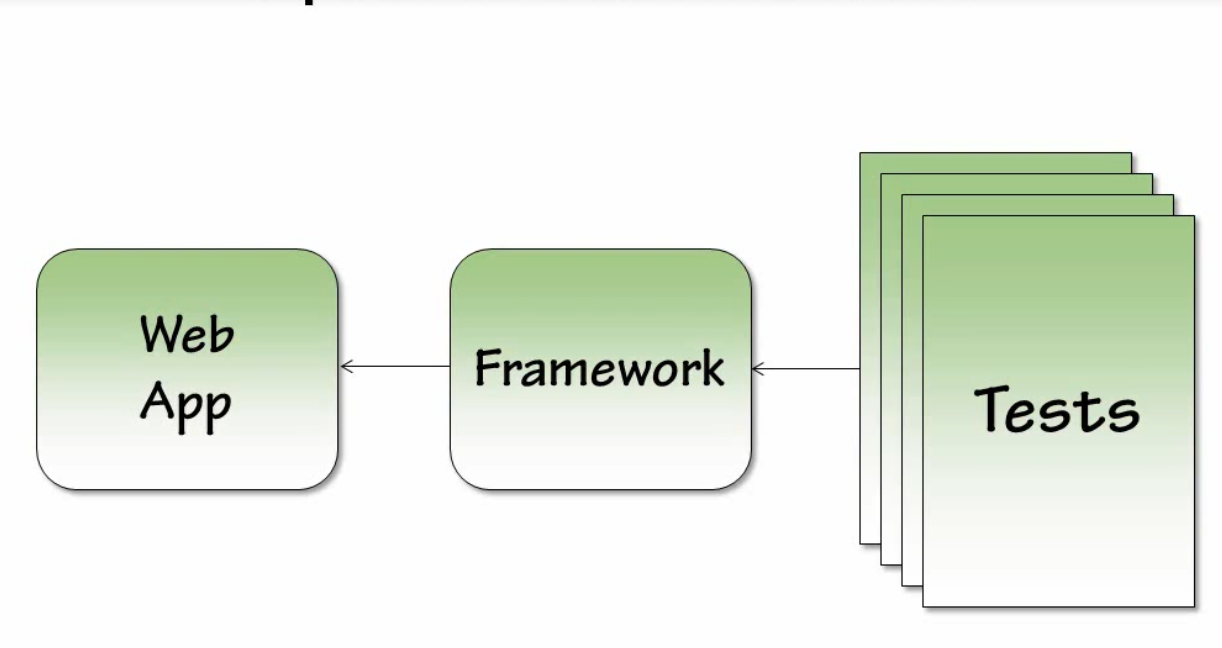
Automation Testing

3 types of testing

1. Unit Testing
2. Integration Testing
3. Black Box testing

In Black box testing, we built a framework that sits between the Web app and the tests. So anything changes in the UI of the Webapp will break the framework and not the tests. This means that fixing the framework will fix all the cases.

Also we do not need to know the actual code behind the Web app for testing this.

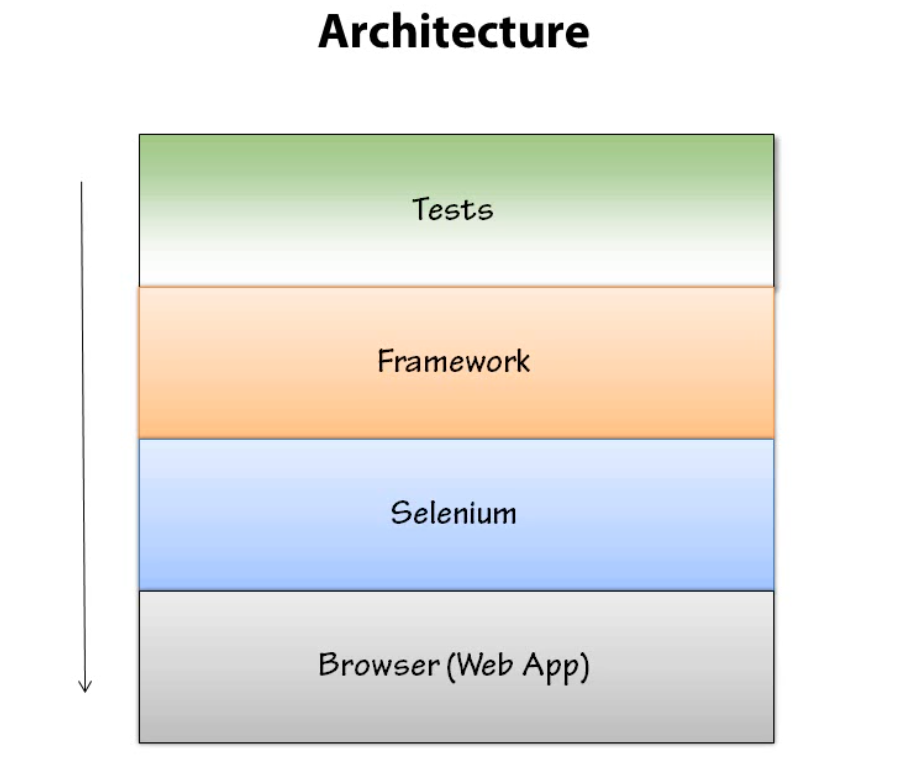


Write simple tests instead of complex tests. This means that tests aren’t going to do a lot and tests themselves express what they are doing in very simple terms. Most of the tasks are done by the framework.

The main key to design the framework is the tests themselves. We will write simple tests and then design the framework such that the tests work.

To create an automation framework, we need an application to test. For this purpose, we will create a framework to test the backend functionality of wordpress.org website. The framework greatly depends on the structure of the application under test. Framework normally mirrors the Website under test structure.

The basic architecture



Selenium is the tool that we are going to use for automation.

Tests are going to be utilizing framework and framework is going to be utilizing the Selenium or any other browser automation tool that you are using. And selenium is going to be the only thing that going to directly manipulating the browser.

Tests do not directly interact with the browser and also don’t interact directly with selenium.

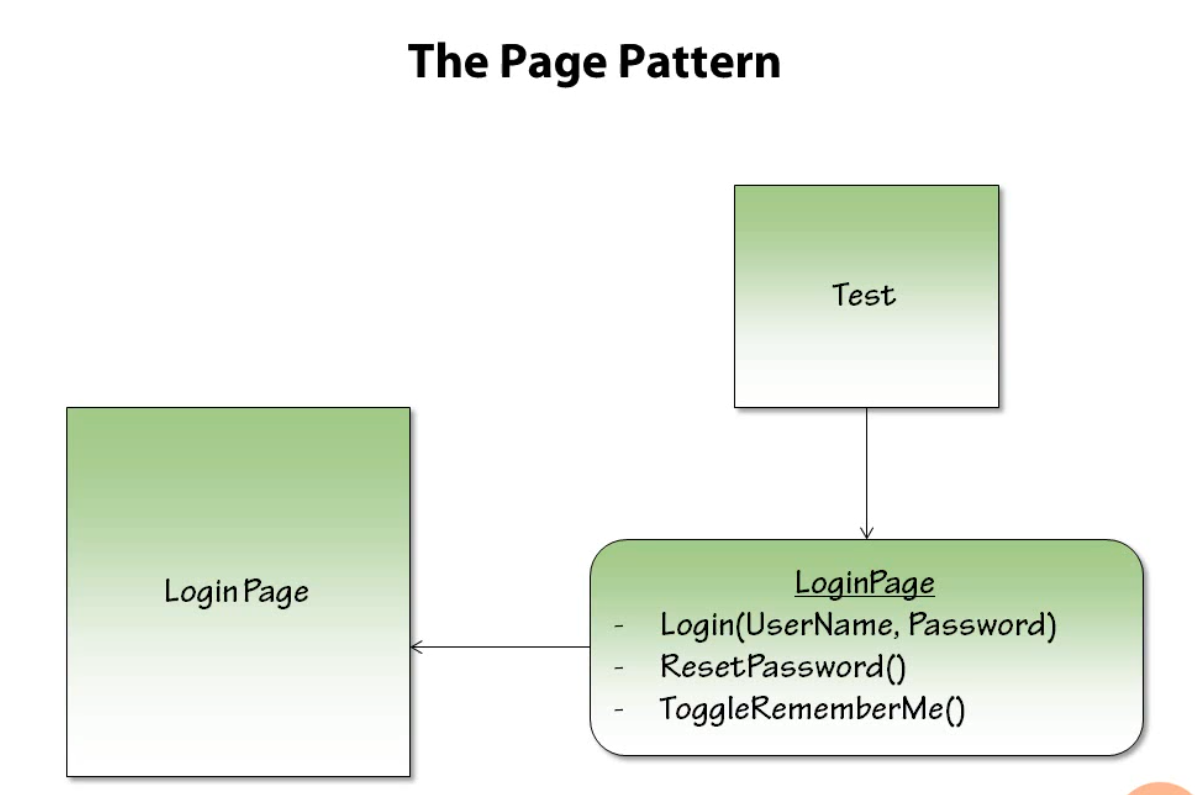
Now when a change comes to the web app, selenium is going to work the same as it work with specific browser, but the framework is going to change the implement to the correct one. By doing so, our tests will automatically start running and getting passed. So any change in the app breaks the framework and not the tests.

We will make a level of abstractions so that anything higher can only communicate with one level below.

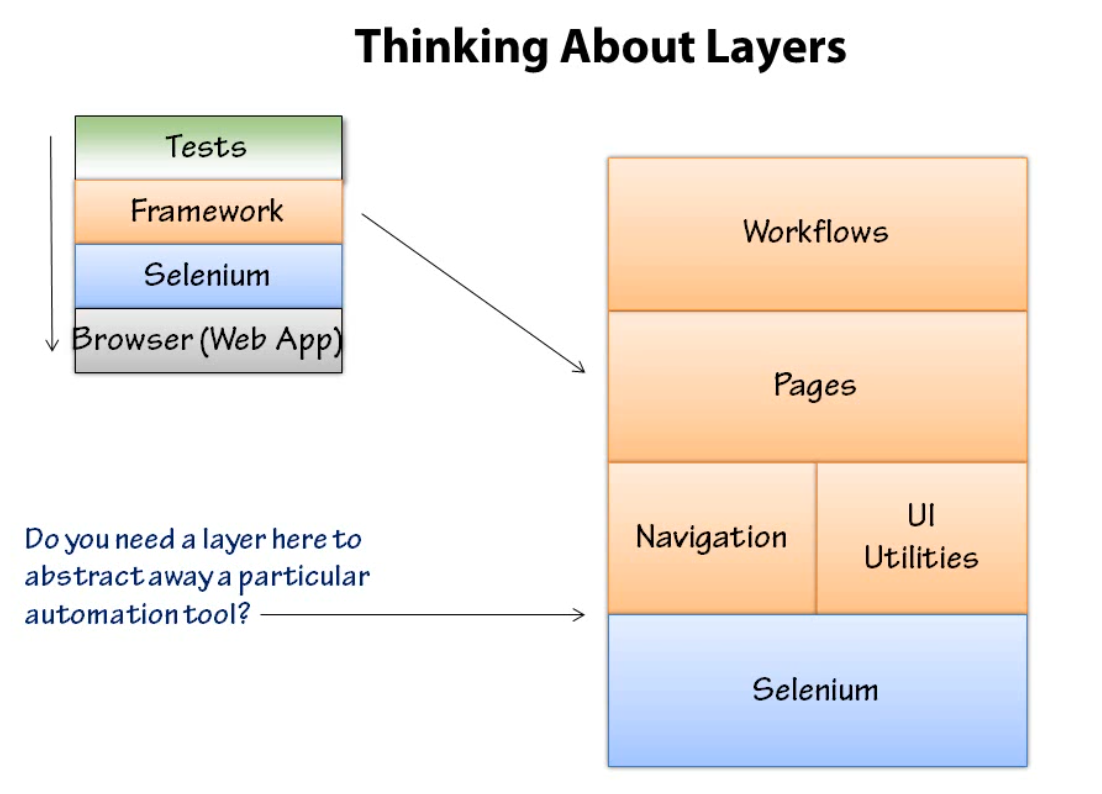
Tests talks to framework, framework talks to selenium and selenium only talks to browser.

In our architecture we are going to use the page pattern. This is most used in automation because it is very effective in creating the proper abstractions between the actual web page we are testing and the tests.

The basic idea of the page pattern that you need to have a class for each page that contains methods on it which represents all the functionality of the page.



Note that the methods on the page class should corresponds to the actual actions the user can do on the page and not to the UI elements on the page itself.



Workflows are things that are bigger than pages. A workflow can be like create a new customer and we might use that workflow to create a customer without having automation going to the new customer page and adding their details if we had some kind of functionality in our framework to do some common things like create a new customer, create a new product, add products to cart. Things you might do outside the scope of a single page, things that will span its functionality to multiple pages in the application.

The page layer -> Workflows are going to depend on individual pages. The pages are going to define the functionality for a single page.

Navigation -> it is the common navigation for your web application. Like header, footer and sidebar if they are present for all the pages.

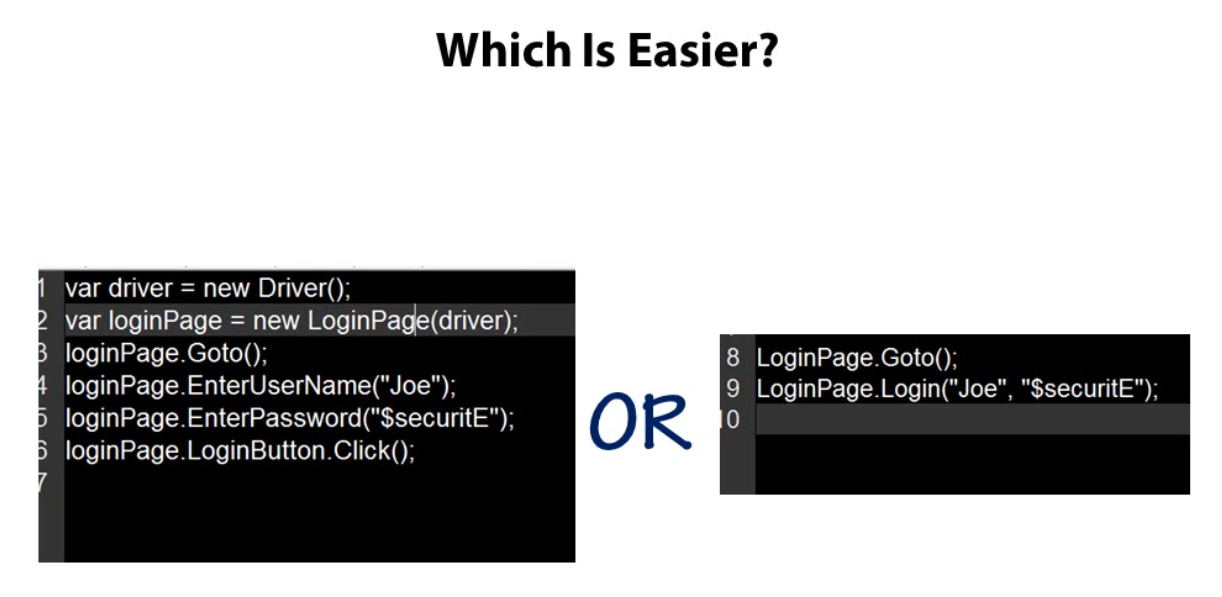
UI Utilities -> you might find that in the application under tests might have some common UI constructs.

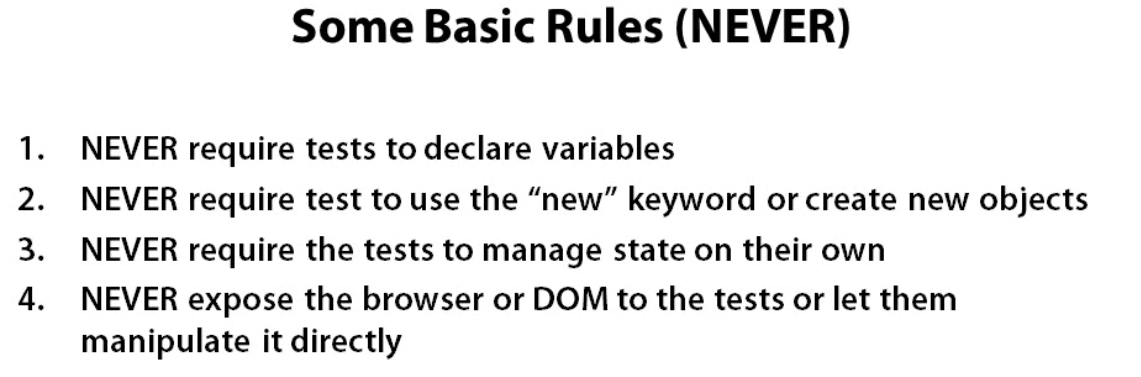
Lastly, the selenium interface

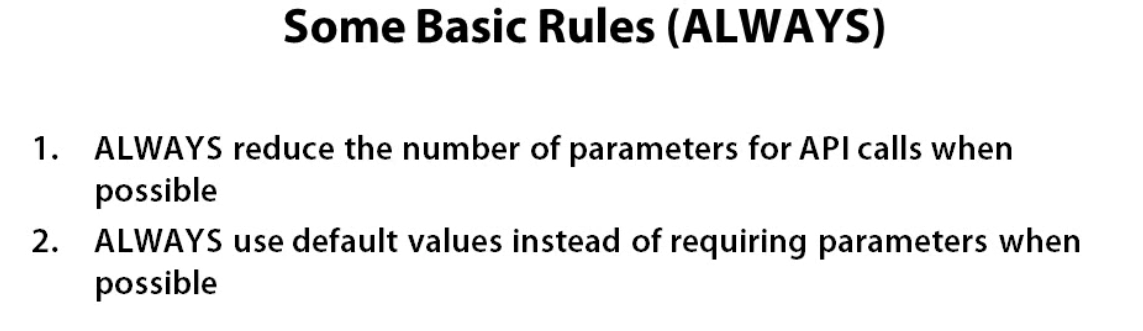
If you are using multiple automation tool then you might want to have a layer between he framework and the Selenium that can switch between the automation tools. Another scenario can be that you have a web app and a windows app under test, then surely you will have to use 2 different automation tools and need of this one extra layer.

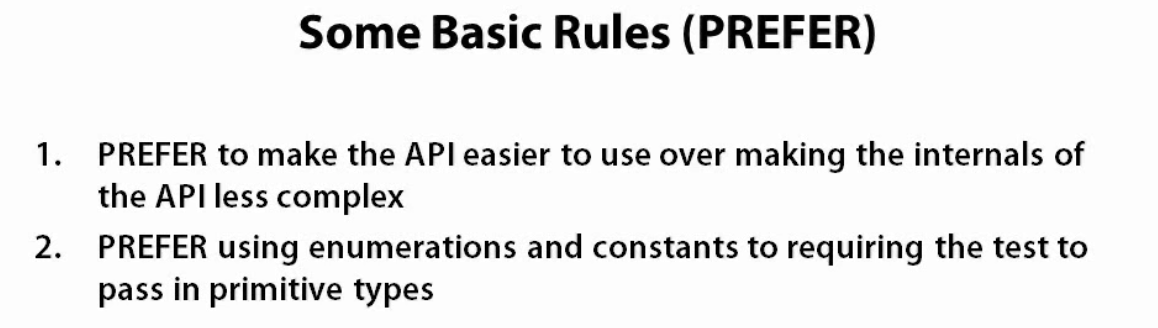
This layer will abstracts a way the actual automation tool so that you could swap it out or you don’t have to know what is being used, and you can write rest of your framework using that abstraction.

Automation frameworks should be easy to use coz if it isn’t then it will not be used at all.





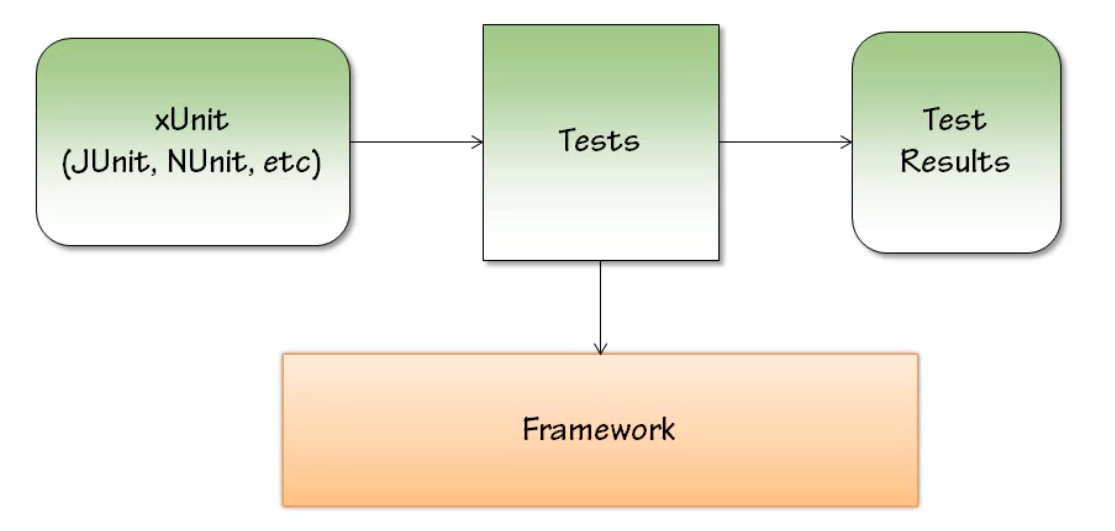




If you want to restrict the tests to use new variables and instantiate new objects then you need to use the static methods. It is generally thought that static methods should be avoided in production but in case of designing an automation framework, it is okay to use this.

The maintainability of the framework is not that important when compared to the ease of use the framework should have. And for this reason it is okay to used static methods in the framework.

Test Drivers are used to run automated tests. This is commonly the most common way that blackbox automated tests are automated or run, which is to use some king of unit testing framework.



You can use Junit, Nunit, C# etc as unit testing framework. Also you want to report the results of the tests.

Smoke tests usually covers the entire system but with a wide brush stroke which means not deeply. The smoke test is requires to test the basics of application as application runs on some hardware. It is to make sure that these basic tests do not put the hardware on smoke and make it unusable.

Smoke tests generally should be short and simple and the suite of these tests should not be too large so that the suite can be run quickly and detect any major problem with the system.



We will be keeping the test project separate from the automation framework to prevent crossing the lines of what code belongs to where and making them distinct.

Create a sample test to check whether selenium is working with our setup or not.

Install through nuget package manager the following in your framework project

1. Selenium WebDriver
2. Selenium Support

A test that never fail is completely worthless. It is a total waste of time both in creating it and running it. The purpose of the test is to find any failures of the system if they exist. so, we have to be sure that our tests are actually tesing for a specific thing.

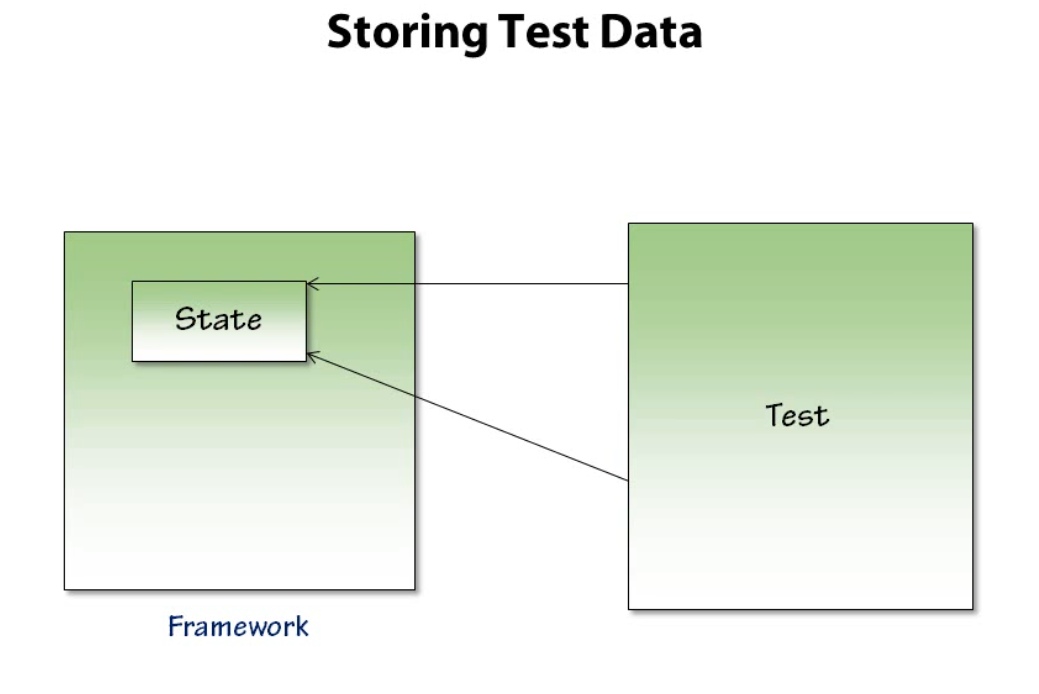
For all the pages in a web site where the url only differs with respect to the query string, the class will be same for this kind of pages and it is the responsibility of the class to handle the querystrings.

If you see a pattern in the implementation of menu then generalise it.

Constraining Options: Try not to leave anything open ended.

* Two good strategies
  + Nested Classes
  + Enumerations

These can be used to develop/generalize a navigation menu.



Use sequencing to remove the need for tests to have variables. This means that inside of tests we will have the framework store the state for us. So we just ask framework to get the data and track that for us. Then we do something with our tests and then again ask framework for the previous stored data and the current data.

Only create variable for those tasks that are multi-threading or multiple tests can call that thing simultaneously. Otherwise it is a good practice to let the framework handle the variables and tests can just ask from the framework.

Workflow classes

These are classes that help us to automate large parts of a system. These tasks in the workflow spread over multiple pages.

Let’s create a workflow to create a post for us and also remember the last post created so that we can easily delete that and all that stuff.

Always use default and dummy data in workflow classes.

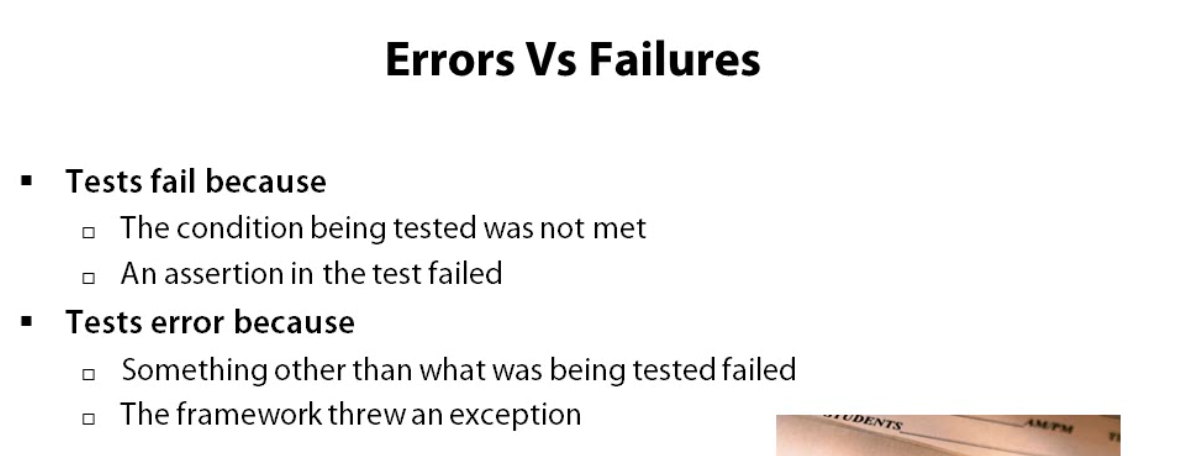
Anytime we can either default some data or use some dummy data.

For the tests, using the title and the body for a post is not relevant. We can just use some default data in that place to make things easier. So, we can create a dummy data that gets created randomly. All we want is that it matches.

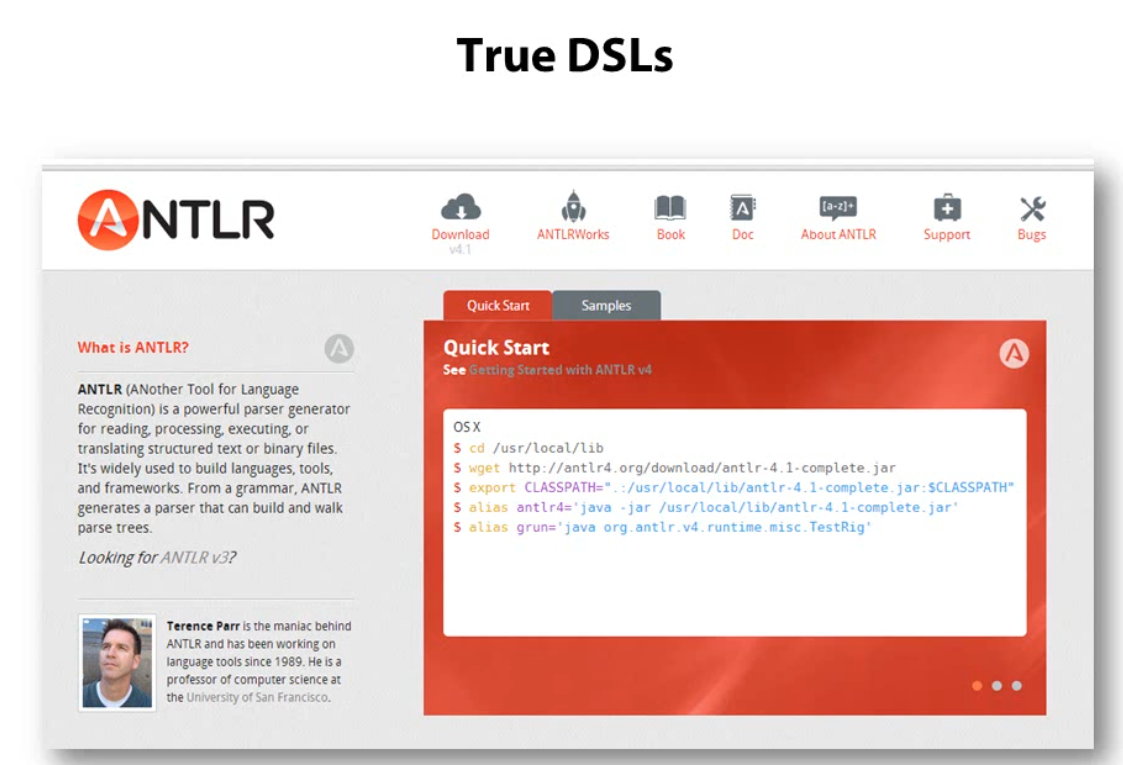
Best practices and tips

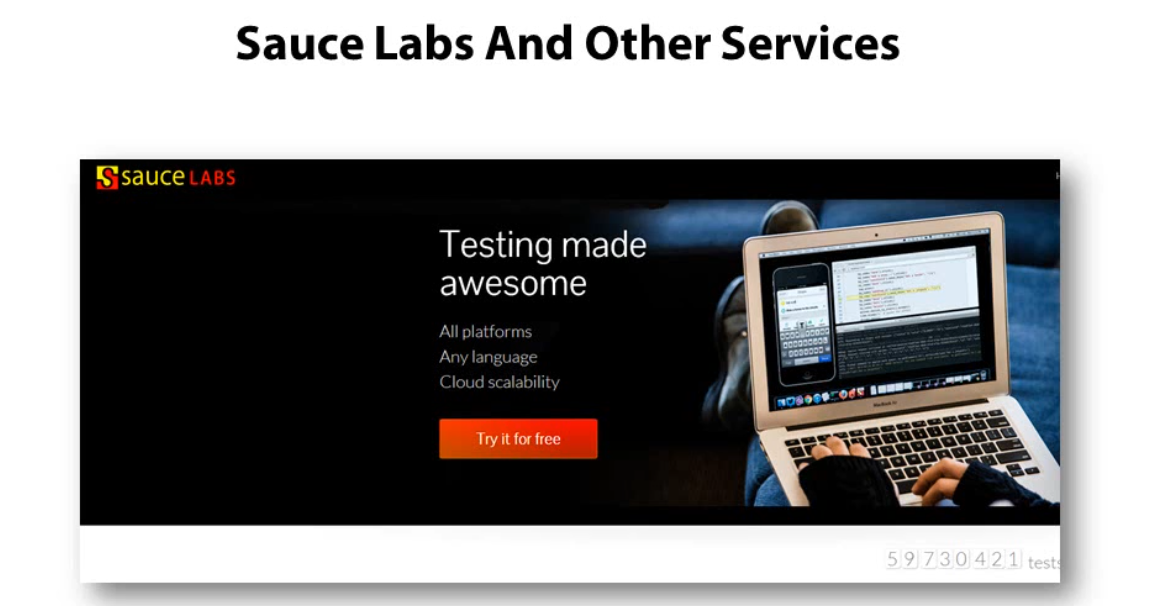
1. It is sometimes hard to automate flash things
2. Asserting image is hard as comparing two images is difficult.

In these kinds of cases, you just have to be pragmatic.

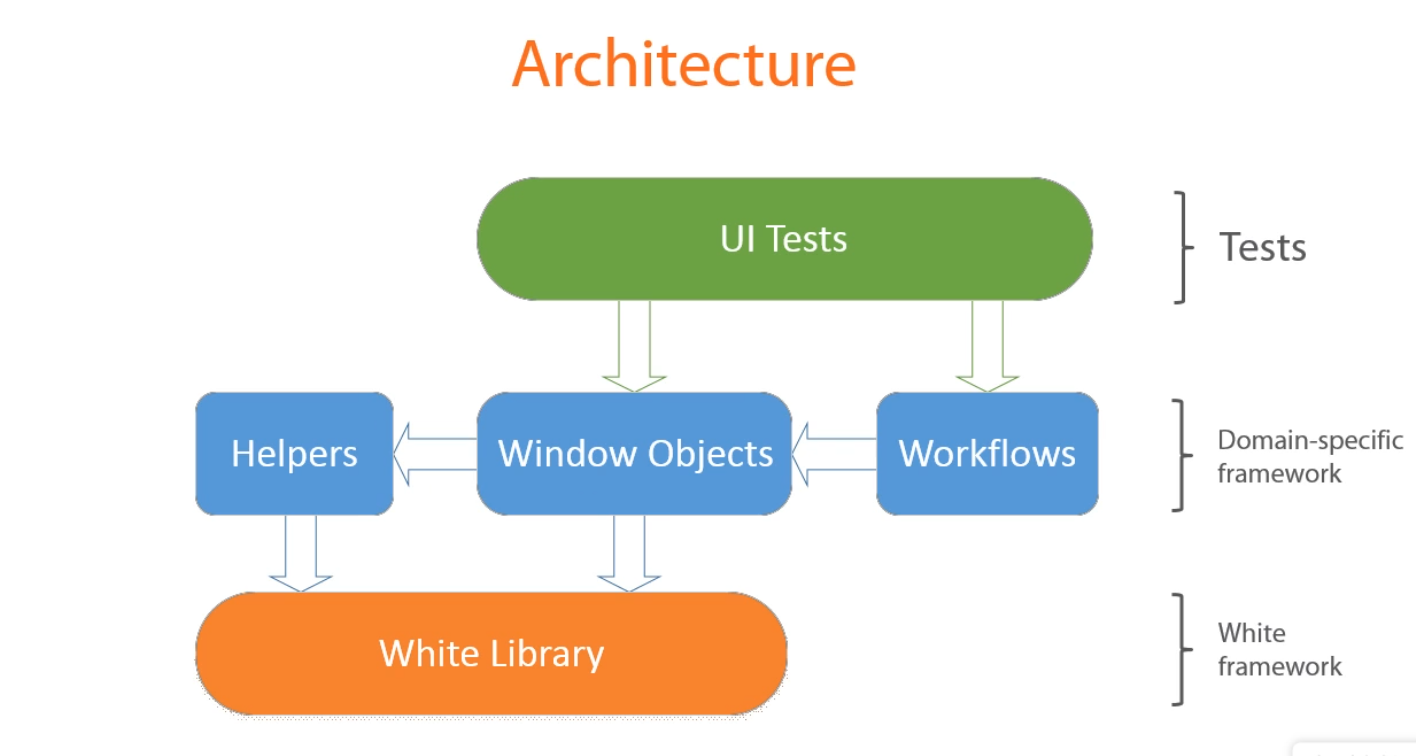


Domain Specific Language





Understanding the architecture of Windows application automation





The window object is the base class and it should not have any public members

Why can’t we replace all the properties to fields in the class. This is because

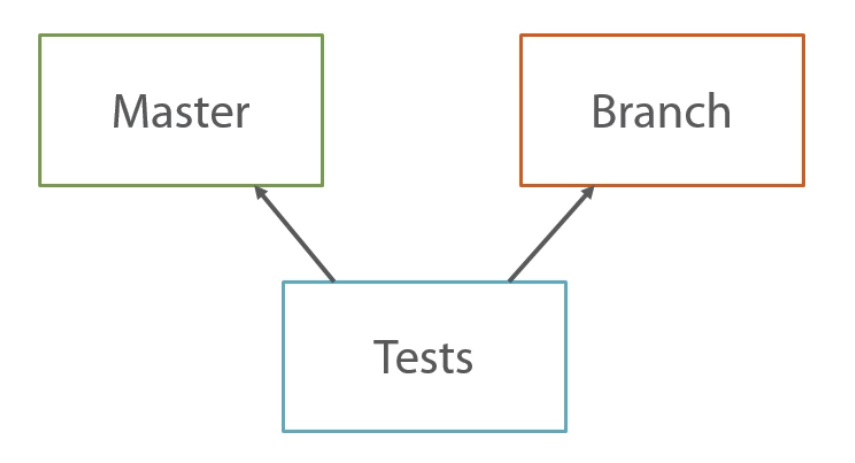
1. Fields look less verbose than properties
2. WPF doesn’t initialize all the UI controls on the screen at once. This is why using fields won’t work here.

Properties are only initialized only when we access and not beforehand. The fields get initialized in the constructor of the class and if those fields corresponding to the UI controls are not yet initialized by the application, it may throw an exception.

Automated UI test branching

There are various approaches for this.

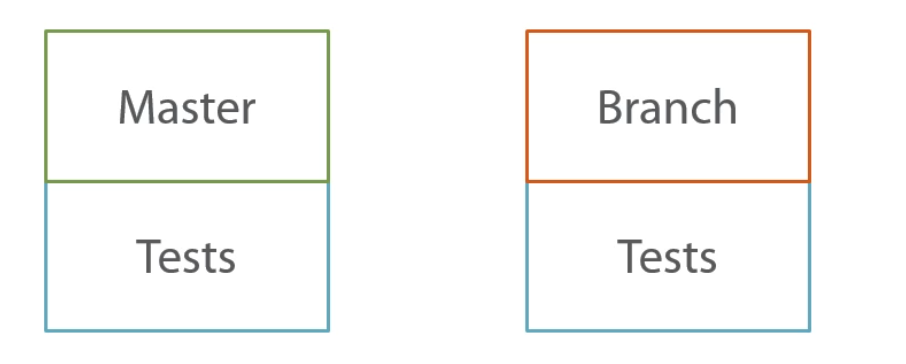
1. Store the tests in separate solution, aside from the code base of application that is being tested. This can work in small projects and is generally a bad idea.



Separate solution with UI tests ameks it extremely difficult to catch up on the differences in several branches of the application you have.

It might happen that you may introduce a breaking change in the UI in one of the branch.

1. Tests should be branched and merged along with the code base they cover. The easiest way to do this is to store the Ui tests with in the application solution. This approach allows us to keep a separate version of each tests for each branch and adjust them as soon as any break in change is introduced.



This also means that you have to merge the tests and resolve the conflicts in them just as we do with regular code.

Test structure

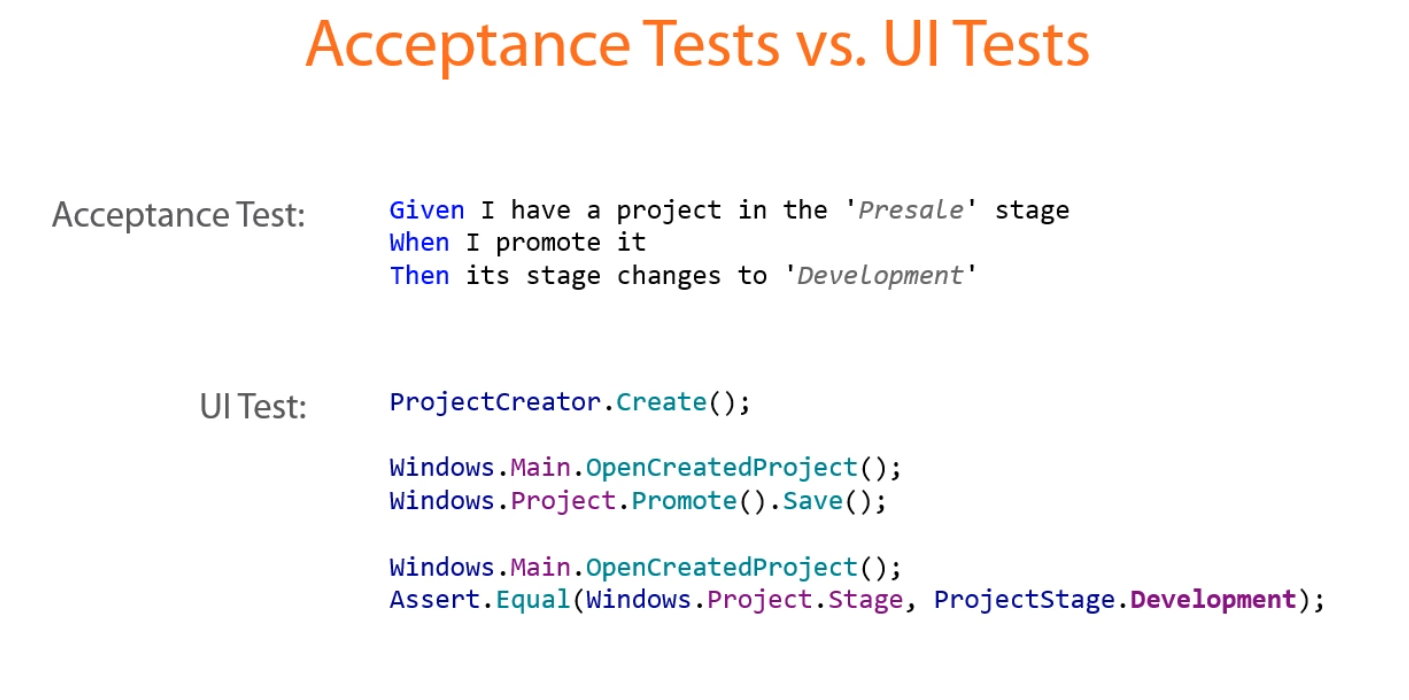
Tests consists of three parts

1. Arrange -> this is the part where all preparations takes place. Only this part if very common and is repeatable needs to be made a workflow.
2. Act -> don’t use workflow here.
3. Assertion

Acceptance Testing

It is a test to determine whether or not an application meets business requirements. When developing a feature in a software project there is always some set of criteria by which you can define that it is completed.

Acceptance tests are basically these criteria.

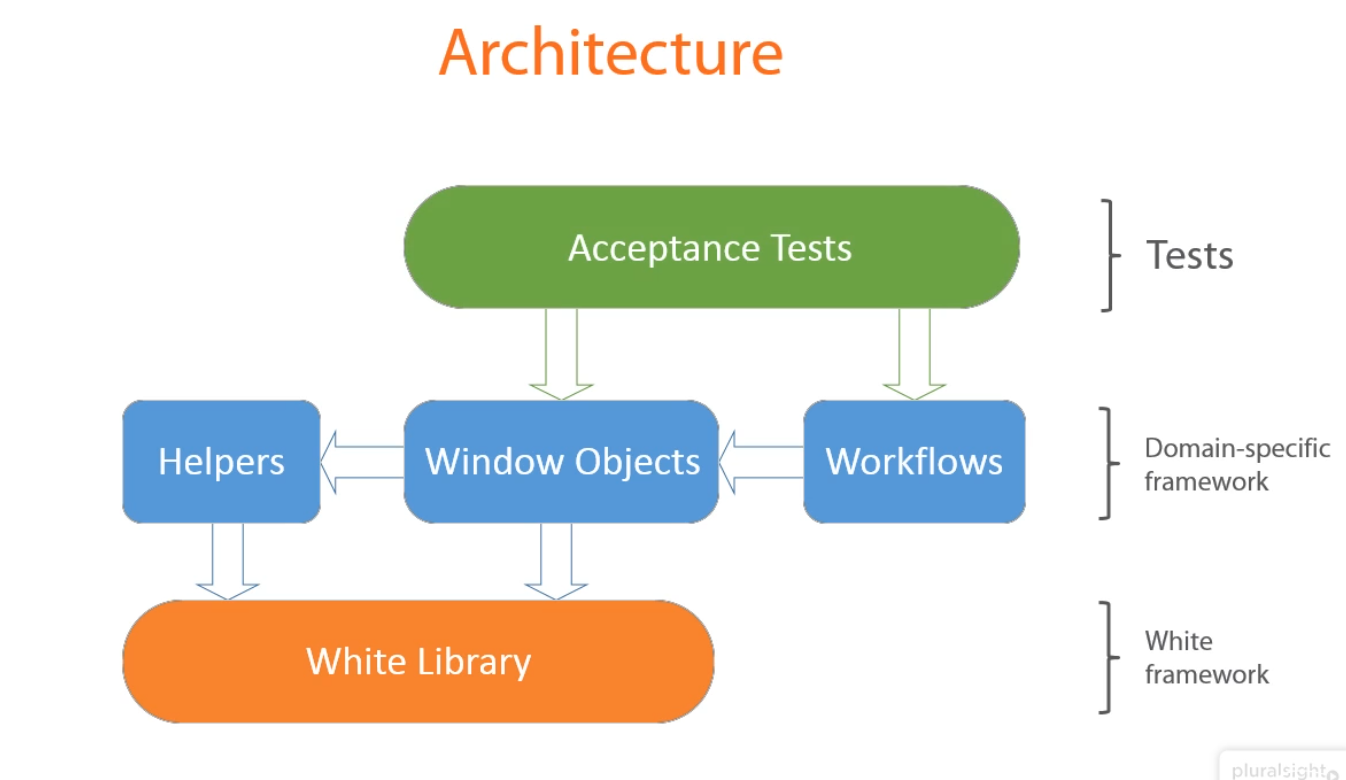


Create acceptance tests as the business requirement document may get outdated as the application evolves.

Acceptance tests notify you if any functionality changes as they fail. So, they will always remain updated with the application evolvement and never gets outdated.

Acceptance tests are easily readable by everyone.

Acceptance tests play the same role of that of the Ui tests.



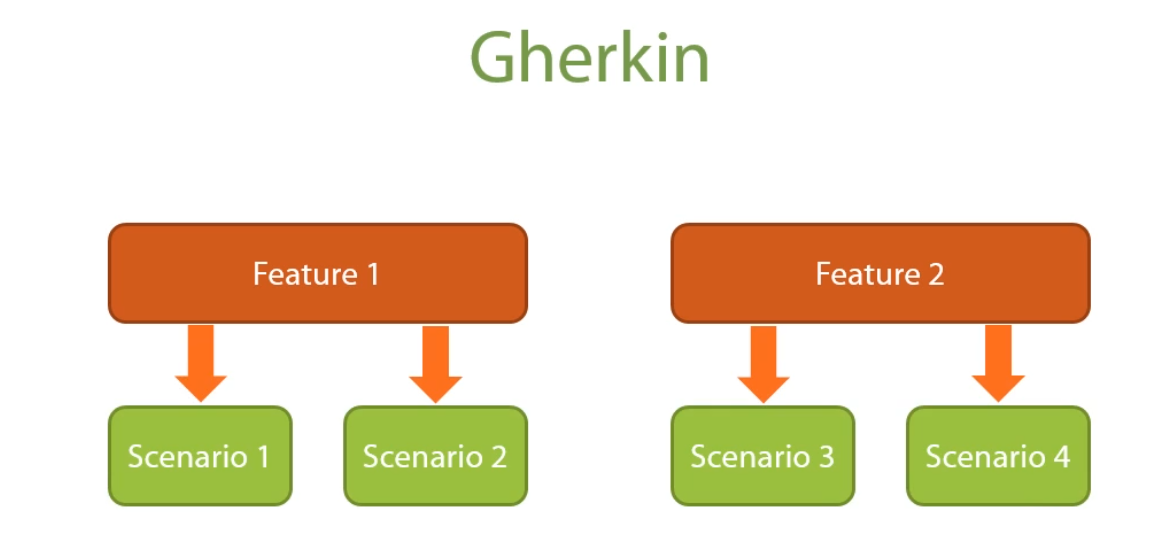
Keep either of the acceptance tests or the UI tests but not both for consistency reasons.

Specflow is used for creating acceptance tests. It is an open source tool that allows us to write specifications in a human readable manner. Those specifications are then translated to executable code which runs just like UI tests are.

Specflow provides us with a technique with human readable code to executable code.

To create tests in specflow we need gherkin. Gherkin is a language with a special structure. Tests written in gherkin consists of features that is a high level description of a functionality we will test and scenarios.

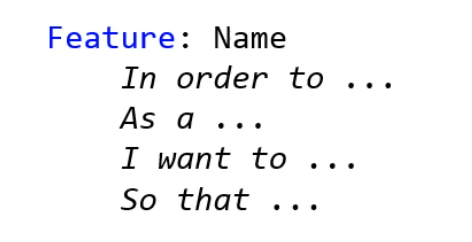
Scenarios are concrete use cases that belong to a feature.



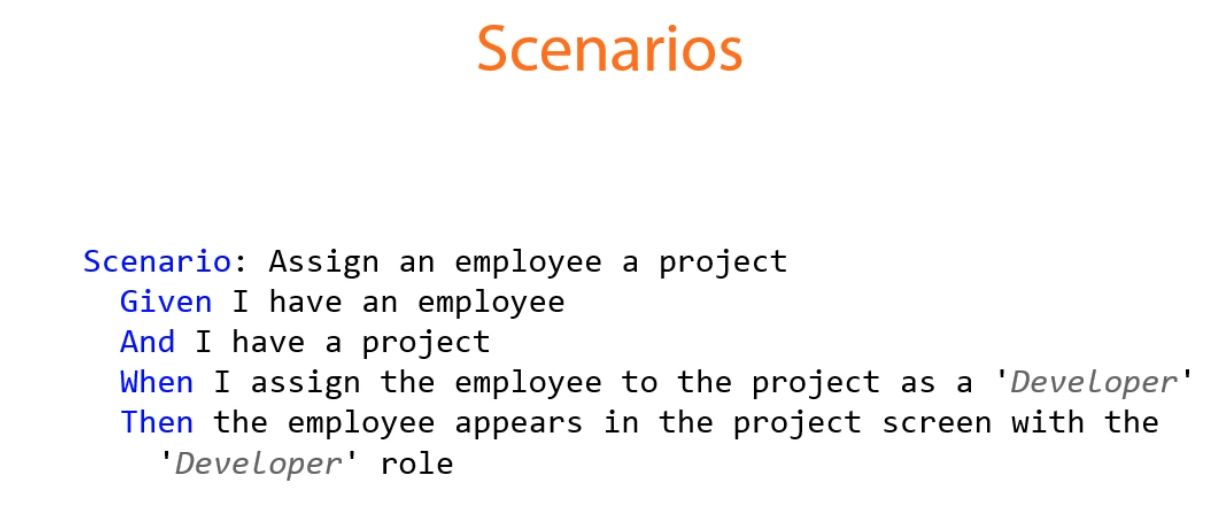
Feature are written in the below way. It has a name which is the first line. Second line starts with “In order to” and describes the particular reason for that feature. Then the role and then the action followed by the result.

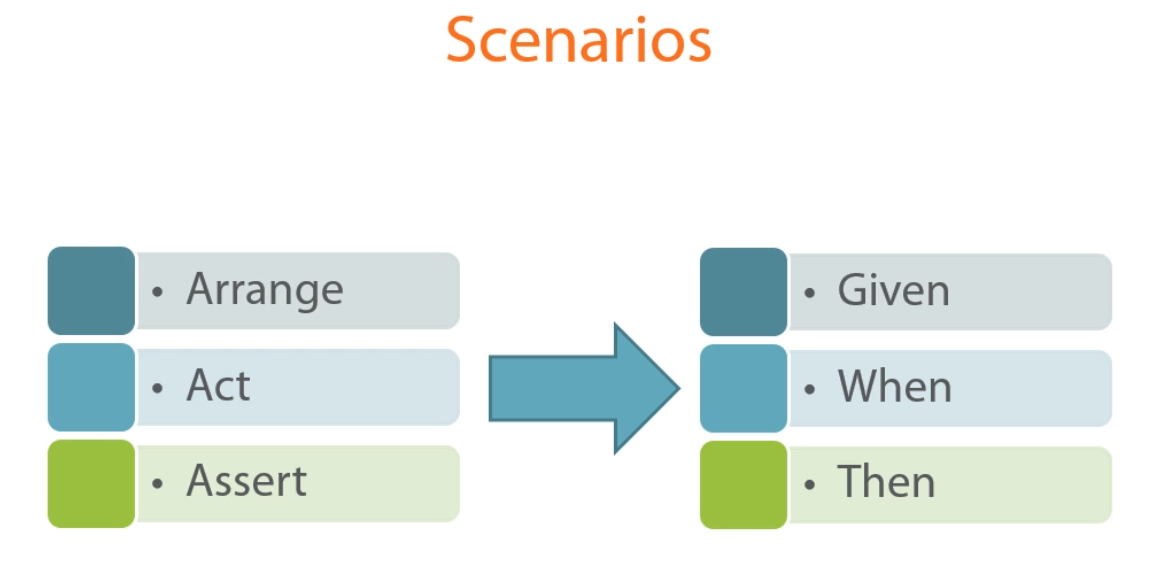


Feature structure



The last line is optional.





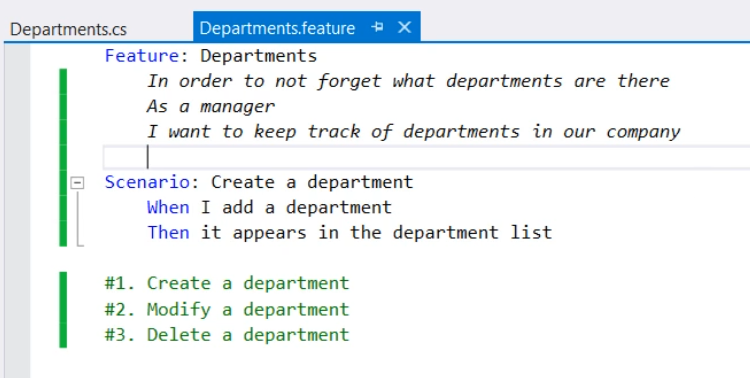
The UI tests written with Arrange, Act and Assert can directly be translated to Gherkin language.

Scenarios changes color from blur to black once the steps code are generated.

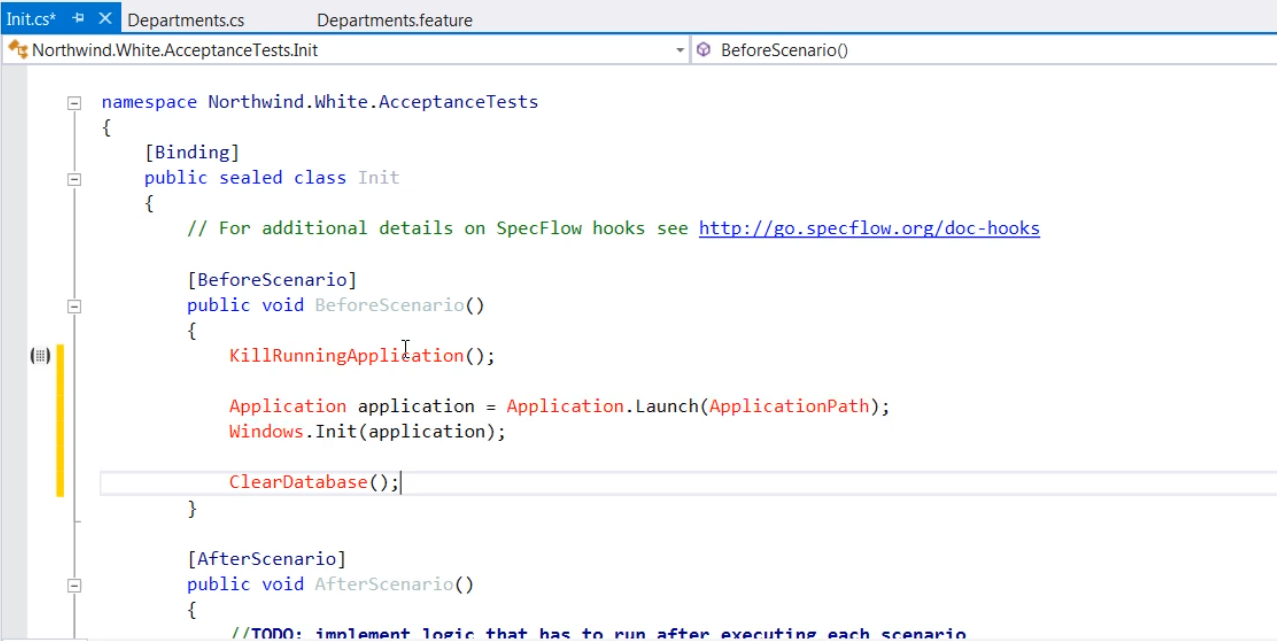
Use nuget package to install specflow

Use the specflow feature file for features

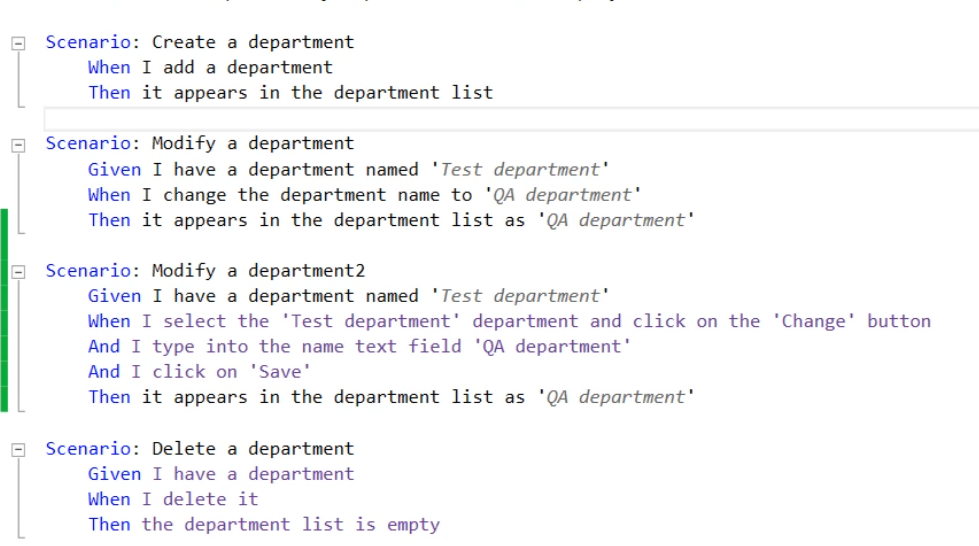
Use the specflow step definition file to bind the scenario to the code for the scenario in the feature.



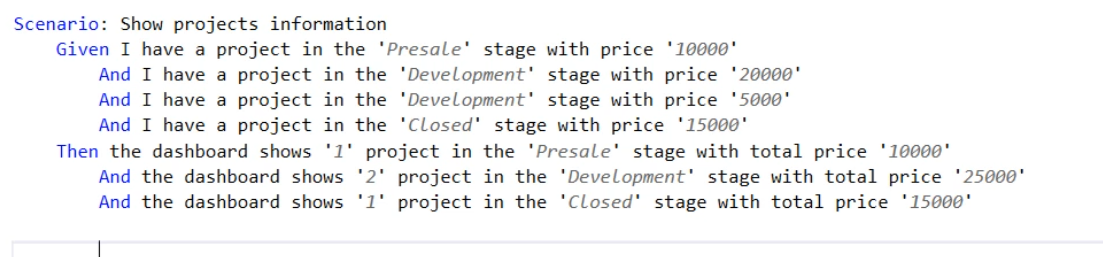
The initialiation logic is kept in the specflow hooks file. These file allow us to execute some logic before and after each scenario.



Don’t treat steps in scenarios as regular code.



Acceptance tests shouldn’t fall down to the implementation details as in the third scenario above.



There is a better way to do the above scenario by using the datatable feature in specflow.

