SpecFlow Masterclass: Lesson 3 – Automating scenarios with SpecFlow

concepts, patterns and useful libraries

by Gaspar Nagy

Your questions matter!

- Please write your questions about the content in the Q&A section of the webinar
- You can also vote the questions to prioritize them up
- The questions will be answered at the end, if not earlier

specflow

bdd newsletter: bddaddict.com

CAUTION!
bdd addict
on air

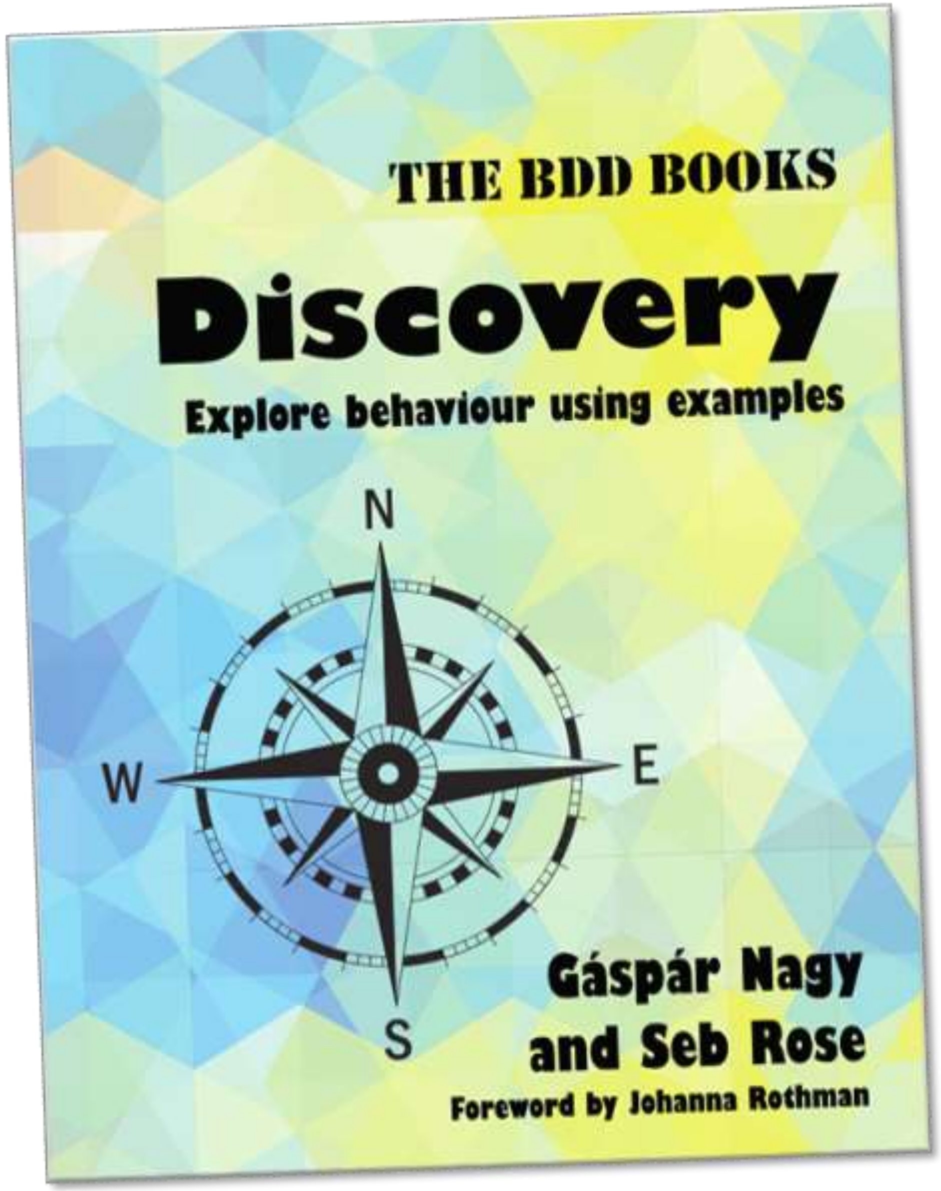
given.when.then

Gáspár Nagy

coach, trainer and bdd addict
creator of SpecFlow

gaspar@specsolutions.eu
https://specsolutions.eu
@gasparnagy

specflow

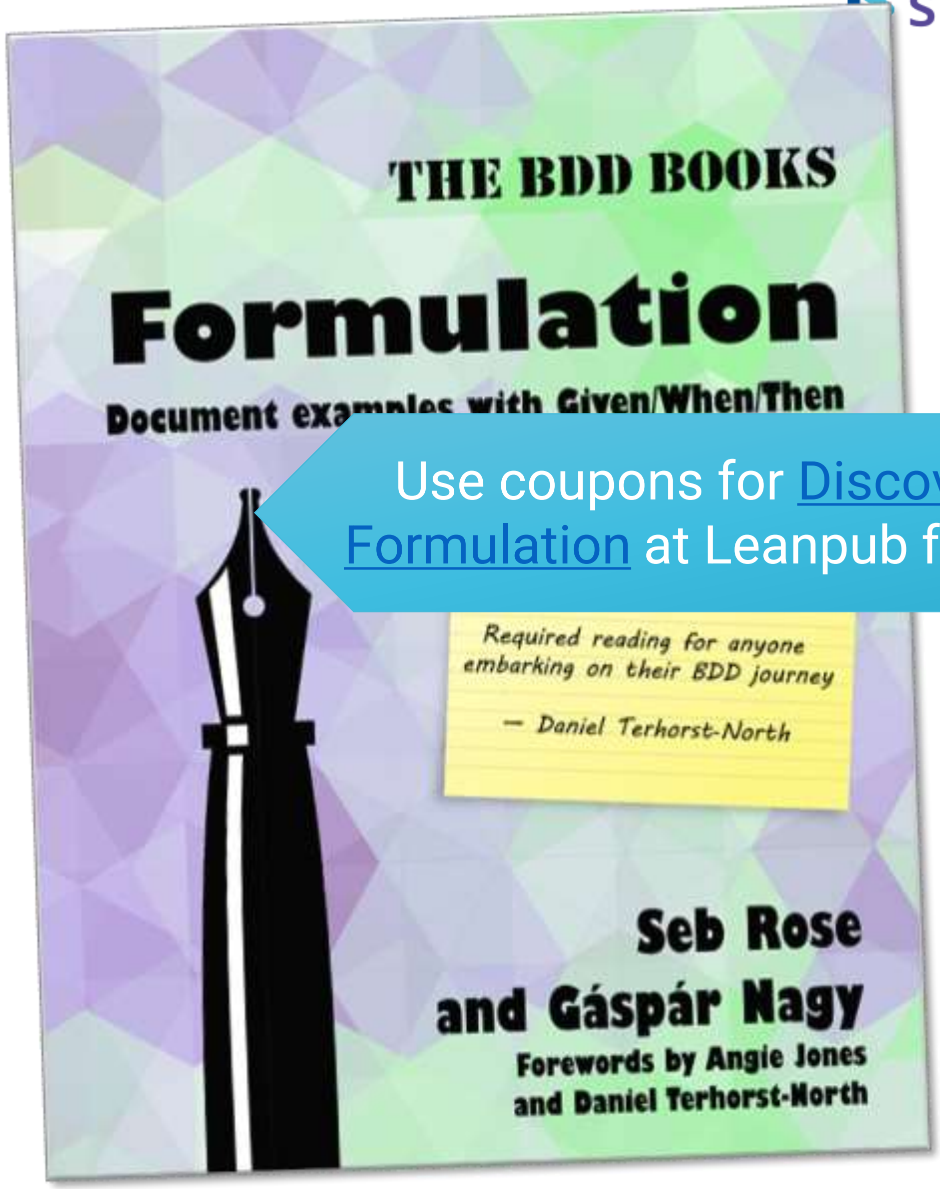

THE BDD BOOKS

Discovery

Explore behaviour using examples

Gáspár Nagy and Seb Rose

Foreword by Johanna Rothman


THE BDD BOOKS

Formulation

Document examples with Given/When/Then

Required reading for anyone embarking on their BDD journey

— Daniel Terhorst-North

Seb Rose and Gáspár Nagy

Forewords by Angie Jones and Daniel Terhorst-North

Use coupons for Discovery and Formulation at Leanpub for 30% off

Find it on Amazon & Leanpub through http://bddbooks.com!

SpecFlow Masterclass

Find my instructor-led online SpecFlow courses at https://www.specsolutions.eu/courses/

- Level up your BDD & SpecFlow knowledge
- Valuable for both existing BDD/SpecFlow users and for people who plan to introduce BDD
- Goes beyond simple “introduction”
- Most lessons don’t need coding skills to follow
- Extended with a longer Q&A

- Masterclass content represents personal view and opinions of Gaspar Nagy

specflow

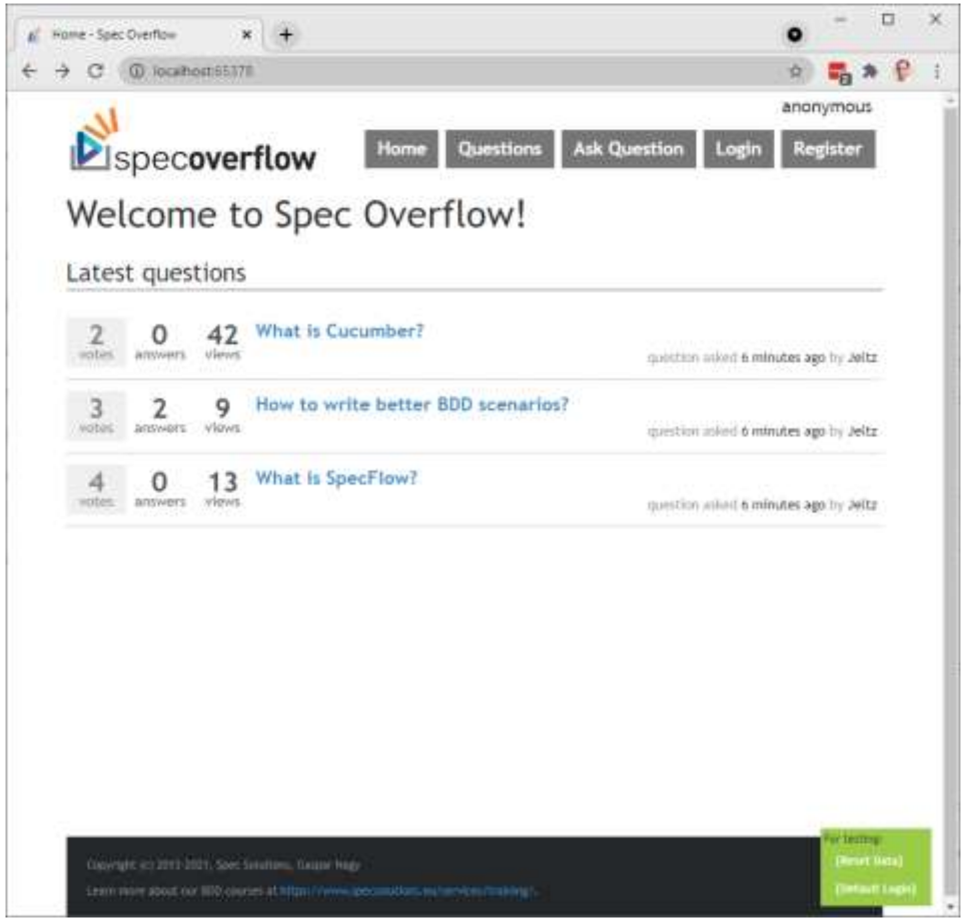
SpecFlow Masterclass – Lessons

- Lesson 1 – Practicing BDD with SpecFlow
- Lesson 2 – Formulation: Best practices for writing SpecFlow BDD scenarios using the Given, When and Then keywords
- **Lesson 3 – Concepts, patterns and useful libraries for automating scenarios with SpecFlow**
- Lesson 4 – Maintaining your BDD solution

Reflecting on the unanswered questions from yesterday

- Confusions with the term “Business rule”
  - This is pretty hard as “business” can be very different, but I don’t have a better term yet. The point is that BDD works well with rules that describe business requirements and not UI/UX, technology, security, performance, etc. requirements
- What is the better way to do only testing if we don't need Given-When-Then, you are mentioning that every time?
  - Use NUnit or some specific tool that focuses on your automation target (e.g. specific UI test tool)
- With the Business Rule #4 example, if validation may note be as part of BDD formulation, what process you do recommended to capture and validate them?
  - Depends on the project: maybe I would just do exploratory testing, but it is also possible to write a simple NUnit test that verifies that the right fields have the right attribute (e.g. Title has [Required])
- Can you use like the CMS like Umbraco to get data and use it on BDD?
  - If you need the data inside your step definitions: just call your CMS from C#
  - If you want to use the CMS to provide examples for Scenario Outlines (be careful with that!): you can make a SpecFlow plugin based on https://github.com/SpecFlowOSS/SpecFlow/tree/master/Plugins/SpecFlow.ExternalData

Demo project Intro: Spec Overflow



Welcome to Spec Overflow!

Latest questions

2 votes 0 answers 42 views What is Cucumber? question asked 6 minutes ago by Jeltz

3 votes 2 answers 9 views How to write better BDD scenarios? question asked 6 minutes ago by Jeltz

4 votes 0 answers 13 views What is SpecFlow? question asked 6 minutes ago by Jeltz

Copyright (c) 2011-2021, Spec Solutions, Gaspar Nagy
Learn more about our BDD courses at https://www.specsolutions.eu/services/training/.

- A simple Q&A site where people can ask questions and post answers
  - User management & Home page
  - Questions
    - List & Details
    - Ask & Answer
    - Vote questions & answers
- Technology:
  - ASP.NET MVC
  - Backend with REST API
  - Simple web frontend accessing the backend using basic jQuery
- On GitHub (open-source)
  - Explore, fork, play, contribute!
  - https://github.com/gasparnagy/SpecFlowMasterClass.SpecOverflow

DANGER
SOURCE CODE
AHEAD
{...}
NEXT 120 MIN
ROADSIGN
© 1985 THE GREAT AUSTRALIAN ROADSIGN Co. ® Made in Australia www.roadsign.com.au

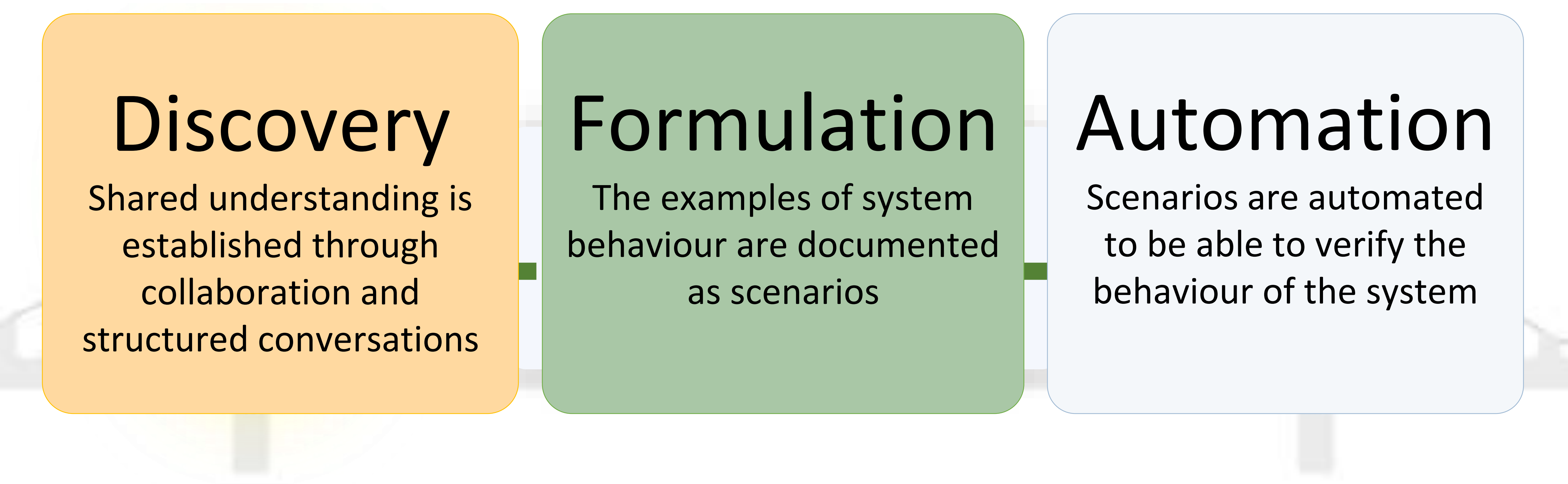
Lesson 3 – Agenda

- The 1 minute BDD summary
- SpecFlow core concepts
- State sharing, content injection and the SpecFlow object activation model
- Automating at different layers
- Feedback from the tests
- Assorted automation patterns and practices (homework)

specflow

The 1 minute BDD summary

3 practices of BDD



Discovery

Shared understanding is established through collaboration and structured conversations

Formulation

The examples of system behaviour are documented as scenarios

Automation

Scenarios are automated to be able to verify the behaviour of the system

SpecFlow

- #1 .NET BDD Framework, official .NET version of Cucumber

- Executes your scenarios using the automation code you provide
  - Execution is done with a help of test execution frameworks, like MsTest
  - The automation is provided for the steps (step definitions), reusable in different scenarios

- Open-source project, since 2009
  - https://github.com/SpecFlowOSS/SpecFlow/
  - 100 contributors
  - 23M downloads on NuGet
  - Maintained by Tricentis with a dedicated SpecFlow team

specflow

SpecFlow core concepts

Approach, setup, main features

SpecFlow concept & project setup (DEMO #1)

- SpecFlow generates tests from scenarios and coordinates their execution ("runs them")
- To execute the scenarios, a Test Execution Framework is used, and SpecFlow supports several:
  - MsTest, NUnit, xUnit
  - SpecFlow+ Runner
- When you setup your SpecFlow project, you need to choose by adding the appropriate NuGet package:
  - SpecFlow.MsTest, SpecFlow.NUnit, SpecFlow.xUnit…
  - … and the project should be prepared to run MsTest/NUnit/xUnit tests otherwise: use "MsTest/NUnit/xUnit Test Project" template
- Usual folder structure (convention): Features, StepDefinition, Support

Visual Studio integration (DEMO #2)

- Visual Studio (and other IDEs) doesn't have built-in support for SpecFlow – An extension has to be installed
- For Visual Studio, this can be done from the extension manager (search online for "SpecFlow")
- Currently two extensions are available (maybe they will merge)
  - SpecFlow for Visual Studio 2019 – the official one
  - Deveroom for SpecFlow – an attempt to rewrite, supports recent Gherkin, but you need to compile to get step definition matches
  - The two has similar feature set, both free & open-source
    - Templates, syntax coloring, generating step definition snippets, navigation
  - From VS2022: SpecFlow for Visual Studio 2022 (based on Deveroom)
- For other IDEs:
  - Rider: SpecFlow for Rider
  - Visual Studio Code: Cucumber (Gherkin) Full Support (see als https://docs.specflow.org/projects/specflow/en/latest/vscode/vscode-specflow.html)

Step definitions (DEMO #3)

- The core building block of making the automation solution
- One step definition can be used for steps in multiple scenarios (automation reusability)
- SpecFlow finds the step definition for the step during execution based on the expression in the Given, When or Then attribute
  - Regular expression or Cucumber expression
- The step can contain parameters that are passed to the step definition methods
  - Conversion applies for standard types (int, bool, Enum, etc.)
  - The conversion can be extended
- Step definitions are “global” – they can be used from any feature file within the same project
  - You can also use step definitions from other, shared projects
  - And don’t forget the [Binding] attribute

Regular vs Cucumber Expressions (DEMO #4)

- For long time Regular expressions were the only way to specify the mapping between the step definitions and the steps (in all Cucumber-family tools, including SpecFlow)
- Regular expressions a bit too complex for the task, so the concept of a simplified expression language has been defined: Cucumber Expressions
- Cucumber Expressions are
  - Simple, but backwards compatible (you can still use Regex if needed)
  - Parameter placeholders: {int}, {word}, {string}, {float}, and extensible
    - {string} uses “quoted” or ‘quoted’ parameters by default
  - Optional, alternating text: user(s), user/client
- With SpecFlow, you can use Cucumber Expression using an additional plugin: CucumberExpressions.SpecFlow.3-7
  - Navigation only works in Deveroom

Data Tables (DEMO #5)

- Data tables are special step parameters of tabular data
- Data tables are not expressed in the step definition expression, but the stepdef method has an additional parameter of type Table
- Table is basically a list of key-value dictionaries
  - Rows[0] – get a particular data row (header excluded)
  - Rows[0][“title”] – get a particular cell value within a row
- Assist helper methods can make the use of them easier:
  - CreateSet, CreateInstance – create object(s) from a table
  - CompareToSet, CompareToInstance – compare a table with object(s)

specflow

Hooks (DEMO #6)

- Another way to provide automation logic – one that runs at specific events
  - Before/After scenario execution
  - Before/After test run
  - Before/After step
  - Etc.
- They are “global”, just like step defs, but can be limited to scenarios tagged with a particular tag
  - And don’t forget the [Binding] attribute
- The execution order of the same type of hooks is undefined, but you can set it with the “Order” parameter
- The [After] hooks are executed even if there was an error

specflow

State sharing

Context injection and the SpecFlow object activation model

Sharing state via fields (DEMO #7)

- The step definitions are autonomous automation building blocks but they need to “communicate” with each other: sharing state (sharing data)
  - When the user checks the home page – produces data (the data that was displayed to the user)
  - Then the home page main message should be … – consumes the data
- The easiest is if the two step definitions are in the same class: they communicate through instance fields
- Safe, because
  - SpecFlow recreates the step definition class instance for each scenario execution
  - parallel running ones have their own instance

Sharing state between step definitions in different classes: Context injection (DEMO #8)

- SpecFlow creates the step definition class instances for each scenario execution
- When this creation happens, it first checks the constructor parameters and tries to “resolve” the requested objects
  - Create them, if they haven’t been created for this scenario execution
  - Reuse them if they have been created already for this scenario execution
  - These objects are disposed once the scenario finished, just like the step definition instances
- Using this you can create data container classes that hold the references to the data you want to share
  - These are usually called as “context objects”
  - You “inject” the context object into any step def class where the shared data is needed
- The feature of SpecFlow that allows this is called Context Injection
  - It is essentially a simple implementation of Dependency Injection

Using context injection for injecting automation logic (DEMO #9)

- While context injection was designed to share context objects, it works with any object also with ones that contain automation logic
- The resolution is recursive, so the dependencies of the dependent object are also resolved once SpecFlow creates them
- By implementing IDisposable, you can even get notified about the scenario finish, so you can use this as an alternative to hooks (before – constructor, after – dispose)
- This allows to build up a decent, structured automation code base

Advanced injection practices

- The framework that provides the context injection feature is a mini DI framework, called BoDI
- For advanced usages, you can replace it with your own favorite DI framework using a SpecFlow plugin
  - Plugins provided for Autofac, Microsoft.Extensions.DependencyInjection, Castle Windsor
- But basic configuration can also be done with BoDI:
  - You can access the container by adding a dependency to ScenarioContext (or IObjectContainer), usually from a hook class
  - You can register types with RegisterTypeAs<Type, Interface>()
  - You can also register concrete instances with RegisterInstanceAs<Interface>(obj)

Quiz time!!!

specflow

Automating at different layers

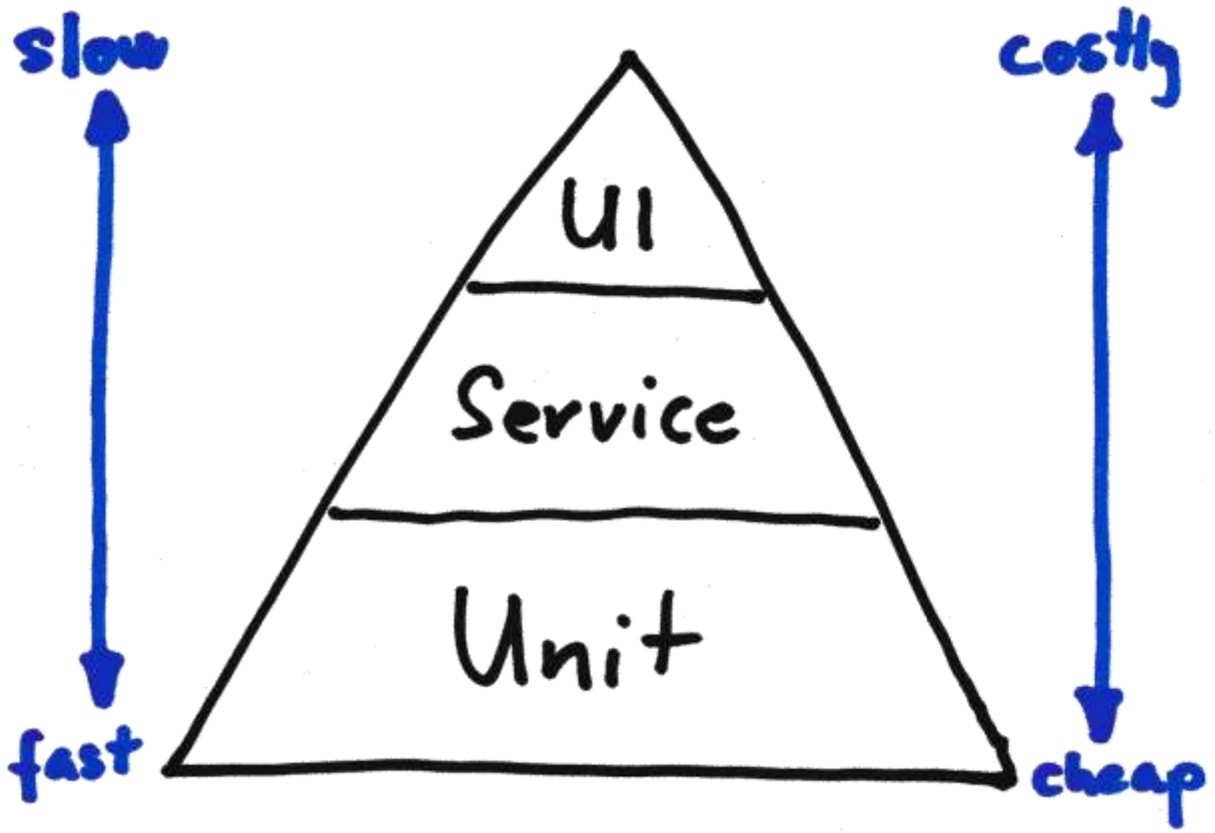
Domain classes, Web API, Web UI, Desktop UI, Mobile, etc…

Automating at different layers

- The scenarios are formulated using business language in an automation-agnostic way
- This enables automating them in various ways
    - Through the user interface – verifies what the user sees, but slow, brittle and costly
    - Through the frontend/backend interface (e.g. REST API) – generic, but does not test the UI
    - Through some service interface – specific to the service, but faster
    - Through the application domain (calling methods on domain classes) – fastest, but needs access to the code of the application
    - Do we stub/mock parts or not?

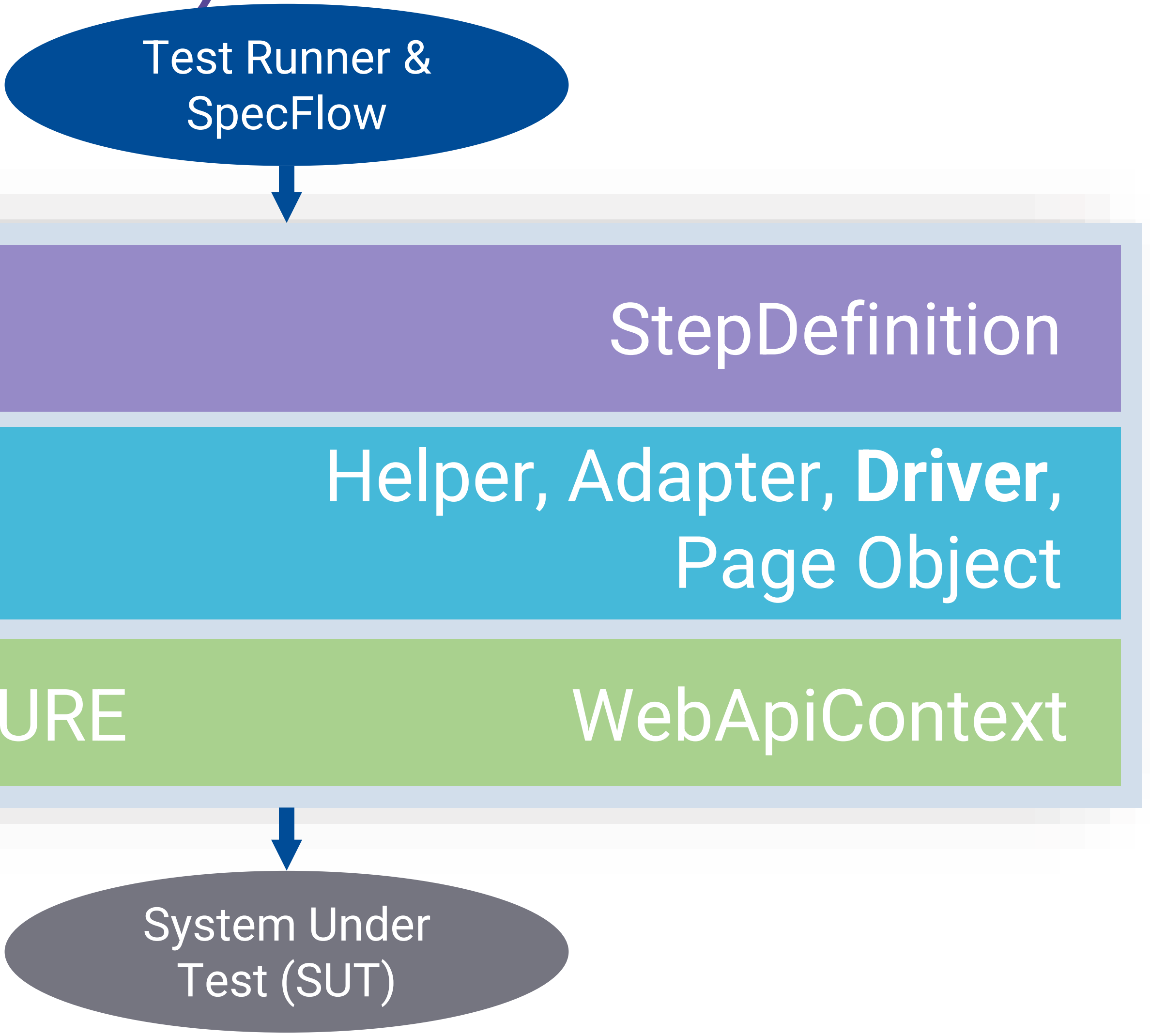
specflow

Automating at different layers



- Automating domain classes is easy
- Automating external interfaces might raise interesting questions
  - You only have access to the selected interface and cannot tweak internals
  - You need to manage the infrastructure for testing (start/stop application, browser, etc.)
  - Needs more configuration
  - Code duplication is more problematic
  - Harder to make it diagnosable (logging, sanity checks, etc.)
- Introduce automation layers!

Layering the automation solution (Demo #10: Web API)



The “Driver” pattern

- Driver – a class that is able to “drive” (automate) a part of the application
- It focuses the automation concerns – no testing decisions there
  - It knows how to automate the part and knows how to interpret the results, but it does not make any decision whether that result is the expected one or not
- Encapsulates the automation details and only expose domain classes on the interface (no HTTP status, CSS selector can escape)
- People usually say: no assertion in drivers (generally true), but…
  - You can have sanity checks (ensuring that the system is operational)
  - You can add assertion helper methods to the driver class, but they must be called from the testing layer
- The “Page Object Pattern” is a driver implementation for UI

Code walkthrough for Web API automation (DEMO #11)

- SpecFlowMasterClass.SpecOverflow.Specs.API project
- WebApiContext, WebApiHooks, AppHostingContext – infrastructure
  - Handle application start/stop – singleton for the test run
  - Handle HTTP calls, cookie management, json serialization, sanity checks, logging
- UserApiDriver, AuthApiDriver, etc – automating specific endpoints of the application (/api/user, /api/auth)
  - Use “input” & “model” classes on the interface

Code walkthrough for Web UI automation (DEMO #12)

Check SpecFlow.Actions.Selenium plugin as well:
https://github.com/SpecFlowOSS/SpecFlow.Actions

- SpecFlowMasterClass.SpecOverflow.Specs.WebUI project
- BrowserContext, BrowserHooks, BrowserFactory – infrastructure
  - Handles browser stat/stop (separate browser for each parallel thread)
  - Handles “Base URL”
  - Handles waiting problems
- RegisterPageDriver, QuestionDetailsPageDriver, etc – automating pages of the application
  - Use Page Object Pattern
  - Use “input” & “model” classes on the interface (e.g. “parse” methods)

Desktop & Mobile APP UI automation options

- Conceptually they work very similar to the API/WebUI solution
  - You need to take care that the application is started/stopped
  - You need a library to automate the application
    - Desktop: e.g. Windows Application Driver
    - Mobile: e.g. Appium
  - Many of these libraries use the WebDriver interface
- As the execution of these interfaces are more problematic and slow, it makes even more sense to consider alternative automation options, at least for the most of the scenarios:
  - Automate them through the backend REST API
  - Automate them through the MVVM interface (at ViewModels)
  - Separate the UI logic from the actual frontend code (e.g. Xamarin)

Even the domain class automation might get more tricky

- Handling of authentication scope (AuthContext)
- “Attempt” pattern – enables asserting on action errors
- Logging – log domain actions

The “driver” pattern makes sense even for domain class automation!

Dynamic drivers

- As the driver classes encapsulate the automation details and only use business details on the interface, you can have multiple implementation of the same driver interface, using different automation targets
- With this, the same tests can be executed on multiple interfaces
- The current interface target can be selected with configuration or environment variables

specflow

Feedback from the tests

How to make test diagnosis easier

Feedback from tests

- Providing useful diagnostic information about failures is very important for BDD scenarios
- Take care of the error message from your assertions! Try better assertion libraries.
- Test your tests:
  - Never trust the test that passed immediately
  - Do exploratory testing to verify your tests (try with wrong expectations, inject bugs to the system)
- Use [AfterScenario] to save log files, screenshots, etc.

specflow

Assorted automation patterns and practices

From the Spec Overflow code base

Life is like a box of chocolates

You never know what automation pattern you’re gonna need.

@gasparnagy

The list of patterns used by Spec Overflow (1/2)

- Current object pattern – manages “the one thing” the scenario talks about (e.g. the current question)
  - QuestionContext class
- Driver pattern – encapsulates the automation concerns of a specific area
  - *Driver classes
- Attempt action pattern – enables asserting on action errors
  - ActionAttempt class and derived classes
- Data mothers – generates test data
  - QuestionMother class
- Data classes – convert or compare them with data tables
  - *Data classes in Support folder

The list of patterns used by Spec Overflow (2/2)

- Domain defaults collection class – collects all default values/behavior in one place
  - DomainDefaults class
- Authentication context – manages authentication session (can also be used for multiple parallel user sessions)
  - AuthContext class, Scenario: Multiple users can vote for the same question
- Data reset patterns – resets the database to a baseline before the tests run
  - DatabaseHooks class
- Asserting error messages – allows you to assert on the message key, that is replaced by the real message text for the comparison
  - ErrorMessageProvider class (concept only)
- Test logger – encapsulates generation of test logs
  - TestLogger class (concept only)

specflow

Wrap-up of Lesson 3

Automating scenarios with SpecFlow

Lesson 3 – Wrap-up

- The 1 minutes BDD summary
- SpecFlow core concepts
- State sharing, content injection and the SpecFlow object activation model
- Automating at different layers
- Feedback from the tests
- Assorted automation patterns and practices

Key Facts

- SpecFlow is a simple tool with only a handful different features
  - The key is how you are using it!
  - Watch for the state management
- Take care of the testing strategy: what you are test & how
  - Testing all scenarios through the UI is rarely a good solution
- Treat automation code as first-class citizen
  - Apply appropriate layering
  - Test your tests
  - Seek for reusable patterns and apply them when needed
  - Choose your helper libraries
- Encourage collaboration between dev and test to make a good automation solution!

specflow

Thank you!

Gáspár Nagy
coach • trainer • bdd addict • creator of specflow
"The BDD Books" series • http://bddbooks.com
@gasparnagy • gaspar@specsolutions.eu

Use coupons for Discovery and Formulation at Leanpub for 30% off

Find my instructor-led online SpecFlow courses at https://www.specsolutions.eu/courses/