Table of Contents

"Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing" - Gojko Adzic

Review and Notes

If you are having troubles collaborating with all stakeholders on user stories, if you find you are not clear on requirements, then the advise in this book will help you get started. This is the book I wish I had when I started down the pathway of using examples and “given-when-them” descriptions in place of conditions of acceptance criteria (see How Can We Improve Collaboration on User Stories? for more information).

One of the ideas that we push with agile development is the idea that we should move away from requirements documents and move to face-to-face communication to ensure that we are all on the same page. When teams start to do this, they often find that there has not been clear communication of requirements or something has been forgotten in the thinking process and so we feel like we are constantly re-visiting work that “if we only had a requirements document” we would have addressed. Another of ideas that really help teams get better in agile is the idea that you don't have a serial workflow in sprints, design - development - testing, but rather try to do more in parallel and in a more focussed way. If you are working either of these issues that this book will introduces you to (thinking) tools that will help.

To quote the book “On most projects even today, writing code is the first time that we try to make the solution really precise. At this point, a developer may have the same understanding of a point as the business person making the request, but I would not bet on it. Work out the probabilities from the experiment, and you’ll get about a 39% chance for this to happen. A tester will need to verify the result which asks for another mind alignment and brings down the probabilities to 20%.”

The book proposes a process of “agile acceptance testing” to help understand what it is we are going to build:

- Use specific examples, in table form, showing conditions that drive the example, the expected inputs and outputs. The idea is that tests and requirements are the same thing. Requirements are often driven from examples, and examples also end up as tests. With enough examples, you can build a full description of the future system.
- Get everyone, analyst, product manager, stakeholder, developer, tester involved in the generation of the examples so that all viewpoints are built into the system.
- Employ “just in time” detailing in the form of a “specification workshop” an event that happens at the beginning of the Sprint (at the latest) before anyone starts working on a story. “The workshop starts with a business person, typically the product owner, business sponsor or an analyst, briefly describing a piece of software that needs to be developed. He then explains how the code should work once it is developed, providing realistic examples. Other workshop participants ask questions, suggest additional examples that make the functionality clearer or expose edge cases that concern

them. One of the main goals of the specification workshop is to flush out these additional cases before the development starts, while business people are immediately available to discuss them."
- Capture the tests in terms of specifications, not scripts. Specifications can be in the form of tables of inputs and outputs based on conditions and / or using the given - when - then format:

> Given <some initial context>
> When <an event occurs>
> Then <ensure some outcomes>

- Tables are more useful for expressing specifications, state machine transitions and calculation-based rules. The expressiveness of natural language and the sequential style of writing help to get a better understanding of the process for workflows (given-when-then)
- These specific examples are also used to document the system (what it is expected to do) and so become a mechanism for traceability (for business situations that require this).
- The idea that acceptance tests are a "mistake-proofing device". In order for them to be effective, we have to be able to execute them frequently and quickly, without taking too much time from developers, analysts, testers or anyone else. The best way to achieve this is to automate as many tests as possible. This needs to be balanced with the requirement that acceptance tests also need to "human readable" so that people on the business side of the house can understand the system as well.
- The idea that we should create and use a common "domain" language and that the language should be used at all levels, even coding. There are many situations where developers will invent their own language to describe a concept, but this leads to miscommunication. If developers come up with a new concept, they should work with the business facing people about what to call that concept, so the system maintains a common language.
- The idea that you need both unit and acceptance tests. Unit tests act as a micro-target for code units and they check whether the code is correct from a technical perspective. Unit tests should examine edge cases such as empty strings, incorrect formats and various combinations of input arguments that the programmer is concerned about. Acceptance tests act as a macro-target for development of whole code modules and they verify that the product is correct from a business perspective. They examine realistic business cases that the customers and business people are concerned about. Unit tests 'will insure the code is built right', and that acceptance tests 'insure the right code is built'.
- Todays "acceptance test" becomes part of tomorrow's "regression test".
- The idea that we should treat defects as evidence of missing tests.

Don't be put off by the word "testing". This is really about specifying the functionality we are delivering and ensuring that we have common understanding of the requirements, have dealt with edge conditions and are clear about what is in and out of scope. If you want to introduce the practice in a corporate environment with strictly defined roles, avoid using the word "testing". Talk about executable specifications or communicating with examples instead of acceptance tests.

If you use the ideas in this book you can expect to see benefits (to quote the book):

- Product owners, Business Analysts and Project Managers: "Developers will actually read the specifications that you write. You will be sure that developers and testers understand the specifications correctly. You will be sure that they do not skip parts of the specifications. You can track development progress easily. You can easily identify conflicts in business rules and

requirements caused by later change requests. You'll save time on acceptance and smoke testing."
- Developer's: "Most functional gaps and inconsistencies in the requirements and specifications will be flushed out before the development starts. You will be sure that business analysts actually understand special cases that you want to discuss with them. You will have automated tests as targets to help you focus the development. It will be easier to share, hand over and take over code."
- Tester: "You can influence the development process and stop developers from making the same mistakes over and over. You will have a much better understanding of the domain. You'll delegate a lot of dull work to developers, who will collaborate with you on automating the verifications. You can build in quality from the start by raising concerns about possible problems before the development starts. You'll be able to verify business rules with a touch of a button. You will have a lot more time for exploratory testing. You will be able to build better relationships with developers and business people and get their respect."

Want to Know More?

- "Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing" by Gojko Adzic.

Book, Learning, Improvement, AcceptanceCriteria, ConditionsOfSatisfaction, Review, BDD, ATDD