specflow

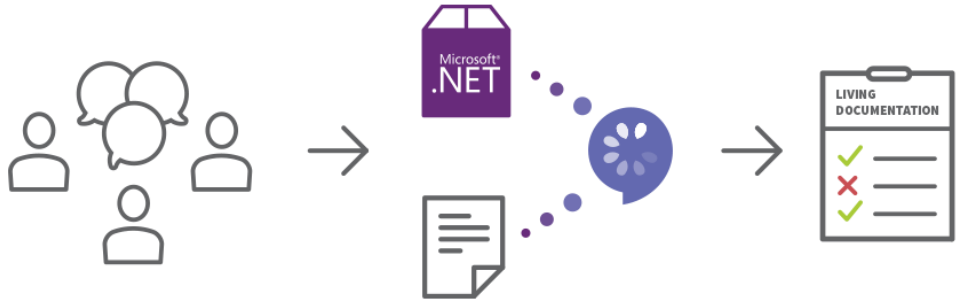
Samuel Janek

Content

- Introduction
- How to get started
- Main components
  - Feature files
  - Step definitions
  - Hooks
- Dependency and context injection
- Execution model

What is SpecFlow?

- **Behaviour-driven development** (BDD) **framework** for .NET based on Gherkin language
- Integration with well-known tools and technologies:
  - Azure DevOps, TeamCity
  - MSTest, NUnit, xUnit
  - Visual Studio, Visual Studio Code, Rider
- Usage:
  - UI testing
  - E2E business process testing
  - API automation and testing

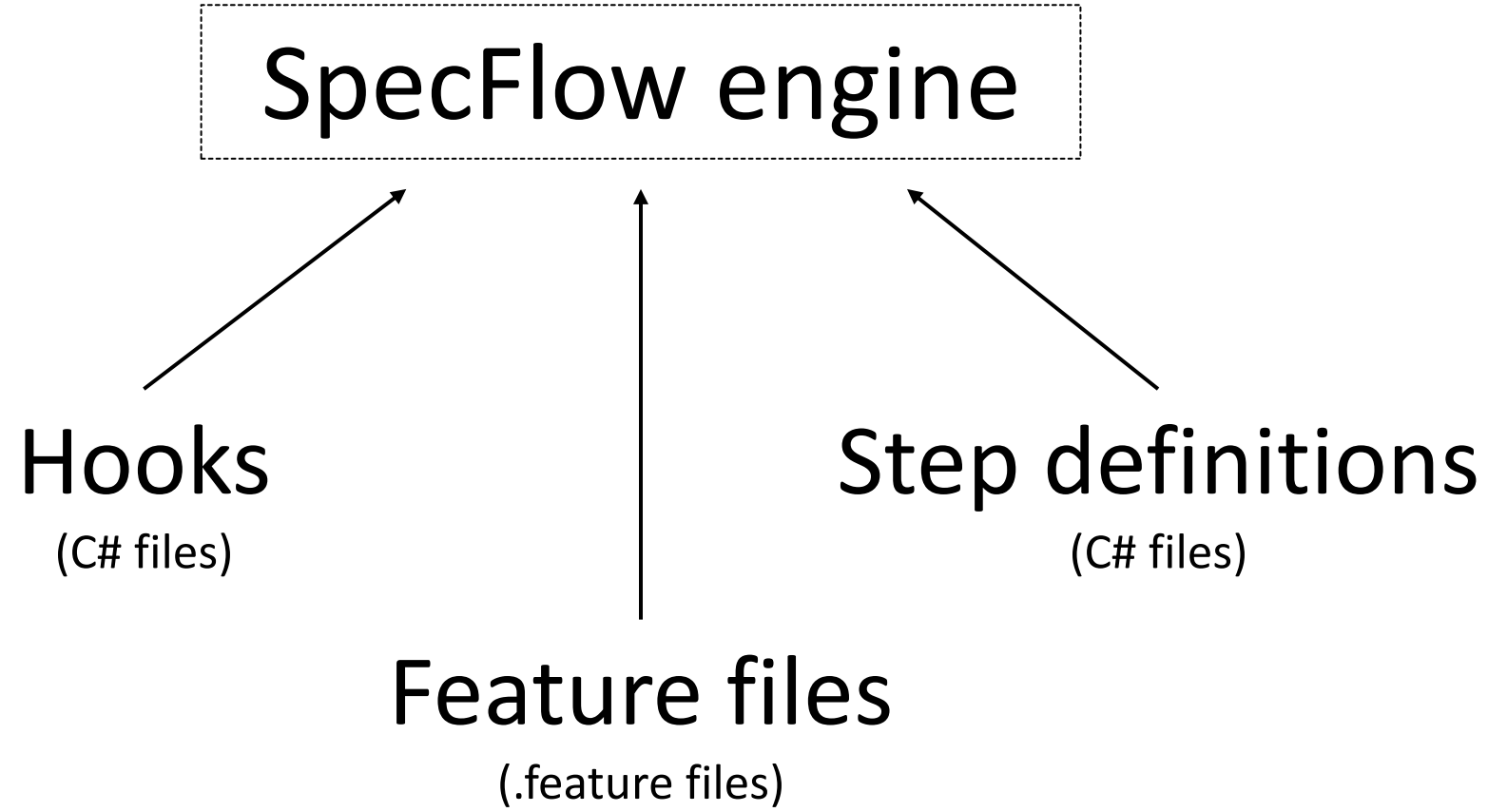


How to get started?

specflow

- Visual Studio
  - Getting stared guide
- Rider
  - Getting started guide
- VSCode
  - Official extension does not exist yet

Main components



Feature files

specflow

- Language of the business
- Follows Gherkin format
  - Set of special keywords
  - Supports many natural languages

```
Feature: Calculator
        In order to avoid silly mistakes
        As a math idiot
        I want to be told the sum of two numbers

Scenario: Add two numbers
        Given I have entered 50 into the calculator
        And I have also entered 70 into the calculator
        When I press add
        Then the result should be 120 on the screen
```

Gherkin keywords

specflow

- *Feature*
  - High-level description of SW feature
  - Groups related scenarios
  - **Always must be the first word in .feature file!**
  - Feature name should be unique
- *Rule*
  - Represents one business rule that should be implemented
  - Used to group together several scenarios that belong to this business rule

```
Feature: Highlander

  Rule: There can be only One

    Scenario: Only One -- More than one alive
      Given there are 3 ninjas
      And there are more than one ninja alive
      When 2 ninjas meet, they will fight
      Then one ninja dies (but not me)
      And there is one ninja less alive

    Scenario: Only One -- One alive
      Given there is only 1 ninja alive
      Then he (or she) will live forever ;-)

  Rule: There can be Two (in some cases)

    Scenario: Two -- Dead and Reborn as Phoenix
      ...
```

Gherkin keywords

specflow

- *Scenario*
  - Concrete example that illustrates a business rule
  - Consists of list of steps
- *Given*
  - Used to describe the initial context of the system
  - The purpose is to put system in a known state before user interaction
  - You can have more than one *Given* in one *Scenario*

```
Feature: Highlander

  Rule: There can be only One

    Scenario: Only One -- More than one alive
      Given there are 3 ninjas
      And there are more than one ninja alive
      When 2 ninjas meet, they will fight
      Then one ninja dies (but not me)
      And there is one ninja less alive

    Scenario: Only One -- One alive
      Given there is only 1 ninja alive
      Then he (or she) will live forever ;-)

  Rule: There can be Two (in some cases)

    Scenario: Two -- Dead and Reborn as Phoenix
      ...
```

Gherkin keywords

specflow

- *When*
  - Used to describe an event, or an action
  - **Single *When* per scenario!**
- *Then*
  - Used to describe an expected outcome or result
  - Place to perform assertion checks
- *And, But*
  - Replacement of consecutive *When*, *Then* and *Given* keywords

```
Feature: Highlander

  Rule: There can be only One

    Scenario: Only One -- More than one alive
      Given there are 3 ninjas
      And there are more than one ninja alive
      When 2 ninjas meet, they will fight
      Then one ninja dies (but not me)
      And there is one ninja less alive

    Scenario: Only One -- One alive
      Given there is only 1 ninja alive
      Then he (or she) will live forever ;-)

  Rule: There can be Two (in some cases)

    Scenario: Two -- Dead and Reborn as Phoenix
      ...
```

Step definitions

- Connection between feature files and app interface
- **Step definitions rules:**
  - Class must be public, marked with *[Binding]* attribute
  - Class can contain either static or instance methods
  - Methods must be public
  - Methods can not have *out* or *ref* parameters
  - Methods can not have a return type or have *Task* as return type
- Step definitions are global for the entire SpecFlow project
  - You can create scoped step definitions using *[Scope]* attribute

Bindings example

specflow

```
1  Feature: Example of bindings
2
3  Scenario: Example of bindings
4      Given There are 3 apples
5      When I eat 1 apple
6      Then There are 2 apples left
7
```

```
[Binding]
0 references
public class BindingsSteps
{
    [Given("There are 3 apples")]
    0 references
    public void PrepareApples() { /* code */ }

    [When("I eat 1 apple")]
    0 references
    public void EatApple() { /* code */ }

    [Then("There are 2 apples left")]
    0 references
    public void CheckHowMuchIsLeft() { /* code */ }

}
```

Hooks

specflow

- Additional automation logic at specific times
- Class must be public, marked with *[Binding]* attribute
- Hooks are global for the entire SpecFlow project
  - You can create scoped hooks using *[Scope]* attribute
- Supported hooks:
  - *[BeforeTestRun]/[AfterTestRun]*
  - *[BeforeFeature]/[AfterFeature]*
  - *[BeforeScenario]/[AfterScenario]*
  - *[BeforeScenarioBlock]/[AfterScenarioBlock]*
  - *[BeforeStep]/[AfterStep]*

Hooks example

specflow

```
[Binding]
0 references
public class Hooks
{
    [BeforeTestRun]
    0 references
    public void BeforeTestRunLogic() { /* code */ }

    [BeforeFeature]
    0 references
    public void BeforeFeatureLogic() { /* code */ }

    [BeforeScenario]
    0 references
    public void BeforeScenarioLogic() { /* code */ }
}
```

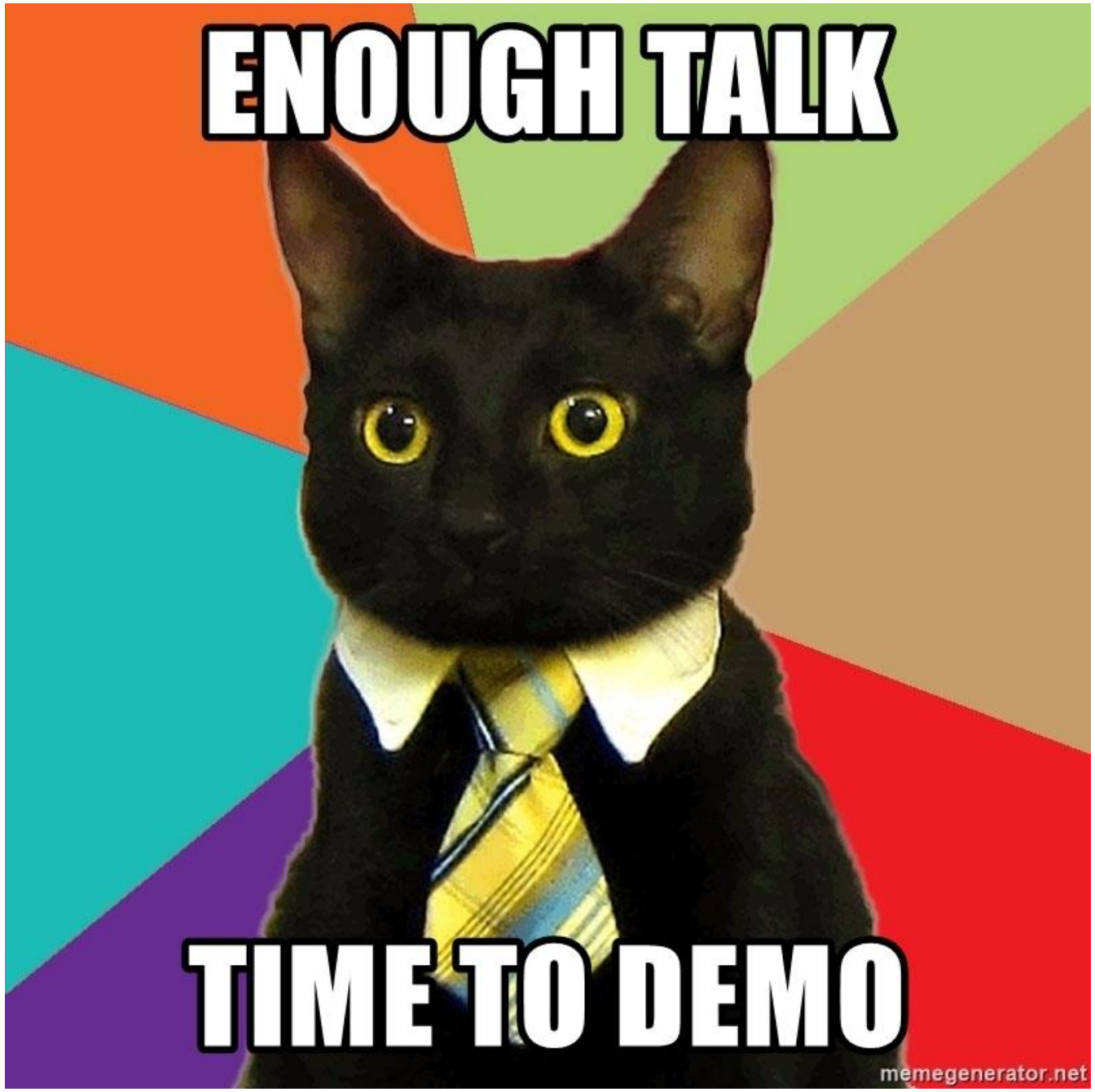
Dependency/Context injection

- BoDi DI framework
  - Dependency injection possible only into classes with *[Binding]* attribute
  - Slightly different from Microsoft DI framework
  - Allows integration with MS DI, Autofac etc.
- ScenarioContext
  - New instance for each scenario
  - Can be accessed in Before/AfterScenario and Before/AfterStep hooks
- FeatureContext
  - New instance for each feature
  - Can be accessed in Before/AfterScenario, Before/AfterFeature and Before/AfterStep hooks

Execution model

specflow

- Parallelization of the execution of features depends on testing framework:
  - NUnit and MSTest does not run features (tests) in parallel by default
  - xUnit runs all SpecFlow features (tests) in parallel
- Tests are running in multiple threads within the same process
- All scenarios in a feature are executed on the same thread
- Scenarios and their hooks are isolated in the different threads (each scenario has its own thread)


ENOUGH TALK
TIME TO DEMO
memegenerator.net