Specflow

* BDD uses Cucumber, Cucumber is a Ruby based tool.
* Specflow works like Cucumber for a .net project
* It is an open source tool.

BDD (userstories)

* User stories are not whole requirements they are instructions & desires to start a discussion.
* Given When Then is a BDD scenario, it is an illustration for requirements as they can be used to automate a test & is the documentation of behaviour and executable specifications
* Misunderstandings can come from user stories as we don’t know what we don’t know ( the unknown unknown)
* For example, is we see a user story referring to a new user. This new user can be someone who just registered, someone who hasn’t bought something. Someone who is registered but never had their session cached etc. We need clear details of how new this user is and everything In between. We need more information so there are no ambiguities
* Given When then are illustrating requirements.
* The examples work together to back up the user story.
* Examples can begin discussions to discover hidden requirements or rules

What is BDD

* Behaviour Driven Development
* Specification by examples
* Acceptance driven test development ADTD
* All concepts above are the same
* BDD is a process to help connect the different process bubbles
* Agile testing book touches on BDD to explain further
* Agile testing moves bug hunting to bug prevention
* Main route cause of bugs is from the requirements (56%)
* If we can spot holes in requirements before anything has been developed we will reduce bug frequency.
* We need to better understand requirements
* We need to better validate their correctness

Approach

* Discovery – Process of coming up with good examples ( taken from user stories)
* Formulation – Taking examples and adapting it to Given When Then
* Automation – Automating the testing using the scenarios
* Supplementary testing saves us time during regression
* Acceptance criteria – Rule
* First break down the rules
* Then create examples from these rules
* We are looking for checkable rules
* The goal is to collect as many examples until the team has a shared understanding of what the story is for.
* For each example we can derive more tests and exploratory tests around the example
* Example mapping is best done using the 3 amigos or some derivative there of so there’s someone from every role and we can work together with the PO, dev and tester.
* Example mapping is not a must have but a good way to collect examples and break down the user stories

Example Mapping

* Get the Given When Then
* Collate the rules – Verify the rules are correct (ask PO)
* Come up with examples derived from the rules
* Make the tests the examples we have collated
* Each scenario should be interesting and there should be a reason it exists
* Gherkin is the syntax
* Names come from Cucumber
* Problem V solution domain, the business works on the problem and the devs work on the solution
* GIVEN – this is the set up (what state do we need to be in to start testing)
* WHEN – This is the action that drives the result
* THEN – This is the result (the expected result) this is the decider on if the test has passed or not (Assertion)

Day 2 Notes using Specflow to automate

* The solutions folder within Specflow is for reference at a later date to see how and why we done what we done.
* Default login = login with default credentials
* Run time = test time
* The features on the right hand side of specflow in the solution explorer ( right hand side of visual) is the same layout as would be in Cucumber.
* Test Name is scenario name without spaces
* Specflow takes scenarios and makes them into tests
* Specflow CANNOT run the test. It can convert the scenarios to a testable test that you can run with a test runner ( resharper)
* WE have to tell Specflow which test runner to use for exa,ple Nunit – and we configure this is the App config file
* In the App.config file we can configure which runner we want to user (In the example of training we were using MStest)
* Specflow cant read English simply i.e.
  + “When I check the home page”
  + The above doesn’t mean anything to Specflow so we right click on the Given statement and we select “Generate step definitions”
  + The ‘Generate step defintiion’ shows you an example of what we can do.
* We need to name the class
* Sometimes we don’t need the Given example in the scenario
* Make step definitions reusable
* We need to call a parameter e.g. ( sting P0) – something understandable
* Don’t leave a ‘Pending’ statement – fix it with code ( right click and generate step definition )
* TDD – Test driven development
  + 1st step – write a failing test
  + 2nd step – Write a correct test (green)
  + 3rd step Refactor (make the code better structured and easy to read and maintainable.
* After refactor we would re run the tests until they are all green
* We could do automation at a controller level then actually manually check the front end of the site
* To generate the code we would write our scenario ( Given, When, Then) Then we would right click the Given scenario and select “Generate step definition
  + This opens the code page we can edit and amend for testing
* If Given When Then scenario appears purple this means the steps have not been defined yet. – In this case we would go to step definition and specify the correct details to get the test running
* If we have more than one scenario on our page you can right click the specific line of given when then and select ‘ Go to step definition’ which will direct the user to this specific step definition
* If we are in the step definition file we can right click the code and select show scenario / step usage.
* If you have a Given step it will generate a method
* On one of the statements in the scenario. If we right click & select ‘ Generate step definition’ which opens a box where we can generate the code for the automation to work
* If Given When Then is black ( the writing) then there are already step definitions specified
* The feature file is the window with the scenario in it
* When we use a regular expression it can take place of what we intend to test for example in the pizza project the regular expression used is a placeholder for the count of pizzas
* Int = integer ( number )
* We need to implement all step definitions for every step
* If we see ‘pending’ we need to fix the code and implement it
* .\* can be anything (regular expression)
* \d = digit. + = 1 digit more
* To assert that 2 tables match or that names match names in a table we can use a for loop ( see formula below)
* In specflow variable names always start with a lowercase but move on to bigger ones.
* A dot can call a method
* Assert.AreEqual (name.actual name): - this is an assertion
* We can reuse steps in multiple tests as they are checking the same thing
* CompareToSet
* Before we refactor tests its good practice to run the tests to see what the results are before the refactor
  + And every time you change something ensure to run a test so if it does fail we know with more clarity how and where
* Step definitions are global to feature files – so we can use them and see them across all files
* Auto complete can list things with the keywords
* You can create a new feature file to group all similar tests together i.e. all menu tests
* Group together sets per test
* If feature file gets too big e.g. > 10 scenarios > try break it up into smaller files ( of similar test goals)
* Specflow doesn’t recognise Given steps – Specflow looks for class
* Don’t forget the binding attribute ( hooks, step argument transformations and Scoped bindings)
  + The automation that connects the specification to the application interface has to be developed first
  + Binding classes and methods can be defined in the specflow project
  + The different bindings have to exchange data during execution
* Moving steps to a class
* [binding] Public class geek pizza
* We want to run tests in isolation
* If we can copy over to another feature file we may get an error
* Project is a container with many feature files
* You can have multiple projects in a solution
* Any code on specflow displaying a red squiggly line this means the system doesn’t know the data and needs to be input – If so we can right click and refactor
* A class will belong to its own file
* Specflow uses Selenium webdriver
  + We must create the webdriver

Var Webdriver = new

ChromeDriver();

webDriver.Navigate().GoToUrl(“http://)

* To share data between 2 methods in the same class we need to use Context Injection
* Hooks – Are a tag for calling the webDriver – which are also global
* We can tag each scenario that needs tested in web with a tag that is specified on the before & after hooks to use selenium
* Specflow doesn’t know about Selenium . It needs the package added which in the case of training is already configured.
* Method call = ()
* Debug starts the test
* Right click and inspect to see ID
* Send keys – Automate browser to type
* Use debug when running tests
* App.config file has all the examples of stuff in it and how you would change chrome to IE etc
* Day 2 we separated infrastructure code with functional testing code

Day 3 notes

* Automation works better after the development cycle
* It works better as a regression test
* Sprint planning is where we should break down user stories to rules to scenarios
* BDD is about the functional part of the system
* But we shouldn’t forget the non-functional things
* TDD – Devs comes up with the test and codes with that in mind. It is self verification of dev as they write usually a wrong test then write code until the test passes
* BDD – Tests come from the outside ( tester or PO ) an so there is more coverage. The dev can code from the tests that have been written
* Examples can also be used as tasks
* The scenarios and debugger would let a new start see code and to understand the code base
* FindElements – lists all elements
* CSS selector
* In ‘support’
* Test without debugging – we can check the error list to see what’s going wrong
* When writing assertions ensure the expected result is written first
* Rows and lists always start on line 0
* Be careful with CSS selector ensure it is the correct thing chosen
* Ensure CSS and assert look for same thing
* If there isn’t a good enough or a class name at all for a table on a site in CSS we can ask a dev to tag some on the site so it’s more testable and easier to test and set up
* Parsing – To split a file or other input into pieces of data that can be stored easily or manipulated. So we are splitting a string into parts and then recognizing the parts to convert it to something simpler than a string.
  + Parsing means accepting textual data as input and translating it into a format more easily processed by the computer. Almost all parsing is performed against human readable language (this is where we add syntax). An example of parsing is command line arguments which are parsed and often do not include readable words but are composed entirely of string data.
  + Parsing always involves syntax, and yes, syntax is just '*format*' but *syntax* is exclusively used to imply that a proper ordering of human understandable words exists.
  + Parsing always involves syntax ( which is just format) but syntax is exclusively used to imply that a proper ordering of human understandable words exist
* To find any CSS tags this can be found when ‘inspecting element’ on a web page
* The best use of Specflow for the team would be:
  + To create Smoke Tests for BAU for any little change (like ops in STV)
  + And to create a regression plan that can be quickly executed.
* A good scenario would be imperative and not declarative
  + Imperative = spelling out with as much detail as necessary to accomplish something
  + Declarative = only specifying (or declaring what needs done)
* A good example however is declarative as this means its less detailed can be changed more and adapted to different things and keeps it short and simple without specific examples so people will be more inclined to read them with less detail and information
* A hook is something that runs right before or after every example
  + If there’s a step that’s always going to be run we can create a hook that creates a new user on the site first before running anything,
  + We can define the hooks anywhere in the project or step definitions layers using the methods @before @after.
  + Cucumber hooks allow us to better manage the code workflow and helps us to reduce the code redundancy. We can say it is an unseen step which allows us to run our tests
  + An @before example would be
    - “clear database ();
* After changing any steps within step definition run the test to see if it passes or fails
* You can add something in the ‘background’ – Put before every scenario
* So, all scenarios are ran with the background stuff ran first. I.e. ‘Given I am logged in
  + Sometimes this doesn’t work as for example on the last test you don’t see the background information unless you scroll up the page
* You could use tags for each scenario so tag each one with @login to show for each scenatrio ran a user must be logged in
* The tag is visible and filterable ad searchable in specflow / visual studio
  + So we can run only tagged scenarios if for example the login page has changed
  + We can also tag scenarios as @wip ( workInProgress) to see what is a work in progress and so we know what we are working on and where we left off
* TD = Table data in HTML
* Parse – extracting information to a different form
* ? shortcut for if else statements
* - I =loop counter. The parse in this case would try and convert text to an integer to show as the int vaue
* To order something just type background and the scenario
* Control K C - makes the line a comment
* Before scenarios are good to keep in support folder – careful not to lose or forget them in all the documents
* Use Code KATA - to train on coding
* String = text
* Int = number
* Step argument transformation – Can change a word to a digt. For example a five to a 5 or V to 5 etc. Or yes / No to a Boolean or dollars to pounds etc
* BINDINGS – 3 types
  + Bind to a step
  + Bind to a hook
  + Bind to a step argument transformation
* Tools used to generate a report or PDF include :
  + Test execution report
  + Nunit
* Specflow.org > Documentation > guides on generating these files “step usage report” , Test execution report etc
* Some conversions are different
* Specflow BDD is good for functional specification testing over non-functional
* Use of specflow in the pipeline – same as using in other pipeline
* Do we need to keep structure UAT, Unit integration smoke etc – we can test all of the above using Specflow as they are tools and not a goal . we can test whatever with Specflow
* Generate pickles documents in support folder. Pickles is a thing that takes the specification on Cucumber and makes it test documentation

Testing different layers of a site

* When we are automating tests, we need to decide what level (layer) we want to test
* If we choose to test at the controller level the tests will run much faster
* If we choose the UI level it slows down the tests and they can break much easier (they are more brittle)
* We have 4 different layers from Browser, UI, Controller and Data.
* Testing in the controller level would be calling to actions
* You can test API level or end to end
* It is easier to test at controller level than at the UI level
* The UI level is much more brittle and likely to break
* UI testing however is good verification for what a user would be seeing
* UI level testing is slow with lots of dependencies nut it tests exactly what a user would see.
* Testing in controller level is fast & easier but doesn’t check what is actually on the front end of the site
* We could do both but it is time consuming
* We need to decide our strategy

C# notes

* A regular expression = (.\*)
  + A dot matches the character a star after the dot means we are saying (any text in any length)
* Regular expressions can be very complex – but we want to keep it simple and understandable
* An @ sign excludes it from the double quote.
* You put a string in double quotes i.e
  + Var s1 =”Hello \t \rn “
* A double quote ends the string
  + @”C:temp”
  + You would use @ to get rid of the double quotes

Computing basics

* Int = integer ( number)
* Class = definition of a thing / object . Class has dara & methods to modify
* Public void = a method of the class
* You cant paste a method into a method
* FOR LOOPS –
  + If we were wanting to compare data from 2 tables and ensure the order matches we would use a FOR LOOP
  + For (int I = 0; i < mylist.count; i++)
  + { Do something with my list [i] .. }
  + The above is the For formula ( this is the same across most languages)
  + The best way to write a FOR loop in Visual studio is to type in “for” and press the tab twice this will autocomplete it
  + The loop counter is the i++ in the formula
  + The length of the Specflow table can be retrieved by writing myTable.RowCount.
  + A row in the Specflow table can be retrieved by myTable.rows[i]
  + A cell within the row can be retrieved by myRow[“Header”] (or myTable.Rows[i] [“Header”]
* String – Username
* Int = P1
* To get element info right click element
* Integer = number ( whole)
* String = text field
* Floating point number = with a decimal point
* Floating double number