Use Cases and BDD/ATDD Scenarios

Ken Pugh (ken@kenpugh.com)

Ivar Jacobson And Alistair Cockburn have a great article on "Use Cases are Essential". You can read it at https://dl.acm.org/doi/pdf/10.1145/3631182. Its summary says, "Use cases provide a proven method to capture and explain the requirements of a system in a concise and easily understood format". I have incorporated use cases in my ATDD/BDD workshops for many years. In this article, I'll show one transformation of the use case they used in their paper into the detailed ATDD/BDD scenarios.

Here's one use case Ivar and Alistair presented in the article:

```
Place an order.
Primary actor: Clerk
Main scenario:
1. Clerk identifies customer, item, and quantity.
2. System accepts and queues the order.
Alternatives:
1a. Low credit & Customer is 'Preferred': System gives them credit
anyway.
1b. Low credit & not 'Preferred' Customer: Clerk accepts only
prepayment.
2a. Low on stock: Customer accepts raincheck: Clerk reduces order to
available stock level.
```

This readable format allows collaboration between customers, business analysts, developers, and testers. The use case gives the overall flow of an activity along with key conditions. It has enough abstraction that it provides a relatively static description of the activity without getting into more changeable details.

A use case forms the basis for developing ATDD/BDD scenarios. Many use cases have pre-conditions (things that must be true before the use case executes) and post-conditions (things that must be true after the use case executes). A scenario has a Given (pre-conditions), a When (main course or alternative or exception), and a Then (post-conditions). A scenario represents one flow through the use case, with the data needed for that flow.

When creating scenarios, one can discover domain terms that represent common concepts. One may find business rules when creating the use case or determine them while creating the scenarios. The scenarios include examples that use these domain terms and business rules.

As with use cases, the customers/business analysts, developers, and testers can collaboratively create more detailed scenarios. During that collaboration, additional flows may be discovered.

Here is one possible set of scenarios that might evolve during that collaboration. The first two scenarios are domain terms for Customer Class and Credit. Having terms defined in this format can help with capturing the business rules for other conditions.

```
Scenario: Domain Term Customer Class
* Customer classes
| Preferred |
| Normal    |

Scenario: Domain Term Credit
* Credit values are
| Low |
| OK  |
```

Perhaps these terms have already been used in other use cases, so the corresponding scenarios can be re-used. If additional values for each term are present and there were different behaviors for those values, then there would be additional alternatives. Using those domain terms, the use case might expand into the following scenarios. These scenarios use a tabular form, rather than sentence form that one may see in examples in other articles. The main course of the use case might be:

```
Scenario: Main Scenario
Given customer is
| Customer  | Class      | Credit  |
| George    | Preferred  | OK      |
And item is
| Name    | Quantity In Stock  |
| Widget  | 5                  |
When order is placed
| Customer  | Item    | Quantity  |
| George    | Widget  | 1         |
Then order result is
| Order queued for 1 Widget |
```

The scenario has example values for each of the data items represented by the column headers. A scenario should make the essential data items transparent. Nonessential fields, such as George's address, would not be included, unless the scenario concerned different behavior for different addresses.

Here's the scenario for one alternative. The only variation is the Credit is Low.

```
Scenario: Low Credit for Preferred Customer
Given customer is
| Customer  | Class      | Credit  |
| George    | Preferred  | Low     |
And item is
| Name    | Quantity In Stock  |
| Widget  | 5                  |
When order is placed
| Customer  | Item    | Quantity  |
| George    | Widget  | 1         |
Then order result is
| Order queued for 1 Widget |
```

Here's the scenario for another alternative. The variation in behavior is due to the customer's class being normal with low credit.

```
Scenario: Low Credit for Unpreferred Customer
Given customer is
| Customer  | Class    | Credit  |
| Sam       | Normal   | Low     |
And item is
| Name    | Quantity In Stock  |
| Widget  | 5                  |
When order is placed
| Customer  | Item    | Quantity  |
| Sam       | Widget  | 1         |
Then order result is
| Prepayment only |
```

In developing the above scenarios, the writer might have copied and pasted the original scenario and then changed a few values. The copy and paste might be a sign that there is a business rule that is being stated through flow scenarios (ones with Given/When/Then), rather than a business rule scenario. The business rule could look like:

```
Scenario: Business Rule Order Result Based on Customer
* Result of order being placed depends on Class and Credit
| Customer Class  | Credit  | Result             |
| Preferred       | OK      | Order queued       |
| Preferred       | Low     | Order queued       |
| Normal          | OK      | Order queued       |
| Normal          | Low     | Prepayment only    |
```

The flow scenarios can be reduced to the main scenario and a scenario that shows the business rule has been incorporated into the flow. In some contexts, business rules are a significant portion of the requirements. So, separating them into their own scenarios can be more readable.

The remaining alternative in the use case could have the following as its corresponding scenario:

```
Scenario: Low on Stock
Given customer is
| Customer  | Class      | Credit  |
| George    | Preferred  | OK      |
And item is
| Name    | Quantity In Stock  |
| Widget  | 1                  |
When order is placed
| Customer  | Item    | Quantity  |
| George    | Widget  | 2         |
Then order result is
| Order queued for 1 Widget |
| Raincheck for 1 Widget    |
```

Creating these scenarios might appear to be a bit of work. But the process can evolve rapidly with some training and/or experience. The result is worth it. The scenarios represent shared agreement on the desired behavior. They also represent the tests for that behavior, as the tests check that the Then part is achieved by the implementation. Some organizations have found zero defects in the behavior covered by the scenarios.

The scenarios form the basis for automated testing. Using the appropriate framework (e.g., Cucumber or SpecFlow), the scenarios as written above can execute production code. If a use case as a work item is too big to implement in an iteration, it can be broken into work items that each contain a scenario. The definition of done on each work item is demonstrating the desired behavior of the scenario. The scenarios can also form examples for training.

So, you get a four-fer use out of scenarios (requirement, test, documentation, scheduling). A little work can go a long way.