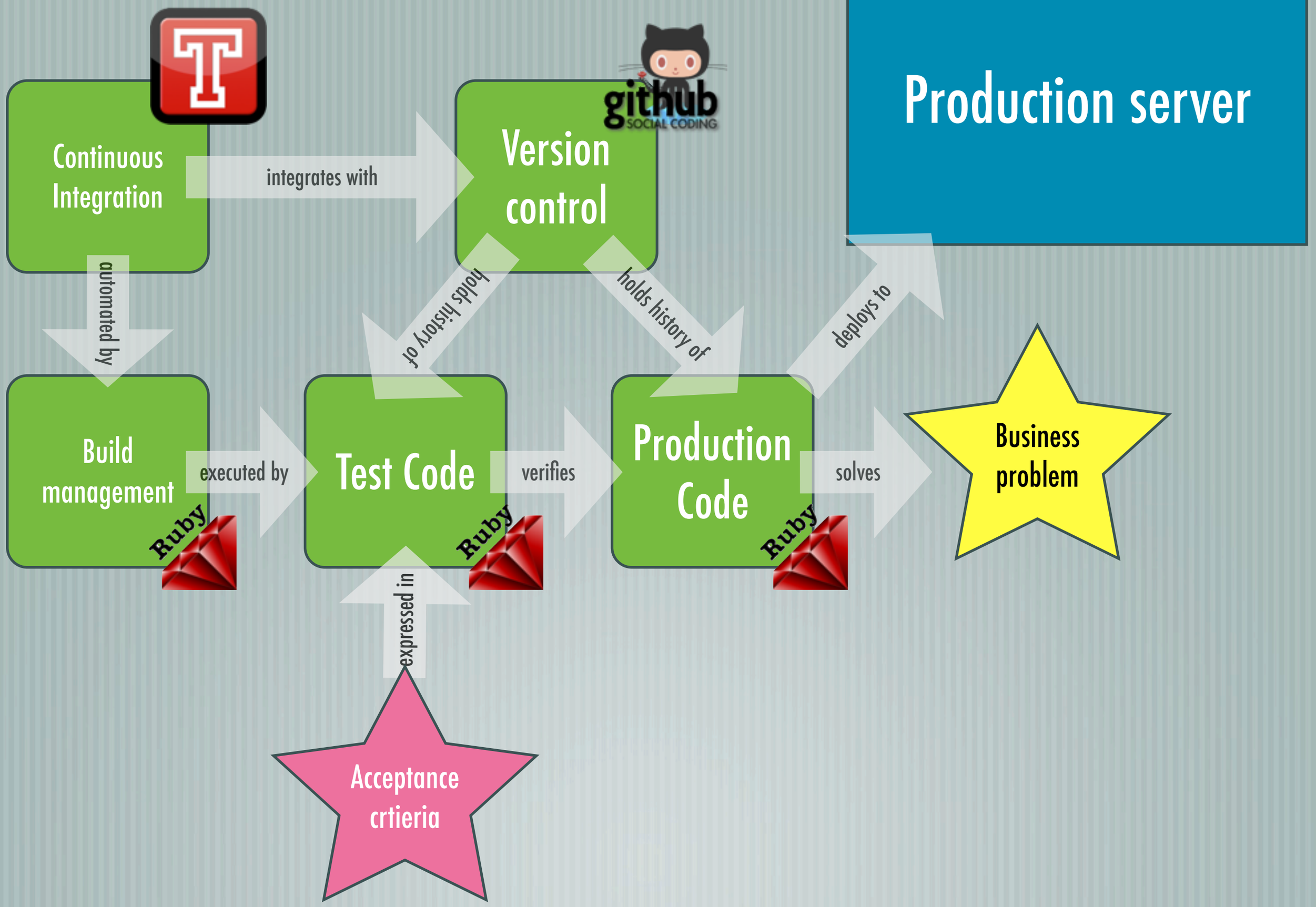
So let's revisit where we started with these workshops and where we've finished.

As I mentioned at the start, we've covered a lot of territory in a short space of time, so we've definitely emphasised breadth over depth. How much breadth? Well lets re-examine the main technical topics I mentioned earlier.



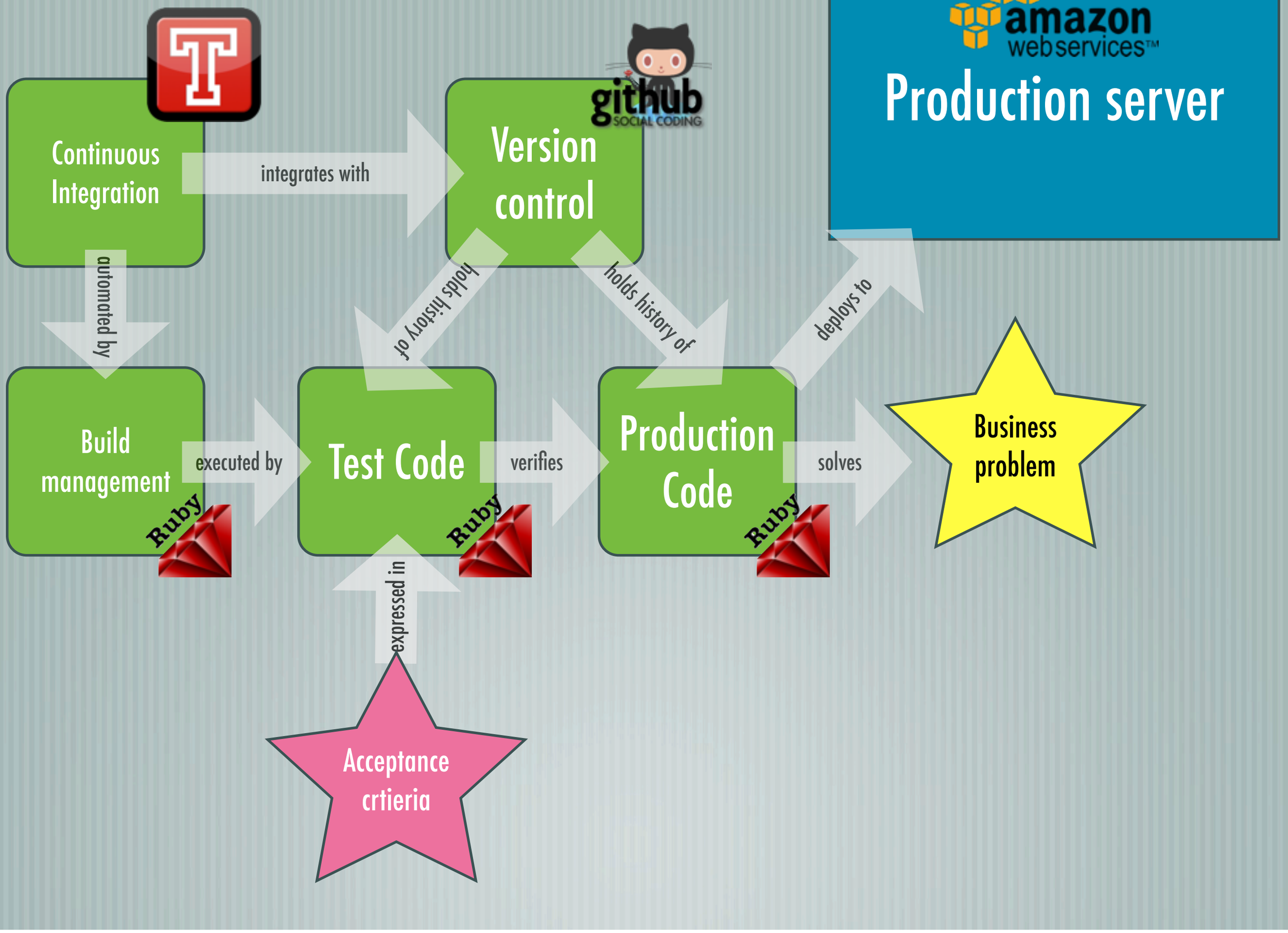
So let's revisit where we started with these workshops and where we've finished.

As I mentioned at the start, we've covered a lot of territory in a short space of time, so we've definitely emphasised breadth over depth. How much breadth? Well lets re-examine the main technical topics I mentioned earlier.



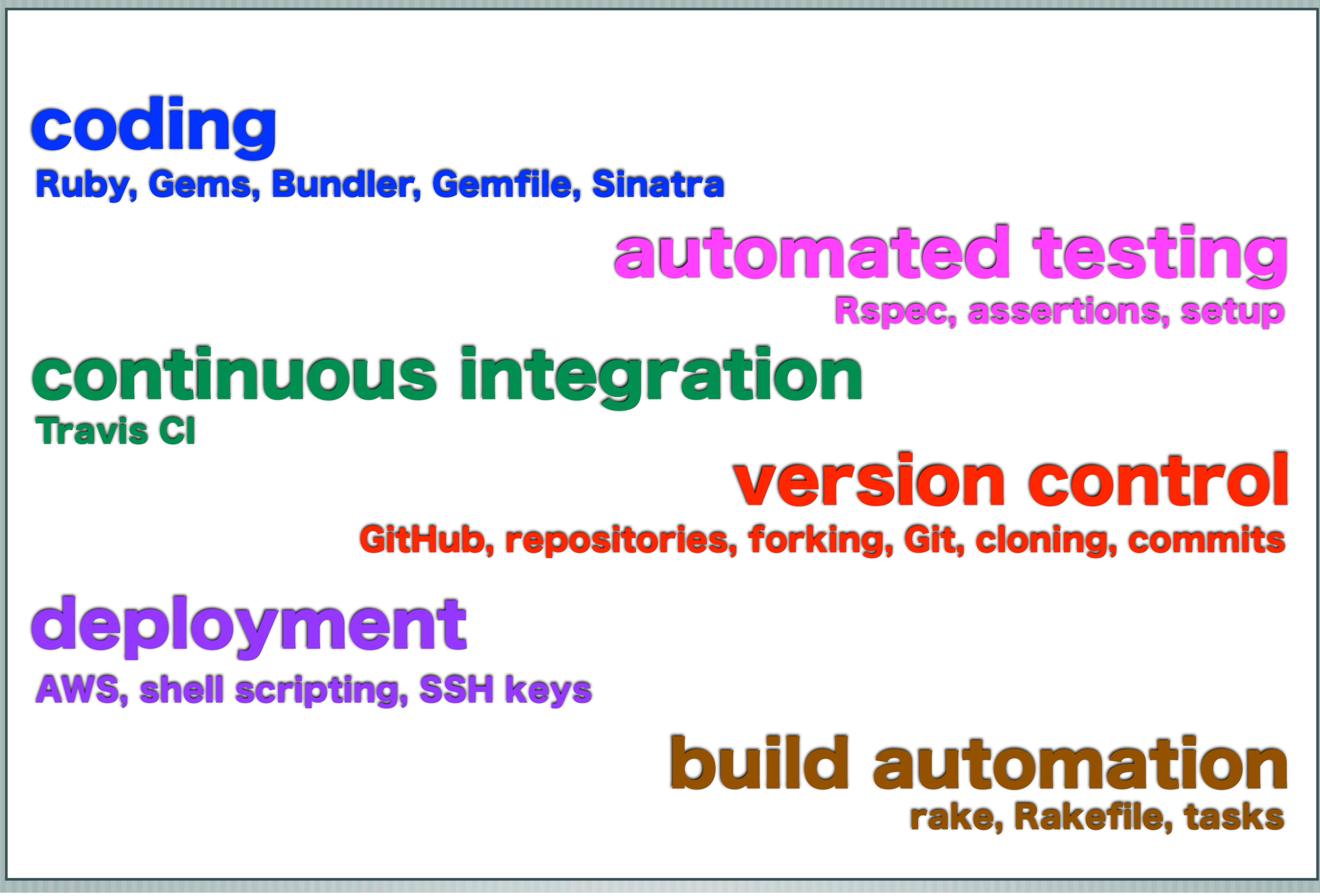
So let's revisit where we started with these workshops and where we've finished.

As I mentioned at the start, we've covered a lot of territory in a short space of time, so we've definitely emphasised breadth over depth. How much breadth? Well lets re-examine the main technical topics I mentioned earlier.



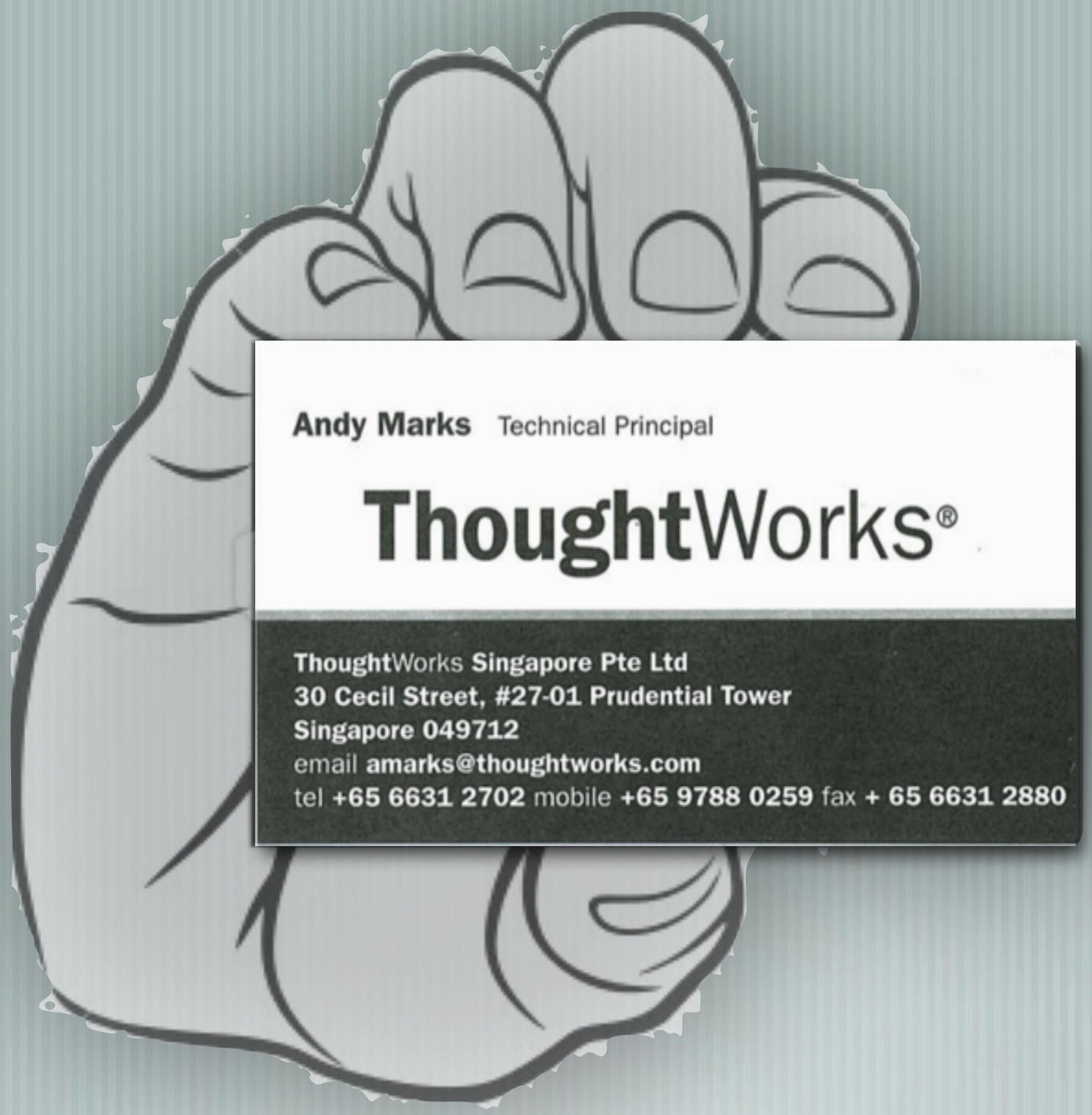
So let's revisit where we started with these workshops and where we've finished.

As I mentioned at the start, we've covered a lot of territory in a short space of time, so we've definitely emphasised breadth over depth. How much breadth? Well lets re-examine the main technical topics I mentioned earlier.



So let's revisit where we started with these workshops and where we've finished.

As I mentioned at the start, we've covered a lot of territory in a short space of time, so we've definitely emphasised breadth over depth. How much breadth? Well lets re-examine the main technical topics I mentioned earlier.


Andy Marks Technical Principal
ThoughtWorks®
ThoughtWorks Singapore Pte Ltd
30 Cecil Street, #27-01 Prudential Tower
Singapore 049712
email amarks@thoughtworks.com
tel +65 6631 2702 mobile +65 9788 0259 fax + 65 6631 2880