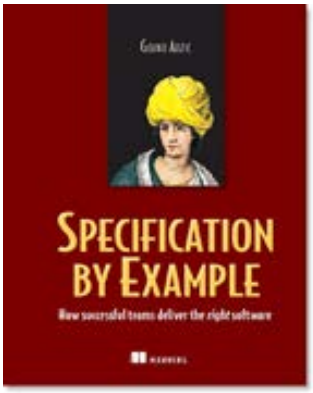
MANNING PUBLICATIONS



Specification by Example
Gojko Adzic

In this article from chapter 4 of Specification by Example, author Gojko Adzic explains how to work together with business users to come up with the right stories and that the key idea to achieve that is not to start with user stories but with business goals and derive the scope from that.

Building the Right Scope

Use cases, user stories, or backlog items already define broadly the scope of a project. Many teams consider such artifacts as something provided externally, a responsibility of the business users, product owners, or customers. Asking business users to provide the scope is effectively relying on someone who has no experience designing software solutions to give us a high level solution design. At this point, I'll play the card of the mandatory Fred Brooks quote in a software-related book: "The hardest single part of building a software system is deciding precisely what to build." Or, you can go to an even higher authority—Albert Einstein said that "the formulation of a problem is often more essential than its solution."

User stories are the most popular way of defining the scope for agile and lean projects at the moment. They did a fantastic job of raising awareness of business value in software projects. Instead of asking business users to choose between developing an integration platform and building transaction CRUD screens, we finally started talking to them about things that they could understand and reasonably prioritize. Each story should have a clearly associated business value. Business users often choose that value statement arbitrarily and that often is just the tip of the iceberg. But, when we know what a story is supposed to deliver, we can investigate that further and suggest an alternative solution if we can think of a better one. Christian Hassa explains that:

> People tell you what they think they need, and by asking them why you can identify new implicit goals they have. Many organizations are not able to specify their business goals explicitly. However, once you derived the goals, you should again reverse and derive scope from the identified goals, potentially discarding the originally assumed scope.

This is the essence of a practice I call challenging requirements. I think that challenging requirements is a very important practice, but it is reactive. While that is definitely better than passive, which best describes how most teams I've seen work with scope, there are emerging techniques and practices that allow teams to be much more proactive in achieving business goals. Instead of reacting to wrong stories, we can work together with our business users on coming up with the right stories in the first place. The key idea to achieve that is not to start with user stories but with business goals and derive the scope from that.

Try this (VIP): Understand the why and who

User stories generally have three parts: As a … I want … In order to…. There are alternative formats to stories, but they all have these three components. Understanding why something is needed and who needs it is crucial to

evaluating a suggested solution. The same questions can be applied to the project scope on a much higher level. In fact, answering those questions at a higher level can push a project into a completely different direction.

Peter Janssens from ILean in Belgium worked on a project where he was on the other end—the person giving requirements in the form of solutions. He was responsible for an application that stored local traffic sign information. It started as a simple Access database for Belgium but quickly grew to cover most of the countries in the world. The company had a data collector in each country and they were all using local Access databases, which were occasionally merged. To make the work more efficient and prevent merging problems, they decided to take the database online and make a web application to maintain it. They spent four months contacting suppliers, comparing bids, and finally selected one offer. The estimated cost for this application was around 100 000 euro. But, then, the project took a completely different turn after they thought about who needs to use the application and why. Janssens says:

> The day before the go/no, a guy from engineering asked again for the problem, to understand it better. I said again: "We need a web solution for the central database". He said: "No no, let's not jump to conclusions. Don't elaborate immediately which solution you want, please explain it to me." I explained again. And then he said: "So, your problem is actually that you need a single source to work on because you are losing time on merging". "Yes," I said, "correct." He had a second question: "Who is working on it?" I said: "Look, at this moment we had 10 groups of countries, so 10 people." I even gave him the names. We had a look at the databases and understood that this type of traffic information doesn't change that often, maybe once or twice a year per country. He said: "Look Peter, your problem will be solved by tomorrow." By tomorrow he added the database to their Citrix [remote desktop] server.

The application had to support ten users in total and they were only using it to update traffic sign information, which did not change very often. The Access application could cope with the volume of data. The only real problem they had was merging. Once a technical person understood the underlying problem, he could suggest a much cheaper solution than the one originally suggested. Janssens explains that:

> What I learned out of that is that this was a real confrontation situation—it's always important to understand the core problem that leads to a request. So understanding "why" is important. Eventually, the Citrix solution came to him when we talked about the "who" question. On average there was one user a month working on it.

Even at a scope level, the solution is already implied. Without going into the possible user stories or use cases and discussing specifications for tasks, the fact that someone suggested a web application implies a solution. Instead of spending five months selecting a supplier and who knows how long to get the project finally live, they actually solved the problem with a quick fix that cost nothing. This is an extreme case, but understanding why someone needs a particular application and how they will use it often leads to better solutions.

Try this (VIP): Understand where the value comes from

In addition to helping us design a better solution, understanding where the value comes from helps immensely with prioritization. Rob Park's team at a large US insurance provider looks at prioritization only from a higher feature level, saving a lot of time from having to go through the same process on the lower story level. Park says:

> We keep things at a high level, describe the business value and what's really core to the feature. We break the feature down into stories, which we try to make as small as possible. An example of a feature would be: Deliver the proof of insurance as PDF for 14 states. The big thing I've been trying to push especially from the business side is how much is this worth—put the dollar value behind it. In that particular case, we were able to get one of the senior guys to say: "Well, 50 percent of the calls were for this and 50 percent of those calls were for the proof of insurance cards, so 25 percent of the calls were dealing with that." They know how many calls they have

> and how much they would be able to save by generating this PDF instead of having to do all the copy and paste stuff that they were doing before, so they were actually able to put some numbers behind this, which was really cool.

Raising the discussion to the level of goals allows teams to deal with scope and priorities more efficiently than just at the story level. One good example where this helps is effort estimation. Rob Park's team found that discussing goals enabled them to stop wasting time estimating effort for individual stories:

> We don't really want to bother with estimating stories. If you start estimating stories, with Fibonacci numbers for example, you soon realize that anything eight or higher is too big to deliver in an iteration, so we'll make it one, two, three, and five. Then you go to the next level and say five is really big. Now that everything is one, two, and three, they are now really the same thing. We can just break that down into stories of that size and forget about that part of estimating and then just measure the cycle time to when it is actually delivered.

In Software by Numbers,[1] Mark Denne and Jane Cleland-Huang build a formal method for prioritization driven by business value expressed through Minimum Marketable Features. From my experience, predicting how much money something is going to earn is as equally hard and error-prone as predicting how long it's going to take to implement. But, if your domain is such that you can actually put numbers on features, this will help a lot with getting the business users involved. Business users can get engaged a lot better on a higher level. Asking them to prioritize features or even business goals works a lot better than asking them to prioritize low-level stories or tasks.

Try this (VIP): Understand what outputs the business users expect

When goals are hard to pin down, a useful heuristic is to start with the expected outputs of the system and investigate why they are needed and how the software can provide them. Once we nail down the expected outputs, we can focus our work on fulfilling the requirements that they imply. Analyzing why those outputs are required leads to the goals of the project.

Starting with the outputs of a system to derive scope is an idea from the BDD community, where it is called Outside-In Design. This idea has been getting a lot of traction recently, because it eliminates a common problem. On a lot of my early projects, we focused on the process flow and putting the data into the system initially. We left the end results of processes, such as reports, for later. The problem with this approach is that the business users can really get engaged only when they see end results, which often led to rework when the visible output finally comes out.

Wes Williams worked on a project at Sabre where they delayed building the user interface and says that this caused a lot of rework:

> The acceptance tests were written against a domain [layer], before our customer could see the GUI. The UI was delayed for about four months. The customer thought completely differently about the application when they saw the UI. When we started writing tests for the UI, they had much more in them than the ones written for the domain [layer]. So the domain code had to be changed, but the customer assumed that that part was done. They had their test there, they drove it and it was passing; they assumed it was done.

Instead of trying to collaborate with business users on specifying how to put things into the system, we should start with examples of outputs. This helps to engage business users in the discussion better and gives them a clear picture of what they will get out of the system. Once we understand the outputs together, we can start looking at how software can help them.

---

[1] Mark Denne and Jane Cleland-Huang. Copyright © 2003. Prentice Hall. Software by Numbers. Low-Risk, High-Return Development. 0131407287.

Try this: Business specify "so that"—developers provide "I want"

When: Business users trust the development team

A great way to get to the right scope for a goal is to clearly put the responsibility for a solution on the development team. The team at uSwitch collaborates with their business users on defining user stories. The business users specify the stakeholder and the expected benefit, and the development team specifies the solution. In the standard user story format ("As a... So that ... I Want"), this would mean that the business users specify the "as a" and "so that" parts, but developers provide the "I want" part.

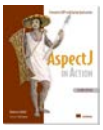
If you have the luxury of high-level control of project scope, make sure to involve developers and testers in the discussions about that and focus the suggested solutions on fulfilling clearly defined business goals. This eliminates much of the unneeded work later on and sets the stage for better collaboration on specifications.

Remember

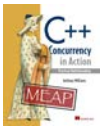
- When you get requirements as tasks, push back and understand the real problem, then collaboratively design the solution.
- Think about the business goal of a milestone and the stakeholders who can contribute or be affected by that milestone to derive the appropriate scope.
- If you can't avoid getting tasks, ask for high level examples of how they would be useful to understand who needs them and why, and then design the solution.
- Specify using examples and define acceptance criteria on both the higher (feature) and lower (story) levels.

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: July 6, 2011