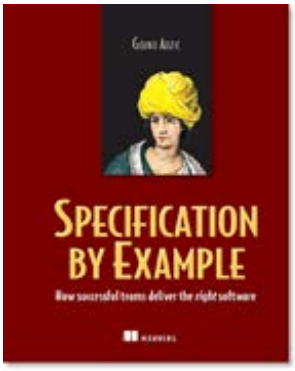
MANNING PUBLICATIONS



Specification by Example
By Gojko Adzic

The business model of ePlan Services is to offer 401(k) to small employers, which benefit from a cheaper cost of operation, resulting in a significant competitive advantage. Business process automation is a key factor. In this article, based on chapter 16 of Specification by Example, Adzic describes how the need to deliver a cheaper service and automate their business processes got ePlan Services on a path of improving their software development process, in which they implemented most of the ideas of Specification by Example.

You may also be interested in…

ePlan Services

ePlan Services is a 401(k) retirement pension service provider based in Denver, Colorado. It's a technology-driven business, relying heavily on an effective software delivery process. According to Lisa Crispin, an agile tester who works there, they use living documentation to deal with a complex domain and to facilitate the transfer of knowledge both for software development and for business operations.

The business model of the company is to offer services to small employers, which benefit from a cheaper cost of operation, resulting in a significant competitive advantage. Business process automation is a key factor. In 2003, they realized that their software delivery process would have to change to support the business. "We weren't getting the software out of the door; there were too many problems with the quality," says Crispin.

The need to deliver a cheaper service and automate their business processes got ePlan Services on a path of improving their software development process, in which they implemented most of the ideas of Specification by Example.

Changing the process

The company convinced Mike Cohn to take over the development team, and he helped them implement Scrum. Early on, they spent two days at the end of every iteration doing manual testing. All members of the team, including testers, developers, and database administrators, were running manual test scripts. This meant that one fifth of their iteration was spent on testing. Because of that, they decided to implement test automation. According to Crispin, unit testing was the first thing they had to get right:

Most of the bugs testers found before were unit-level bugs. You spent all your time with that and didn't have time for anything else.

While the developers were getting used to unit testing, the testers started with functional test automation. Without the help from the developers, the testers could automate tests only through the user interface. Eight months later, they had hundreds of unit tests and enough automated functional smoke tests to eliminate the need to do manual regression checking for unit-level bugs. Crispin says that this allowed them to start looking at the bigger picture:

We found really quickly that once developers had mastered TDD, we didn't have these bugs any more. We had more time for exploratory testing. Anything that we reported as a bug was usually because the developer didn't understand the requirement. Bugs in production were often something that we didn't understand.

Like in so many other cases, without efficient test automation the team had little time to deal with anything else. Once the technical unit-level bugs were no longer causing trouble, they could see the other problems as well.

Although they had some automated functional tests in place, this wasn't enough to prevent problems. Those tests were very slow, so they ran overnight and only checked happy-path scenarios. The team started looking at alternative ways to automate functional tests, to run more checks more quickly. They found FitNesse, but that required developers to help with automation. Crispin says that getting the developers engaged was a challenge:

Programmers are used to being rewarded for writing production code.

On Mike Cohn's suggestion, I just picked a story, went to a developer working on it, and asked if we could we pair up writing FitNesse tests on it. In the next sprint I picked a different story and a different person. We found a bug right away, where he didn't really understand the requirement. So developers immediately saw the value.

Collaborating on writing tests brought testers and developers together to discuss requirements and helped them write better tests. Crispin says that this eliminated most of the big problems:

> Within a year after we started doing agile, we felt comfortable that really bad bugs weren't going to production.

They also realized the importance of collaboration. Crispin says:

> The biggest benefit from this is getting us to talk together so that we have a mutual understanding of the requirements. That's more important than test automation. After we saw the benefits of collaboration, the product owner got excited about it as well and heard about Acceptance Test-Driven Development.

Efficient functional test automation required the developers to get involved, which resulted in much tighter collaboration between the developers and the testers. It also gave the team visible benefits, which helped to build a business case for further improvements. This inspired the team to take the process even further and prevent bugs from coming into the system instead of catching them with automated tests later.

Working to improve the process even more, they started using acceptance tests as specifications and collaborating on them. Crispin worked with their product owner up front to prepare examples. They had some early success, but they still looked at the examples as functional tests. When the team was working on automating compliance testing, one of the most complicated parts of the system, they over-specified tests and had to think much more about what they wanted to achieve with this approach. Crispin explained:

> The product owner and I sat down and wrote all these FitNesse tests to test algorithms. There are so many permutations, so we wrote a lot of very complex tests and a couple of sprints in front. When the developers started coding, they looked at the tests and were confused. They couldn't see the forest for the trees.

They realized that the developers couldn't handle too much information up front. After several experiments, the team decided only to write a high-level test up front to give the developers the big picture. When a developer picked up a story, he would pair with a tester on writing a happy-path scenario test and automate it. A tester could then extend the specification by adding more examples. Testers used this automated framework for exploring the system. If they found a case that failed the test, they'd go back to the developer to get it fixed. This changed the way they look at acceptance tests as specifications, according to Crispin:

> We had a vague idea at first that we could write acceptance tests ahead of time so that they could be the requirements. What changed over time is how much detail we need to put into tests up front, how many test cases is enough. I'm a tester; I could probably test something forever and keep thinking of things to test, but we have only two weeks. So we had to figure out how to internalize the risk analysis and say: Here are the tests we really need; here are the really important parts of the story that have to work.

As they shifted from thinking about automated tests to thinking about automated specifications, it became clear that the structure of what they specify and automate is primarily a communication tool, not a regression check. They simplified and refined them to ensure that developers have enough specifications just in time when they need them.

Good test design

Lisa Crispin is a well-known agile tester and the co-author of *Agile Testing*. I asked her about what makes good acceptance test design. This was her response:

- Good test design is key long term. People start testing and make a big suite of tests. All of the sudden, the effort they spend maintaining them is more than they are worth.
- Each test has to be clear about the essence of the test.
- As soon as you have some duplication, you have to extract it.
- Programmers or someone with strong code design skills needs to help design these tests. Once you have a template, it's easy to put in details.

Living documentation

Looking at examples more as specifications than tests, the team realized how powerful they are as documentation. Crispin says that having a living documentation system saved them a lot of time when they investigated issues:

> We'd get a call: "We have this loan payment and the amount of interest we applied isn't correct. We think there is a bug." I can look at the FitNesse test and put in the values. Maybe the requirements were wrong, but here's what the code is doing. That saves so much time.

At one point, the person who was a manager and senior developer at ePlan decided to move back to India and wouldn't be available for a couple of months. Crispin says that they started looking at applying Specification by Example to extract the unique knowledge he had about the system:

> When there was a strange problem, he always knew how to fix it. So we really had to get the knowledge that he had about the legacy parts of the system. We then decided that one person will get some time during each sprint to go over parts of their business process and document them.

This led them to start documenting the other parts of the system as well. Although they wrote tests for anything they were developing, there were still parts of the legacy system without test automation, and this sometimes caused problems. Creating an automated living documentation for these areas helped them discover inconsistencies in business processes. Crispin explains that:

> I'd been with the company for four years at that point, but I never understood how the cash accounting worked. I learned that we have five different bank accounts outside of the automated application. The money in these accounts is moved around via emails and phone calls, but the cash amounts must balance. When they don't, the accountant needs a way to research why. After the accountant explained this process to us, we documented it for future reference on our wiki. We were then able to produce reports with useful information about money in and out of the system. Now, when the cash is out of balance, she can use the reports to find out why.

Building a living documentation system to share the knowledge helped the development team learn about the business processes, and it gave the business users visibility of what they were actually doing. Writing things down

exposes inconsistencies and gaps. In this case, it made people think harder about what they're actually doing from a business perspective.

Current process

All these changes were implemented a while ago, and with a living documentation system the team has a relatively stable process. At the moment, the team consists of four programmers, two testers, a Scrum master, two system administrators, a database administrator, and a manager. They work in two-week sprints. Two days before the start of a sprint, the team meets with the product owner and the stakeholders. The product owner introduces all the stories planned for the next sprint, and they write high-level tests on a whiteboard. This enables the team to provide feedback on the plan and ask questions before the real sprint planning session.

With such a complex business and a small team, the product owner is a bottleneck. To enable him to work upstream with the business users, the testers take over some analysis responsibilities. The product owner often creates a "story checklist" upfront, containing the purpose of a story and rough conditions of satisfaction. For stories that involve user interfaces, he adds a UI mock-up into the story checklist. For stories that deal with complex algorithms, he adds a spreadsheet with examples.

Ultimately, the product owner also took on more work unrelated to software, so he often doesn't have enough time to prepare for the meeting. To work around that, the testers get his approval to get in touch with the upstream stakeholders directly and work with them on the specifications.

The iteration starts with a planning meeting, when they go through all the stories again and the product owner answers any open questions. They create screen mockups and illustrate requirements with examples. Testers merge that information with the story checklist, if it's available, and then refine the specifications and put them on a wiki site.

On the fourth day of the sprint, the two testers meet with the product owner and go over all the specifications and test cases in detail, to make sure they understand everything correctly. This allows the product owner to review the specifications and what the team is going to do in the iteration.

As soon as the specifications start appearing on the wiki, the developers start implementing the stories, and they show the results to the business users as soon as they are finished.

A-ha moments

I asked Crispin about her key ah-ha moments related to Specification by Example. She replied:

- I didn't own quality. My job was to help the customer understand quality and help the whole team define quality and make sure that it happens.
- Developers need to be engaged.
- The process requires patience. We had to take baby steps and couldn't take everything at once.
- A tool such as FitNesse can really help with collaboration. You think of it in a technical sense, that it's going to help you automate, but it changes the team culture and helps you communicate better.
- The real value of this is that we're talking.

Key lessons

Because of their business strategy, ePlan Services relies heavily on business process automation and efficient software delivery. As a way to improve quality and speed up software delivery, they had to move away from manual software testing. They initially focused on functional test automation but then found out that the shared understanding that comes from collaboration leads to much better software.

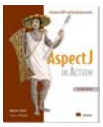
At first, they thought about the collaboration from a testing perspective and over-specified tests, making it hard for developers to use those documents as a target for development. Instead of covering every possible combination of values, they moved to specifying key examples, which made the process more efficient and provided developers with good specifications, just in time when they needed them.

Once they had a comprehensive set of executable specifications for a part of the system, they realized how useful it is to have living documentation, especially as a way to capture specialist knowledge. When they started documenting other parts of their business, a consistent living documentation system exposed inconsistencies and errors in their existing business processes.

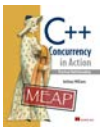
A living documentation system made the software delivery process much more efficient and enabled them to discover inconsistencies in their business processes.

Here are some other Manning titles you might be interested in:

Becoming Agile
...in an imperfect world
Greg Smith and Ahmed Sidky



AspectJ in Action, Second Edition
Enterprise AOP with Spring Applications
Ramnivas Laddad



C++ Concurrency in Action
Practical Multithreading
Anthony Williams

Last updated: August 4, 2011