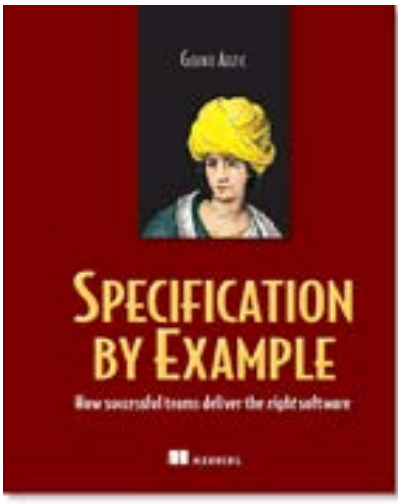
MANNING PUBLICATIONS




Specification by Example
By Gojko Adzic

*Specification by Example is conceptually different from traditional specification or testing processes, especially in the way it relies on collaboration. Manning author Gojko Adzic interviewed a number of teams to see how they approached collaboration on specifications. Their approaches varied greatly, from large all-hands workshops to smaller workshops, and even to informal conversations. In this article, based on chapter 6 of Specification by Example, Adzic describes the most common collaboration models along with specific benefits of each.*



# *The Most Popular Collaborative Models*

Although all the teams I interviewed collaborated on specifications, the ways they approached that collaboration varied greatly, from large all-hands workshops to smaller workshops, and even to informal conversations. Here are some of the most common models for collaboration along with the benefits the teams obtained.

## *Try big, all-team workshops*

### *When: Starting out with Specification by Example*

Specification workshops are intensive, hands-on domain and scope exploration exercises that ensure that the implementation team, business stakeholders, and domain experts build a consistent, shared understanding of what the system should do. The workshops ensure that developers and testers have enough information to complete their work for the current iteration.

> Big specification workshops that involve the entire team are one of the most effective ways to build a shared understanding and produce a set of examples that illustrate a feature.

During these workshops, programmers and testers can learn about the business domain. Business users will start understanding the technical constraints of the system. Because the entire team is involved, the workshops efficiently use business stakeholders' time and remove the need for knowledge transfer later on.

Initially, the team at uSwitch used specification workshops to facilitate the adoption of Specification by Example. Jon Neale describes the effects:

> It particularly helped the business guys think about some of the more obscure routes that people would take. For example, if someone tried to apply for a loan below a certain amount, that's a whole other scenario [than applying for a loan in general]. There's a whole other raft of business rules that they wouldn't have mentioned until the last minute.

> Specification workshops helped them think about those scenarios up front and helped us go faster. It also helped the development team to interact with the other guys. Having that upfront discussion helped drive the whole process—there was a lot more communication straight away.

### Implementing Specification workshops into PBR workshops

Product Backlog Refinement (PBR) workshops are one of the key elements of well-implemented Scrum processes. At the same time, I've found that most teams that claim to run Scrum actually don't have PBR workshops. PBR workshops normally involve the entire team and consist of splitting large items on the top of the backlog, detailed analysis of backlog items, and re-estimation. In *Practices for Scaling Lean and Agile*,[1] BasVodde and Craig Larman suggest that PBR workshops should take between5and 10 percent of each iteration.

Illustrating requirements using examples during a Product Backlog Refinement workshop is an easy way to start implementing Specification by Example in a mature Scrum team. This requires no additional meetings and no special scheduling. It's a matter of approaching the middle portion of the PBR workshop differently.

The Talia team at Pyxis Technologies runs their workshops like this. André Brissette explains this process:

"This usually happens when the product owner and the Scrum master see that the top story on the backlog is not detailed enough. For example, if the story is estimated at 20 story points, they schedule a maintenance workshop during the sprint. We think that it's a good habit to have this kind of a session every week or every two weeks in order to be certain that the top of the backlog is easy to work with. We look at the story; there is an exchange between the product owner and the developers on the feasibility of it. We draw some examples on the whiteboard, identify technical risk and usability risks, and developers will have to make an evaluation or appraisal of the scope. At this time we do planning poker. If everyone agrees on the scope of the feature and the effort that it will take, then that's it. If we see that it is a challenge to have a common agreement, then we try to split the story until we have items that are pretty clear and the effort is evaluated and agreed to."

Large workshops can be a logistical nightmare. If you fail to set dates on a calendar up front, people might plan other meetings or not be readily available for discussions. Regularly scheduled meetings solve this issue. This practice is especially helpful with senior stakeholders who want to contribute but are often too busy. (Hint: call their secretary to schedule the workshops.)

If you have a problem getting enough time from business users or stakeholders, try to fit into their schedule or work on specifications during product demos when they're in the room. This is also effective if the business users and delivery team don't work from the same location.

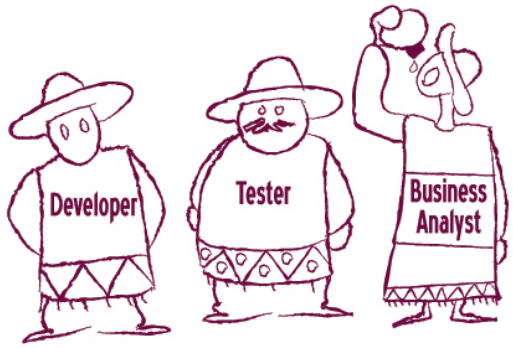
Large workshops are an effective way to transfer knowledge and build a shared understanding of the requirements by the entire team, so I highly recommend them for teams that are starting out with Specification by Example. On the other hand, they cost a lot in terms of people's time. Once the process matures and the team builds up domain knowledge, you can move on to one of the easier alternatives.

## *Try smaller workshops ("Three Amigos")*

### *When: Domain requires frequent clarification*

Having a single person responsible for writing tests, even with reviews, isn't a good approach if the domain is complex and testers and programmers frequently need clarification.

[1] Craig Larman and BasVodde, *Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum* (Pearson Education, 2010).




Run smaller workshops that involve one developer, one tester, and one business analyst.

A popular name for such meetings is Three Amigos. Janet Gregory and Lisa Crispin suggest a similar model for collaboration in *Agile Testing*,[2] under the name The Power of Three. (I used to call such workshops Acceptance Testing Threesomes until people started complaining about the innuendo.)

A Three Amigos meeting is often sufficient to get good feedback from different perspectives. Compared to larger specification workshops, it doesn't ensure a shared understanding across the entire team, but it's easier to organize than larger meetings and doesn't need to be scheduled up front. Smaller meetings also give the participants more flexibility in the way they work. Organizing a big workshop around a single small monitor is pointless, but three people can sit comfortably and easily view a large screen.

To run a Three Amigos meeting efficiently, all three participants have to share a similar understanding of the domain. If they don't, consider allowing people to prepare for the meeting instead of running it on demand. Ian Cooper explains this:

> The problem with organizing just a three-way is that if you have an imbalance of domain knowledge in the team, the conversation will be led by the people with more domain expertise. This is similar to the issues you get with pairing [pair programming]. The people knowledgeable about the domain tend to dominate the conversation. The people with less domain expertise will sometimes ask questions that could have quite a lot of interesting insight. Giving them an option to prepare beforehand allows them to do that.

A common trick to avoid losing the information from a workshop is to produce something that closely resembles the format of the final specification. With smaller groups, such as the Three Amigos, you can work with a monitor and a keyboard and produce a file. Rob Park worked on a team at a large U.S. insurance provider that collaborated using Three Amigos. Park says:

> The output of the Three Amigos meeting is the actual feature file—Given-When-Then. We don't worry about the fixtures or any other layer beneath it, but the acceptance criteria is the output. Sometimes it is not precise—for example, we know we'd like to have a realistic policy number so we would put in a note or a placeholder so we know we're going to have a little bit of cleanup after the fact. But the main requirement is that we're going to have all these tests in what we all agree is complete, at least in terms of content, before we start to code the feature.

Stuart Taylor's team at TraderMedia has informal conversations for each story and produces tests from that. A developer and a tester work on this together. Taylor explains the process:

[2] Lisa Crispin and Janet Gregory, Agile Testing: A Practical Guide for Testers and Agile Teams (Addison-Wesley Professional, 2009).

> When a story was about to be played, a developer would call a QA and say, "I'm about to start on this story," and then they would have a conversation on how to test it. The developer would talk about how he is going to develop it using TDD. For example, "For the telephone field, I'll use an integer." Straightaway the QA would say, "Well, what if I put ++, or brackets, or leading zeros, etc."
>
> The QA would start writing [acceptance] tests based on the business acceptance criteria and using the testing mindset, thinking about the edge cases. These tests would be seen by the BA and the developer. During showcasing we'd see them execute.

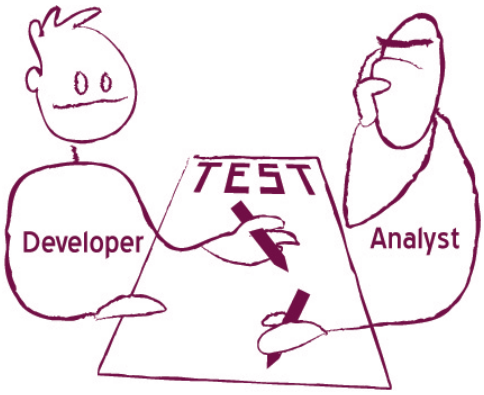
Producing a semiformal test collaboratively ensures that the information won't get distorted during automation later on. It also helps to share knowledge about how to write good specifications with examples; this is only feasible if the entire group can sit around a single monitor and a keyboard. Don't try to draft semiformal documents in an all-hands workshop, because it won't encourage everyone to get involved.

> Teams that work on mature products and already have a good knowledge of the target domain don't necessarily have to run meetings or have separate conversations to discuss the acceptance criteria for a story. Developers and testers might not necessarily need to provide as much input up front into the specifications, and they can resolve small functional gaps and during implementation. Such teams can collaborate with informal conversations or reviews.

## *Pair-writing*

### *When: Mature products*

Even in cases where the developers knew enough to work without big workshops, teams found it useful to collaborate on writing specifications with examples.




> Analysts can provide the correct behavior but developers know the best way to write a test so that it's easy to automate later and fits into the rest of the living documentation system.

Andrew Jackman's team at BNP Paribas works on a relatively mature product. They have experimented with different models of writing tests and concluded that they need to get both business analysts and developers involved in writing the tests. He says:

> When developers were writing the tests, it was easy to misunderstand what the story is about. If you don't have the interaction with the business analysts, it's only the developers' view of a thing. We moved to BAs writing the tests and that made a big difference. The challenge is when they write a story, that story might influence a number of existing tests, but they can't foresee that. The BAs like to write a test that shows a workflow for a single story. Generally that leads to

> a lot of duplication because a lot of the workflows are the same. So we move bits of the workflow into their own test.

Some teams—particularly those in which the business analysts cause a bottleneck or don't exist at all—get testers to pair with programmers on writing tests. This gives the testers a good overview of what will be covered by executable specifications and helps them understand what they need to check separately The team at Songkick is a good example. Phil Cowans explains their process:

> QA doesn't write [acceptance] tests for developers; they work together. The QA person owns the specification, which is expressed through the test plan, and continues to own that until we ship the feature. Developers write the feature files [specifications] with the QA involved to advise what should be covered. QA finds the holes in the feature files, points out things that are not covered, and also produces test scripts for manual testing.

Pairing to write specifications is a cheap and efficient way to get several different perspectives on a test and avoid tunnel vision. It also enables testers to learn about the best ways to write specifications so that they're easy to automate, and it allows developers to learn about risky functional areas that need special attention.

## *Have developers frequently review tests before an iteration*

### *When: Analysts writing tests*

> Get a senior developer to review the specifications.

The business users that work with Bekk Consulting on the Norwegian Dairy Herd Recording System don't work with developers when writing acceptance tests, but they frequently involve developers in reviewing the tests. According to Mikael Vik, a senior developer at Bekk Consulting, this approach gives them similar results:

> We're always working closely with them [business users] on defining Cucumber tests. When they take their user stories and start writing Cucumber tests, they always come and ask us if it looks OK. We give them hints on how to write the steps and also come up with suggestions on how our Cucumber domain language can be expanded to effectively express the intention of the tests.

If developers aren't involved in writing the specifications, they can spend more time implementing features. Note that this increases the risk that specifications won't contain all the information required for implementation or that they may be more difficult to automate.

## *Try informal conversations*

### *When: Business stakeholders are readily available*

Teams that had the luxury of business users and stakeholders sitting close by (and readily available to answer questions) had great results with informal ad hoc conversations. Instead of having big scheduled workshops, anyone who had a stake in a story would briefly meet before starting to implement it.


Informal conversations involving only the people who will work on a task are enough to establish a clear definition of what needs to be done.

"Anyone who has a stake" includes the following:

- The analysts who investigate a story
- The programmers who will work on implementing it
- The testers who will run manual exploratory tests on it
- The business stakeholders and users who will ultimately benefit from the result and use the software

The goal of such informal conversations is to ensure that everyone involved has the same understanding of what a story is about. At LMAX, such conversations happened in the first few days of a sprint. Jodie Parker explains:

> Conversations would be done on demand. You've got the idea and your drawings, and you really understand how it is going to be implemented. If you've not already written down the acceptance tests, a developer and a tester can pair on this. If the conversations didn't happen, things would end up being built but not being built right.

Some teams, such as the one at uSwitch.com, don't try to flush out all the acceptance criteria at this point. They establish a common baseline and give testers and developers enough information to start working. Because they sit close to the business users, they can have short conversations as needed.

Some teams decide whether to have an informal discussion or a larger specification workshop based on the type of the change introduced by a story. Ismo Aro at Nokia Siemens Networks used this approach:

> We have an agreement to have ATDD test cases [specifications], not necessarily a meeting. If the team feels it's coming naturally, then it's OK not to do the meeting. If it seems harder and they need input from another stakeholder, then they organize an ATDD meeting [Specification workshop]. This might be due to the team knowing a lot about the domain. When you are adding a small increment to the old functionality, it's easier to figure out the test cases.

## *Remember*

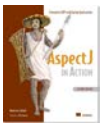
- Specification by Example relies heavily on collaboration between business users and delivery team members.
- Everyone on the delivery team shares the responsibility for the right specifications. Programmers and testers

have to offer input about the technical implementation and the validation aspects.

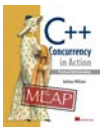
- Most teams collaborate on specifications in two phases: Someone works up front to prepare initial examples for a feature, and then those who have a stake in the feature discuss it, adding examples to clarify or complete the specification.
- The balance between the work done in preparation and the work done during collaboration depends on several factors: the maturity of the product, the level of domain knowledge in the delivery team, typical change request complexity, process bottlenecks, and availability of business users.

**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: October 19, 2011