Requirement Driven Development

Ken Pugh (ken@kenpugh.com)

Introduction

Software applications should satisfy the requirements. To put it in reverse, requirements drive the development. Every requirement should have a test for it that checks that the requirement is implemented properly. Any code that does not satisfy a requirement (functional or non-functional) is not needed. [1]

Specification

There are many ways to specify requirements such as a formal requirement document, a set of use cases[2], or a set of stories. Many agile processes focus on creating the smallest set of requirements which can deliver some business value. They are typically defined from a user's point of view. One way to describe these requirements is through scenarios, using the Gherkin syntax. The syntax allows for a consistent way of expressing requirements. Each scenario typically represents a flow through a portion of the application . The scenario focuses on the how the application should behave for that requirement.

An example of a scenario written in Gherkin is:

```
Scenario: Transfer money to another account
Given customers have accounts
| Customer  | Balance  |
| Sam       | $100     |
| Bill      | $50      |
When customer does a transfer
| Customer  | Recipient  | Amount  |
| Sam       | Bill       | $10     |
Then accounts are now
| Customer  | Balance  |
| Sam       | $90      |
| Bill      | $60      |
```

The scenario's Given / When / Then is analogous to a use case's Pre-conditions, Main Course, and Post-conditions. Use cases also have exceptions. These can be represented in Gherkin with another scenario:

```
Scenario: Attempt to transfer amount over balance generates an error
Given customers have accounts
| Customer  | Balance  |
| Sam       | $100     |
| Bill      | $50      |
When customer requests a transfer
| Customer  | Recipient  | Amount  |
| Sam       | Bill       | $120    |
Then transferred is denied and output is
| Message                        |
```

```
| Insufficient funds to transfer |
```

If you're familiar with standard requirement documents, you might look at the scenarios as examples of the way the requirement should work. This scenario could be equivalent to a textual "In the event that an amount over the balance is requested to be transferred, then the system should generate an error."

Requirements and Tests

Each requirement should have a corresponding test. One benefit of writing requirements in the Gherkin format, is that they can easily be used as tests. You need to set up two accounts, make the transfer, and then check that the amounts are correct. That is, everything is done according the Given and When in the scenario and the then is:

```
Then CHECK accounts are now
| Customer  | Balance  |
| Sam       | $90      |
| Bill      | $60      |
```

If the check fails, then the requirement is not met. This test is necessary, but not sufficient. You also need other tests for usability, security, and other non-functional requirements.

The requirement test is independent of the implementation. The application could be written in Java, Javascript, or use a SaaS platform, such as SAP. The test specifies what should be done, not how it should be done.

The tests do not have to be automated. They just need to be executed against the application. However, automation simplifies further development. When a new requirement is added, one can run all the tests against the previous requirements and ensure that they still work the same. Trying to run all those tests manually can lead to carpal tunnel syndrome.

The tests act as the documentation for the system. If they are automated, then anyone can run them to make sure the documentation represents the way the application really works.

Since the scenario represents both the requirement and the test for that requirement, then a test-requirement traceability matrix is not needed.

Business Rules and Domain Terms

Often one creates additional scenarios which represent the business rules and the customer terms (the ubiquitous language). For example, in the above scenario, there could be a business rule that states that the transfer must be no greater than ½ of the balance:

```
Scenario: Business Rule Amount in a Transfer
* Transfer amount must no greater than 1/2 of the balance
| Amount  | Balance  | Allowed  | Notes        |
| $50.00  | $100     | yes      | at limit     |
| $50.01  | $100     | no       | above limit  |
```

Since the business rule and the original scenario both involve dollars, which is customer term, then there would be a scenario that described the acceptable values for dollars.

```
Scenario: Domain Term Dollars
```

```
* Amounts in Dollars
| Value   | Valid  | Notes                    |
| $5.00   | yes    |                          |
| $5.0B   | no     | contains non-digit       |
| $5.001  | no     | only two decimal digits  |
```

Context

But do requirements drive the design of the code? It all depends on the context. In many areas, such as finance, the applications include numerous workflows and business rules. The application automates these workflows and rules. The previous scenario is a simplified version of a workflow that has numerous rules. The rules might include how often can a transfer occur, how much is a fee for the transfer, can a transfer be reversed, and so forth. Each of the rules would be described in a scenario. Most of the applications behavior would be expressed with customer understood scenarios.

In other areas, the requirements might specify just the external behavior. For an application that delivers videos, a scenario might be:

```
Scenario: Watch a movie
Given a customer wants to watch a movie
When the movie is playing
Then the movie plays smoothly without any noticeable pauses
```

There might be a scenario which documents a performance requirement for a large number of users:

```
Scenario: Lots of people watching movies
Given a million customers wanting to watch a movie
When the movies are playing
Then each movie plays smoothly without any noticeable pauses
```

You could have a little more detail in what “noticeable pauses” means. But most of the work will be in the technical details, such as how to represent the video stream, the caching that is required, and so forth.

In other application areas, you could specify scenarios with examples that show the form of the desired output, such as:

```
Scenario: Determine the fastest route between two points
Given a set of data on streets
When the user requests the fastest way to go from
| Current location  | Destination             |
| 732 Ninth Street  | 1802 West Main Street   |
Then the result is
| Drive                | Distance   |
| South on 9th Street  | .2 miles   |
| Left onto West Main  | .1 miles   |
```

The scenario documents what is expected and other scenarios can gives additional examples, such as shortest route or route with the fewest number of left turns. Other scenarios might show long trips across the country or a trip that cannot be made because there is no street to the destination. In these cases, you are showing variations in the request that will needed to be implemented. But most of the effort will be in the design of the algorithm that produces the results.

There are other applications where the desired behavior can be specified, but more in terms of statistics. This often occurs with artificial intelligence systems:

```
Scenario: Determine the objects in an image
Given input files are:
| Image File     | Object          |
| something.jpg  | African Lion    |
| other.jpg      | Siberian Tiger  |
| someother.jpg  | Black Bear      |
# … more
When input is processed
Then at least 95% of objects are correct.
```

The implementation learns on a training set of images and then the test for this scenario should pass. Otherwise, it may need more training or some configuration tweaking.

Naming Alternatives

The scenarios describe how the system behaves in meeting a requirement, so Requirement Driven Development can be translated to Behavior Driven Development. Because the same scenario acts as a test that the behavior is acceptable, Requirement Driven Development can be translated to Acceptance Test Driven Development. And since the requirement is specified with an example of real data, it can be translated to Specification by Example.

Summary

Specifying requirements with Gherkin scenarios can provide numerous benefits including a shared understanding among the team, reduced documentation, and a testable application. The benefit is greatest for applications which execute customer specified workflows and business rules. However other types of applications can have clear specifications for what are the requirements. The Gherkin scenarios can be used as the requirements, the tests, and the documentation, thus decreasing the effort to produce business value.

---

[1] There are some formal definitions of such as *A Requirements-Driven Software Development Methodology* http://www.cs.toronto.edu/km/tropos/tropos-icse.pdf

[2] Writing Effective Use Cases by Alistair Cockburn