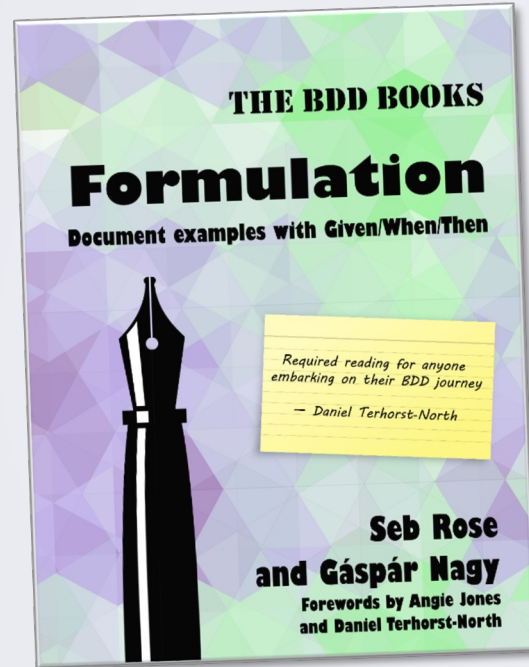
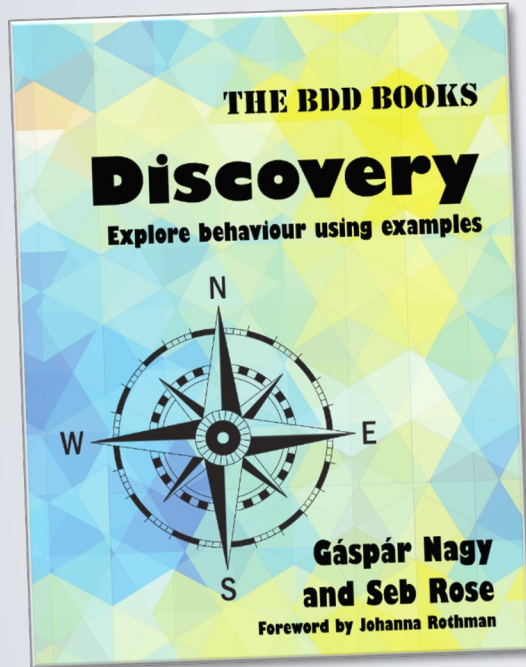


expo IQA 25

MADRID
May 20th,
21st & 22nd
2025

**Ramp Up Your Testing Solution:
Test Automation Patterns**

expoqa.eu



Find them on Amazon & Leanpub through <https://bddbooks.com>!

Gáspár Nagy

coach, trainer and bdd addict
creator of SpecFlow & Reqnroll

gaspar@specsolutions.eu

www.specsolutions.eu

linkedin.com/in/gasparnagy

Recognize this?



Today

- About design patterns
- Challenges of test automation today
- Benefits of using design patterns for test automation solutions
- Characteristics of test automation design patterns
- Documenting test automation design patterns

Design Patterns

recognized construct that works

Why design patterns?

Better
Faster
Easier collaboration

Challenges of Test Automation

Why to build a good quality test automation



Tests automation as “first class citizen”

Coding standards
Reviews
Tests

...

Test automation is difficult

Requires “smart” solutions

Needs time and efforts

Involves expertise, research & collaboration

This is a STUB

How could patterns
help?

Tests are mostly composed of “usual tasks”

Reset database to a baseline
Perform an action and capture result
Query state to verify expectations

“Design patterns” can be used & reused!

- Patterns for “Reset database to a baseline”
 1. Restore baseline database backup
 2. Create an empty database and insert base data records
 3. Use in-memory database
 4. Use file-based database and copy baseline file
 5. Wrap test to a DB transaction & cancel
 6. Truncate (empty) tables used by the tests
 7. Track data changes and undo
 8. Detect read-only tests and skip reset after them

RAMP up your testing solution

Reusability
Abstraction
Maintainability
Performance

Test Automation Design Patterns

The UI analogy

Both

- represent an external interface of the application
- are active (click on a button <> perform action on SUT)
- are event-based, async and responsive (display data as arrives <> verify result as becomes available)
- have simple structure: sequence, containment (e.g. no recursion)
- contain repeating needs (controls <> usual test tasks)
- include a mix of concerns (view/controller <> test-description/actions)

Hexagonal Architecture

Onion Architecture

Generic & project-specific patterns

The high level of reusability encourages teams to think about discovering project-specific patterns.

- Easier to develop (more specific context)
- Easier to document (sometimes the example in the code is enough)

Some test automation patterns are usable in broader context

- They can become part of the tool-belt of the test automation experts

test & dev collaboration

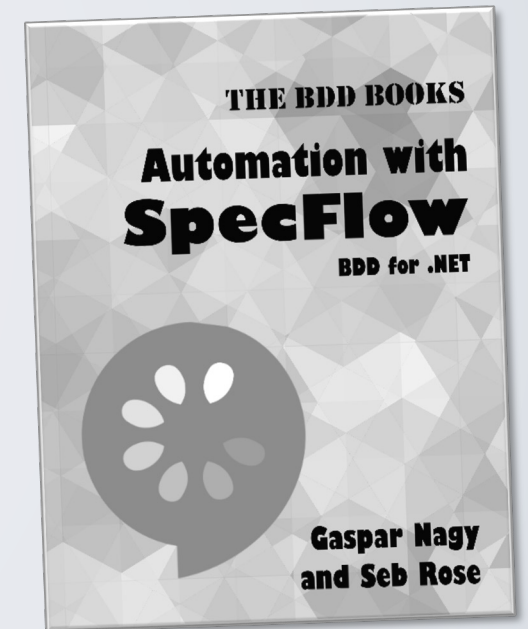
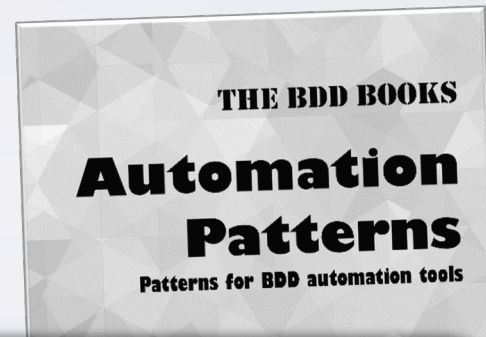
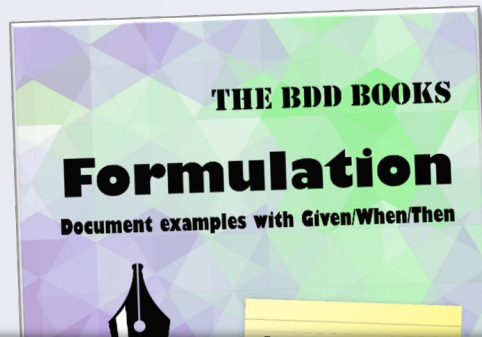
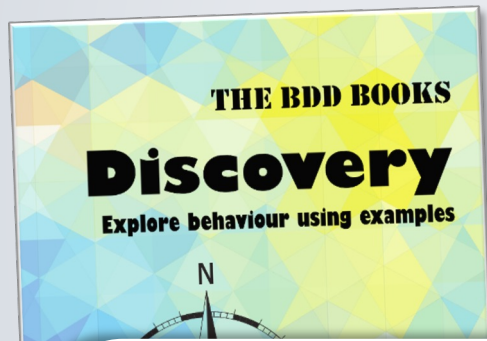


Patterns

This is shared
knowledge

Documenting Test Automation Design Patterns

The book project

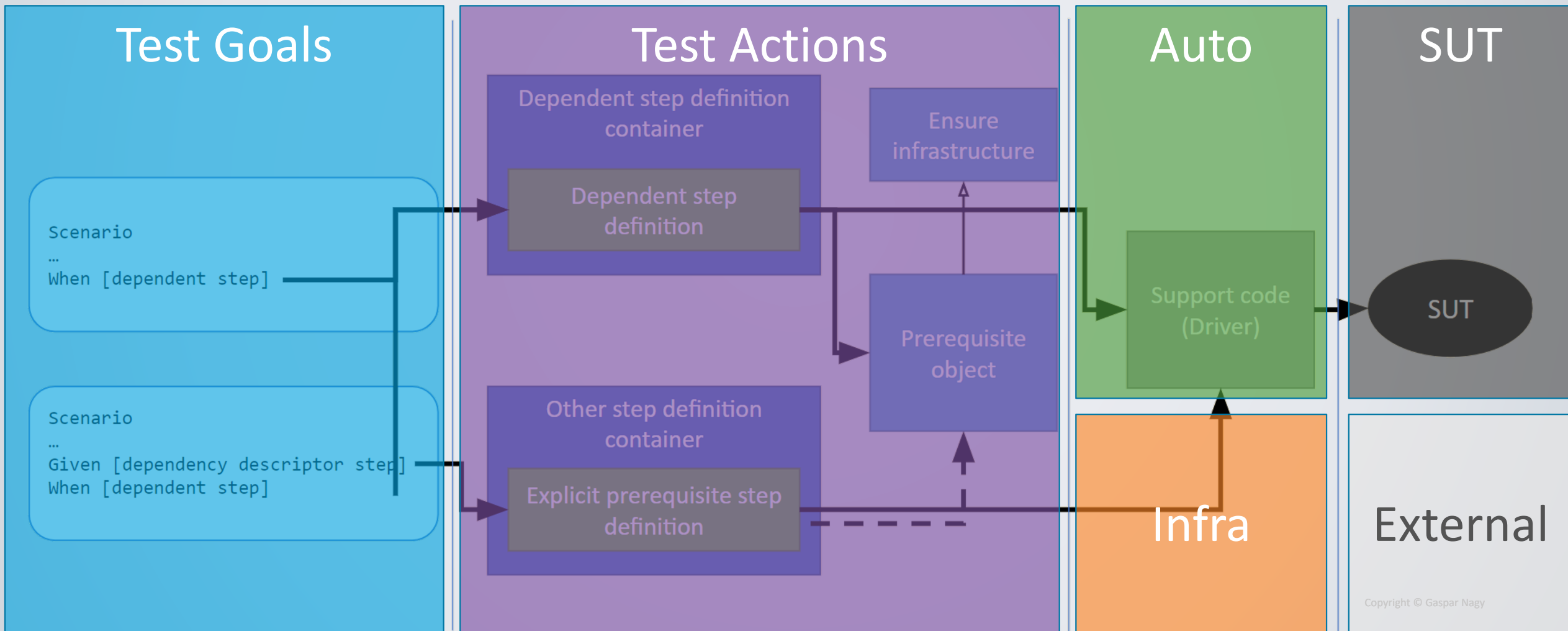


MANNING PUBLICATIONS

How to document the patterns?

- Name & Intent – short summary
- Motivation – situation example
- Applicability – list of useful contexts
- Structure – diagram
- Participants & Collaborations – abstract description of the pattern
- Consequences – benefits, trade-offs
- Implementation & Sample – implementation notes & concrete example

The Layering Problem



Samples

Sample Patterns: Ensure

Intent

Maintainable management of scenario context, whether expressed explicitly or implicitly.

Motivation

The formulated BDD scenarios often contain implicit context: expectations about the context that we don't want to explicitly express as Given steps, because they are obvious from the scenario (See Formulation 4.11). This is a commonly used technique and every scenario has several implicit context expectations that we never write out (e.g. the system is in an operational state).

In the WIMP application there are many requirements (and therefore many scenarios) that describe what happens with a particular order. For example when the user chooses to collect their order from the restaurant, they need to confirm their contact details. This is described with the following scenario.

Rule: Any visitor to the website can place a customer-collection order

```
Scenario: Authenticated customer chooses to collect order
  Given the customer is authenticated
  When they choose to collect their order (1)
  Then they should be asked to confirm contact details
```

In this scenario at (1) we talk about the customer's order, but we never explicitly mention that the customer has placed an order or what pizzas they have ordered. It is obvious from the scenario that the customer has placed an order somehow.

In contrast to that there are some cases when some details about the order are important. The following example describes the expectation that for customer-collection orders we need to print a collection receipt that contains the number of boxes to be collected, so that it should be easy to verify that all items have been handed over to the customer on collection.

Rule: A collection receipt has to be printed for customer-collection orders

```
Scenario: The number of pizzas to be handed over is indicated on the receipt
  Given the customer has placed an order for 3 pizzas (2)
  When they choose to collect their order (3)
  Then a collection receipt should be printed with
  | boxes to be collected |
  | 3                     |
```

The scenario above contains the same When step as the previous one (3), but now we have an additional Given step (2) that describes the details of the order that has been placed.

As it might be clear from these examples, the When step (1) and (3) is reused in two different situations: when there were no previous steps about the order and when the order specifics have been described in a previous step (2).

How can we automate the When step so that it is reusable for both situations? How can we make sure that the details about the order placement are not disturbing the automation logic we would provide to simulate that the customer chooses to collect their order?

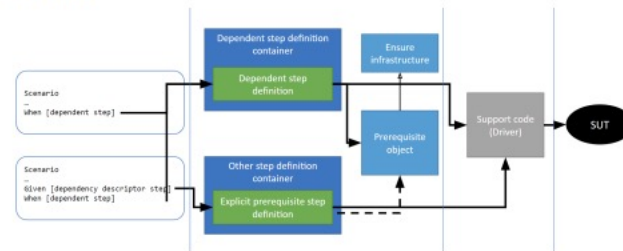
The placement of an order is a prerequisite for choosing customer-collection for the order. The Ensure pattern provides a solution to ensure that the prerequisite has been fulfilled by tracking or checking if the order has been placed already and automatically placing some default order if not.

Applicability

Use the Ensure pattern when

- A step has a prerequisite (prerequisite step) that we must make sure has happened to be able to execute the step correctly
- A step has a prerequisite that is sometimes explicitly expressed in the scenario (explicit prerequisite step), but sometimes it is considered to be implicit

Structure



Participants

- Dependent step definition (=> WhenTheyChooseToCollectTheirOrder method)
 - The step definition that requires the prerequisite to be fulfilled
- Explicit prerequisite step definition (=> GivenTheCustomerHasPlacedAnOrderForPizzas method)
 - The step definition(s) that perform the actions to fulfill an explicit prerequisite

- Prerequisite object (=> OrderPlacementPrerequisite class)
 - This is the class that implements the ensure logic
 - It contains the necessary *fulfill functionality* to satisfy the prerequisite
 - It contains the necessary *tracking or querying functionality* to determine whether the prerequisite has already been fulfilled
 - It exposes *ensure functionality* that the dependent step can call
- Ensure infrastructure (=> PrerequisiteBase class, TrackedPrerequisiteBase class)
 - It contains the infrastructure code to perform the ensure logic
 - It may also include logging to improve the diagnosis of the prerequisite management

Collaborations

- The *dependent step definition* obtains the *prerequisite object* using the state sharing mechanism of your BDD automation tool, e.g. World object or DI (see TODO:ref).
- The *dependent step definition* invokes the *ensure functionality* of the prerequisite.
- The *ensure functionality* (in the ensure infrastructure) determines whether the prerequisite has been fulfilled using the *tracking or querying functionality*. If it turns out that the prerequisite was not fulfilled yet, it invokes the *fulfill functionality* of the *prerequisite object*.
- If the *prerequisite object* is tracking fulfillment, any *explicit prerequisite step definition* will need to signal that the prerequisite has been fulfilled.

Consequences

Here are key consequences of the Ensure pattern:

1. **Allows briefer scenarios, through omission of explicit prerequisites.** The scenarios without the obvious context steps will be easier to understand and maintain. This may also eliminate the need for using background steps that make the scenarios less readable.
2. **Permits more flexible semantics, by making expression of prerequisites optional.** It is easier to use implicit contexts and therefore people will be better encouraged to not state obvious context statements as additional Given steps.
3. **Avoids code duplication.** Instead of duplicating the statements to fulfill the prerequisite to each step definition where the prerequisite is needed, they can be implemented in a single location.
4. **Makes execution faster.** The automation code does not need to perform the steps that are required to fulfill the prerequisite multiple times.
5. **Can make prerequisite-related problems better diagnosable.** When including logging statements into the ensure infrastructure classes, all prerequisite related code will provide the necessary log information to make the diagnosis of any prerequisite-related problems easier.
6. **Complex prerequisite graphs may be harder to track.** When there are many prerequisites of this kind in the automation solution and especially when there are even

Wrap-up

Wrap-up

- Design patterns are powerful tools
- Only good tests make sense
- Tests should really be first class citizens
- Tests composed of “usual tasks”, so design patterns are super-powerful
- You can discover project-specific patterns – this also helps dev-test collaboration
- But many patterns are even more broadly usable
- Think of “UI analogy” and test layers



Watch for recurring problems.
Take the time to develop a pattern for them.
(let dev and test collaborate)
Apply pattern.
Repeat.

expo QA 25

MADRID
May 20th,
21st & 22nd
2025

Thank you for attending

Gáspár Nagy

coach • trainer • bdd addict • creator of specflow
“The BDD Books” series • <https://bddbooks.com>
[linkedin.com/in/gasparnagy](https://www.linkedin.com/in/gasparnagy) • gaspar@specsolutions.eu

expoqa.eu