specsolutions
given.when.then.

Tree in the Forest

Managing Details in BDD Scenarios

HUSTEF 2023
5th October, 2023

HUSTEF
HUNGARIAN SOFTWARE TESTING FORUM

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

Gáspár Nagy

coach, trainer and bdd addict
creator of SpecFlow

gaspar@specsolutions.eu
https://specsolutions.eu
@gasparnagy
http://bddbooks.com

specsolutions HUSTEF

Copyright © Gaspar Nagy

Today…

- Super brief BDD intro
- Scenarios as Tests
- Scenarios as Specification
- Types of Details
- Wrap-up, Q&A

Copyright © Gaspar Nagy

BDD in 3 minutes...

Copyright © Gaspar Nagy

BDD scenario: bridge between requirements and the solution

```
Scenario: User votes up a question
Given a question asked with 2 votes
And the user is authenticated
When the user votes up the question
Then the votes should be changed to 3
```

make requirements testable

make tests understandable

specsolutions HUSTEF

Copyright © Gaspar Nagy

The bridge and the lightbulb by Unknown Author is licensed under CC BY-SA

Behaviour Driven Development is about

understanding, documenting & validating

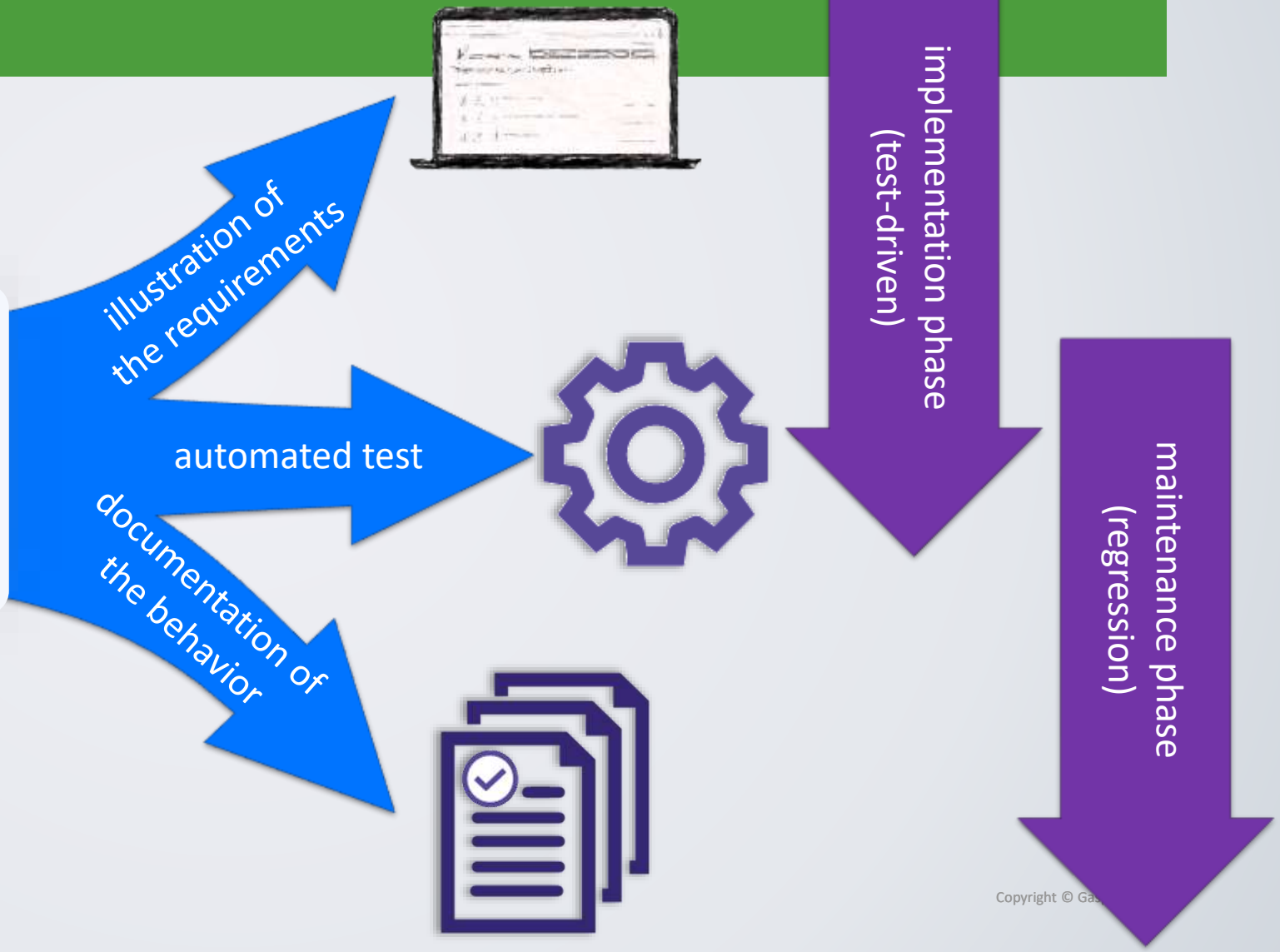
business requirements

through illustrative scenarios

Copyright © Gaspar Nagy

Purposes of scenarios

```
Scenario: The one where…
Given some context
When perform an action
Then the outcome happens
```



BDD is not (only) testing!

Copyright © Ga

Scenarios as Tests

Copyright © Gaspar Nagy

All *details* that are required for exercising the application has to be available *during execution* latest

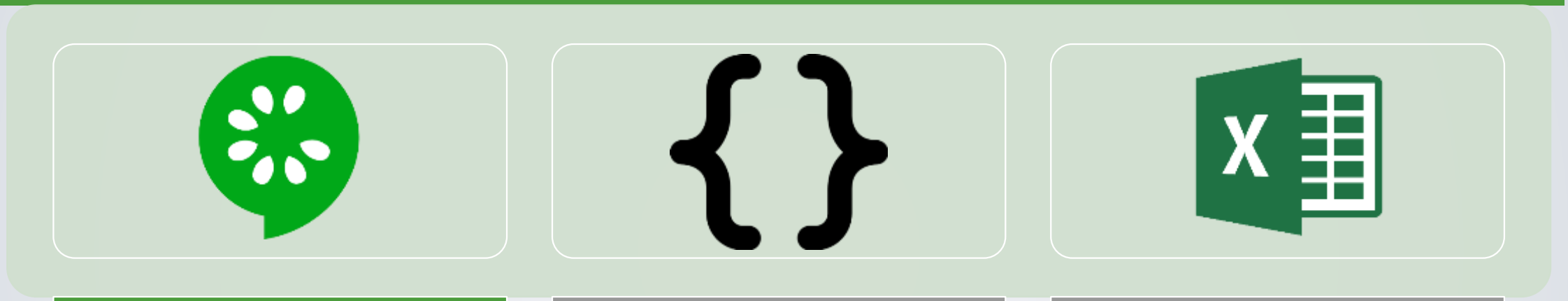
Copyright © Gaspar Nagy

This brings up a couple of interesting questions…

- What are those *details*? What kind of details we should consider?
- How do we agree on those details?
- Where do we document (store/code) those details?

Copyright © Gaspar Nagy

Where to document the details?



In the text of the **scenario** in the feature file

In the automation **code**

In **external** sources (Excel, existing data, etc.)

managed by code

Copyright © Gaspar Nagy

The details are not removed from the scenario, but **pushed down** to the code

Copyright © Gaspar Nagy

Scenario

VS

Code

Details in Gherkin

Pro

- Visible
- Easier test result analysis (debugging)

Con

- Makes scenario hard to read
- Maintainability problems

Details coded or processed by code

Pro

- Easy to reuse
- Different abstractions can be defined
- Maintainable
- Visible during execution (dynamic visibility)

Con

- Harder to see details in code (static visibility)
- Code is not a shared asset

Copyright © Gaspar Nagy

It is easier to manage details in code,
but we need to work extra for visibility

Copyright © Gaspar Nagy

Scenarios as Specification

Copyright © Gaspar Nagy

How to make scenarios to good specification

- Easy to read, review & discuss
  - Not too long (brief)
  - Free of “noise”
  - Business language (ubiquitous language)
- Should only change when the requirement changes
  - No technical/solution details
  - Focus on the what (intention) and not the how (steps to achieve)
- Concrete, so that the validity can be checked
  - Contain essential (relevant) details
  - Contain real/concrete data
  - Should focus on a single business rule

Copyright © Gaspar Nagy

6 BRIEF principles of good scenarios

- **B**usiness language — Business terminology aids cross-discipline collaboration
- **R**eal data — Using actual data helps reveal assumptions and edge cases
- **I**ntention revealing — Describe the desired outcomes, rather than the mechanism of how they are achieved
- **E**ssential — Omit any information that doesn't directly illustrate behaviour
- **F**ocused — Each scenario should only illustrate a single rule

- Brief — Shorter scenarios are easier to read, understand and maintain

Copyright © Gaspar Nagy

What details are essential?

There are no fixed guidelines to decide on this… and the answer might be context specific…

- The details that influence the outcome (=Then steps)
- The details that are relevant for the “rule” (=AC, business rule, requirement) the scenario illustrates (“focuses on”, purpose)

Copyright © Gaspar Nagy

Deciding on essential details in scenarios without clear outcome and purpose is not possible

You need to fix those first

Copyright © Gaspar Nagy

Types of Details

Copyright © Gaspar Nagy

Detail types

Scenario Related

1. Entity property
2. Entity existence
3. Hierarchical data
4. Technical details
5. Contextual details (scenario execution context)

Implicit Context Relate

6. Baseline
7. Workflow steps
8. System status

I show 4 of them today, but the slides contain examples for the others too

Note: Not necessary a complete list and there are overlaps

Copyright © Gaspar Nagy

1. Entity property

```
Given the customer has an order
  | status  | payment method | issue date | due date  |
  | created | bank_transfer  | 7/25/2023  | 8/24/2023 |
```

used defaults

```
Given the customer has an order
```

defaults can be overridden

```
Given the customer has a dispatched order
```

```
Given the customer has an order
  | status     |
  | dispatched |
```

- Many properties of the entity might not be relevant for the scenario
- These properties change and extended independently of the scenario
- Push down: maintain a default (test) values for the properties of the entities in code

Copyright © Gaspar Nagy

2. Entity existence

```
Given there is a customer
And the customer has an address
  | street                | city      |
  | 18 Holyrood Park Road | Edinburgh |
```

```
Given there is a customer
```

pre-populate with default address

required prerequisites

```
Given there is a customer
And there is an order of the customer
```

```
Given there is an order
```

ensure customer prerequisite

- The existence of some entities might be required for execution, but not relevant for the outcome (e.g. address record of a customer; customer for an order)
- These requirements might change independently of the scenario
- Push down:
  - populate the records with defaults automatically
  - ensure that all prerequisite is created

Copyright © Gaspar Nagy

3. Hierarchical data

```
Given there is a customer "AA“
And the customer "AA“ has an order "0293445“
And the order "0293445" has an item
  | item id | product   | quantity |
  | 268     | Flipchart | 13       |
And the order item "268“ has a comment “…“
```

relevant details only

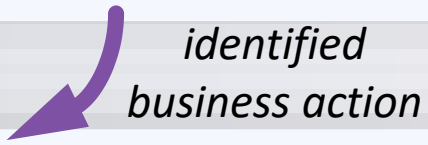
```
Given the is an order with
  | product   | quantity | comment |
  | Flipchart | 13       | ...     |
```

- Some tests require a hierarchical set of data
- Complex data hierarchies are hard to understand, so should be simplified
- Push down:
  - ensure that prerequisite is created
  - build up hierarchy as a story by relevant steps
  - use custom format to express (e.g. yaml)
  - use internal IDs

Copyright © Gaspar Nagy

4. Technical details

```
When the user clicks on button
        "//*[@id="orders_0293445"]/td[3]/a"
```




```
When the user attempts to start changing in
the order
```

- Technical details and steps might be needed to execute a particular business action (e.g. enter the name to a textbox with ID “tb1234”)
- These details are bound to the implementation, not the specification
- Documenting this details in scenarios requires intensive maintenance
- Push down:
  - Identify & express business needs or actions
  - Translate these to the technical details in code (reusable!)

Copyright © Gaspar Nagy

5. Contextual details (scenario execution context)

```
Given there is a customer "AA"
And the customer "AA" has an order "0293445"
When the customer "AA" changes
                        the order "0293445"
```

```
Given the customer has an order
When they change the order
```

can use more explicit form as well

```
Given the customer has an order
When the customer changes the order
```

- The “story” that the scenario explains might need to introduce details that are referred in later steps – they are part of the scenario execution context
- Duplicating these details makes the scenario verbose and hard to understand
- Push down:
  - Store these contextual details in scenario-lifetime storage provided by the BDD framework (World, ScenarioContext)
  - Resolve the references to these details from the storage

Copyright © Gaspar Nagy

6. Baseline

```
Given there are the following products
  | name          | unit price |
  | Flipchart     | 250        |
  | Sticky Notes  | 18         |
And the is an order with
  | product    | quantity |
  | Flipchart  | 13       |
```

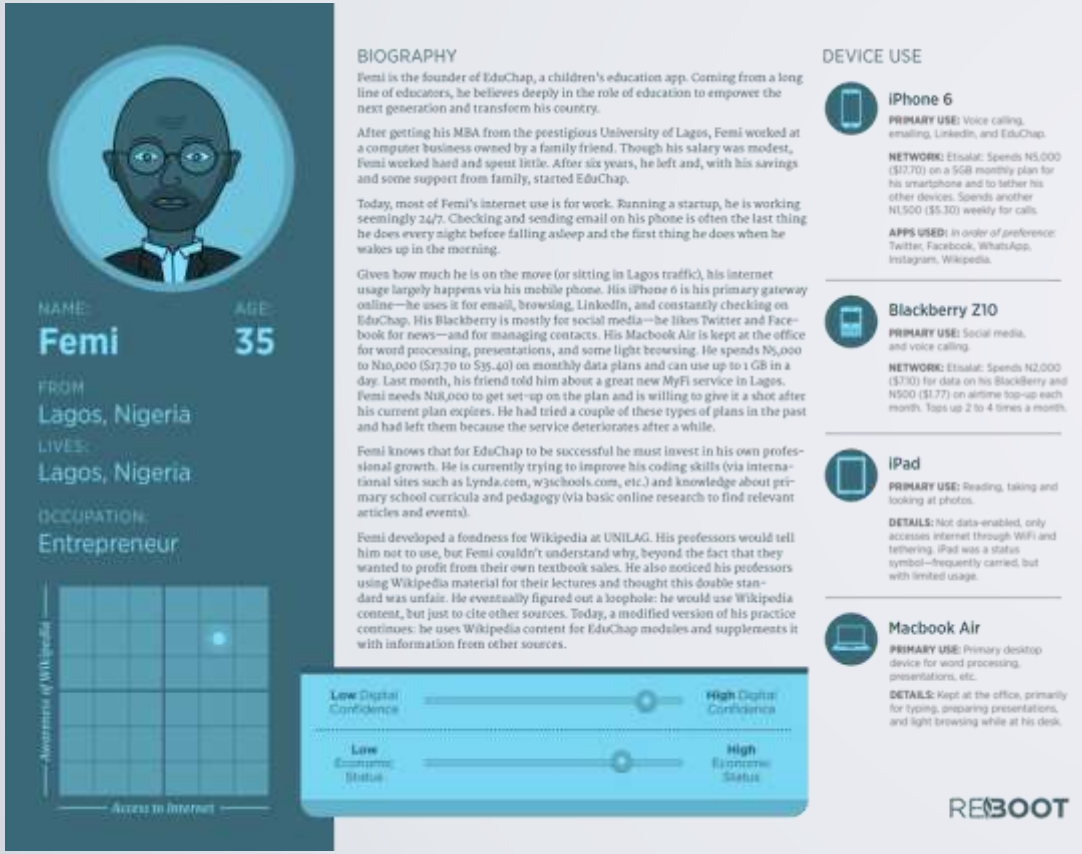
Flipchart and Sticky Notes are always there

```
Given the is an order with
  | product    | quantity |
  | Flipchart  | 13       |
```

- The scenarios of a product might need “usual” data for expressing the examples
- Many team has an implicit agreement to use the same data over and over, so it might became noise
- Push down:
  - Make a team agreement of “default” test data and make sure those can be used without explicitly listing them
  - Special scenarios can still override
  - There might be multiple baseline sets

Copyright © Gaspar Nagy

Personas in “Baseline” details



Picture source: Wikipedia

- A persona is a fictional character created to represent a user type that might use a […] product in a similar way. (Wikipedia)
  - Well known to the team
  - Helps discussions as it saves time of explaining the context
- Similarly you might have fictional, well-known data that we use in discussions
  - Jenny Martin refers to these as “Data Personas” (https://jennyjmar.com/2016/04/15/data-personas/)
- Personas and Data Personas are also useful for specification & testing!

spacesolutions HUSTEF

Copyright © Gaspar Nagy

7. Workflow steps

```
Given the customer has logged in
And the customer dismissed the newsletter subscription
And the customer started to create an order
And the customer added the line to the order
  | product   | quantity |
  | Flipchart | 13       |
And the customer has placed the order
```

removed non-relevant workflow steps

```
Given the customer has placed an order with
  | product   | quantity |
  | Flipchart | 13       |
```

- The different user or workflow steps might also represent details that are not relevant
- Keeping these details in the scenario causes maintainability issues once the workflow changes
- It is better to state the intentions and the relevant steps only
- Push down:
  - Express intention revealing steps and ensure the state by repaying the necessary steps or by applying a shortcut in the automation code

Copyright © Gaspar Nagy

8. System status

```
Given there is a customer
And the customer is not blocked
```

removed non-relevant status declaration

```
Given there is a customer
```

- The details might be shown as declaration of the system or entity statis
- Declaring “normal” status makes the scenarios harder to understand
- Push down:
  - Remove unnecessary status declarations from the scenario
  - Optionally add precondition checks to different actions to make sure that the system is in the expected state

Copyright © Gaspar Nagy

Wrap-up

Copyright © Gaspar Nagy

We covered…

- That scenarios need details for execution, but you don’t necessarily need to store those in the scenario itself, but push down to automation code
- That in order to use scenarios as specification, we need to make them focused and express only essential details in the steps
- That there are different detail types, watch for them and apply the mentioned “push-down” strategies if required

Copyright © Gaspar Nagy

Be aware of the details

Push them down

Make a team agreement on implicit details

Copyright © Gaspar Nagy

specsolutions
given.when.then.

Thank you!

HUSTEF
HUNGARIAN SOFTWARE TESTING FORUM

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