Gojko Adzic

Specification by Example

How successful teams deliver the *right* software

MANNING

Table of Contents

6

Specifying collaboratively

Specification by Example is conceptually different from traditional specification or testing processes, especially in the way it relies on collaboration. Specification by Example won't work if we write documents in isolation, even if we implement all the other patterns described in this book.

In *Bridging the Communication Gap*, I focused on large, all-team specification workshops as the primary tool for collaborating on specifications. Probably the biggest lesson I've learned in working on this book is that the situation is a lot more complicated. Different teams in different contexts have their own way of collaborating on specifications, to the extent that even teams from the same group approach collaboration differently.

In this chapter, I present the most common models for collaboration on specifications, including big workshops, smaller workshops, and the most popular alternatives to workshops. This will help you understand the benefits and downsides of various approaches to collaborative specifications. I also present good practices for preparing for collaboration and ideas that will help you choose the right collaboration model for your team. But let's first deal with the question of whether collaboration is required at all.

In order to properly present an example of a collaboration on specifications, we also need to review a related practice: illustrating using examples. You'll read an example of how a specification workshop would play out in chapter 7, in the section "Illustrating using examples: an example."

Why do we need to collaborate on specifications?

Specifying collaboratively is a great way to build a shared understanding of what needs to be done and to ensure that different aspects of a system are covered by the specifications. Collaboration also helps teams produce specifications that are easy to understand and tests that are easy to maintain.

Chapter 6. Specifying collaboratively. Specification by Example: How successful teams deliver the right software, ISBN: 9781617290084

According to Jodie Parker, failure to collaborate on specifications was one of the biggest problems when they started implementing Specification by Example at LMAX. She says:

> People just don't realize how valuable a conversation could have been. Developers initially thought that testers aren't interested in the conversations because they were technical, but testers could learn about how to interrogate the code base or they could advise on the potential impact on other tests or changes to the language. Testers also thought that they were too busy. You can only see how valuable this [collaborating on specifications] is by doing it.

Even with perfect understanding of the business domain covered by a software system (and I've never seen a team with that), it's still worth collaborating on specifications. Analysts and testers may know what they want to specify and test but not necessarily how to organize that information to make it easy to automate and drive development—programmers will. Marta Gonzalez Ferrero worked on a project where the testers initially wrote all the acceptance tests themselves, without thinking of them as specifications. She says that, frequently, the developers couldn't use such tests:

> At the very beginning, testers were working on FitNesse tables and handed them over to developers. This caused problems because developers were coming back saying that pages weren't easy to understand or easy to automate. After that, they started working together.

A failure to collaborate on defining specifications and writing acceptance tests is guaranteed to lead to tests that are costly to maintain. This was one of the most important lessons about test design for Lisa Crispin. She explained:

> Whenever we had to make a change, we had too many tests [executable specifications] that we had to change. It's hard to refactor when you have many tests. I should have paired with developers to help me design the tests. I could easily formulate the questions; I see what's wrong. Testers knew basic concepts as Don't Repeat Yourself but didn't have a good understanding of the tools.

Because Crispin didn't collaborate with developers on writing and automating executable specifications, she wrote too many specifications and they weren't automated in a way that made long-term maintenance easy.

Many teams I interviewed made similar mistakes early on. When developers wrote specifications in isolation, those documents ended up being too closely tied to the software design and hard to understand. If testers wrote them in isolation, the documents were organized in a way that was hard to maintain. In contrast, successful teams quickly moved on to more collaborative work models.

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. I explain them in detail in *Bridging the Communication Gap*. 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*,[†] Bas Vodde and Craig Larman suggest that PBR workshops should take between 5 and 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.”

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

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.)

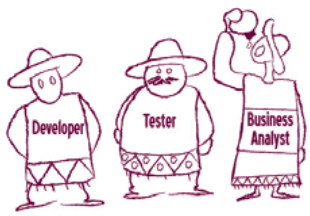
If you have a problem getting enough time from business users or stakeholders, try to fit into their schedule or work on specifications during product demos when they're in the room. This is also effective if the business users and delivery team don't work from the same location.

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.



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*,[1] 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.

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

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:

> 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.

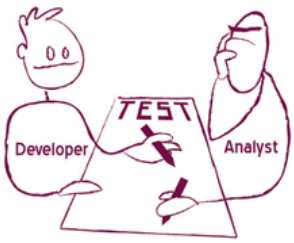
Teams that work on mature products and already have a good knowledge of the target domain don't necessarily have to run meetings or have separate conversations to discuss the acceptance criteria for a story. Developers and testers might not necessarily need to provide as much input up front into the 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 (see chapter 12 for more information).

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.

Preparing for collaboration

Collaborating on specifications is a great way to ensure shared understanding and flush out intricate details that people would never think about in isolation. If the topic of discussion requires a lot of up-front analysis or the team members don't have the same level of knowledge, starting from scratch in the discussions can be inefficient and frustrating. To address this, many teams introduced a preparatory phase, shown in figure 6.1, to ensure that the features are described in enough detail to facilitate a fruitful discussion.

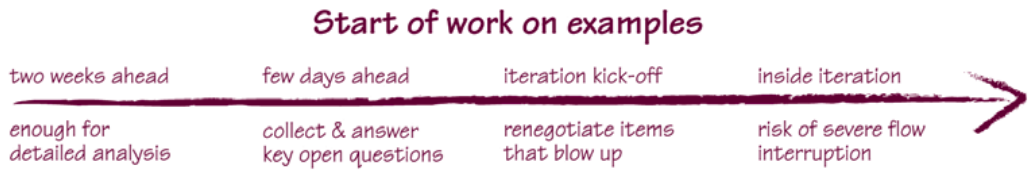


Figure 6.1 Teams generally fall into four groups depending on when they start working on examples. Those who need more time for analysis and chasing open questions start earlier.

This preparation involves working with stakeholders upstream to prepare some initial examples and initial analysis. Depending on the availability of team members, it can be done by either a single person—often in an analyst role—or a small group of senior people.

Hold introductory meetings

When: Project has many stakeholders

Teams with many stakeholders (for example, when the software is used by many departments inside a company or has several external customers driving requirements) in general you should run an introductory meeting several days before the start of an iteration. Some teams call this meeting pre-planning.

The purpose of an introductory meeting is to gather some initial feedback on upcoming stories and filter the ones that are too vague to be accepted into planning.

The introductory meeting isn't supposed to deliver perfectly refined specifications but to give the team enough time to gather external feedback on key issues that could be quickly identified. This isn't the iteration planning or Scrum planning meeting. Running the introductory meeting several days before the start of a sprint gives the team an opportunity to discuss open questions with remote stakeholders before the real specification refinement or planning meeting.

Many teams define high-level acceptance criteria in this introductory meeting, with bullet points rather than detailed examples. This helps focus the later work by specifying the basic cases they will test.

With smaller teams, such as the team at ePlan Services, developers, stakeholders, the project manager, and the product owner participate in this introductory meeting. With larger teams or groups of teams, only a few people participate. At Sky Network Services, which has six teams, each team sends two or three people to this meeting.

Involve stakeholders

A collaborative specification process works because it taps into the collective brain of business users and development team members and ensures that they all understand the specifications in the same way.

Many teams involved their business analysts or product owners, but not the customer stakeholders, in the discussions. In those cases, the teams consistently delivered products that met the business analysts' or product owners' expectations. But these expectations often weren't what the end users wanted. As far as I'm concerned, business analysts are part of the delivery team, not customer representatives.

To get the best results, actual stakeholders have to be involved in the collaboration on specifications. They are the ones who can really make decisions.

When a project has many interested parties, all the requirements are often funneled through a single person, typically called a product owner. This works well for scope and prioritization but not specifications. Lisa Crispin's team at ePlan Services ran into this problem. She says:

> The product owner wanted to own everything but at the same time he can't get everything right. He is doing the job of three or four people. Nobody has the bandwidth to do everything. Sometimes we need an answer to complete a story, but he can't provide that answer. He didn't understand the accounting requirements, for example. We still had to go and talk to stakeholders directly to understand that.
>
> He felt that we were going around him, so we had to really find the balance of keeping the product owner in the loop and still getting the information from the people who are going to be using that functionality. If there was a difference, we had to get them in the room to discuss it.

A single person can't possibly know everything about everything. Having a single decision maker on board to determine priorities is a must, but once the top-priority story gets selected, the team must try to collaborate with the relevant stakeholders on specifications for particular stories. In *Practices for Scaling Lean and Agile*, Larman and Vodde make the distinction between clarification and prioritization. They argue that prioritization must always be done by one person but that clarification can be done by the team itself.

It's important to involve the end stakeholders even if the team thinks they know the domain well enough to build good specifications by themselves. Mike Vogel worked on a technical data management project where the developers understood parts of the domain and its technical constraints better than the end users. To meet the project schedule, they were frequently forced to limit or exclude the stakeholders from collaboration on specifications—which Vogel thinks was one of their biggest mistakes. He says:

> We started doing too much of the test creation and definition of the acceptance criteria ourselves. So we could set up the meta programming that drove the system faster than them and we were under heavy schedule pressure. But there would be subtleties that neither we nor the customer understood, and they weren't able to pick that up from the tests.

If possible, include the actual stakeholders in the collaboration on specifications. This will ensure that you get the right information from an authoritative or dependable source and reduce the need for up-front analysis.

In larger organizations this might require some persuasion and politics, but it will be worth it. If your team has one, work with the product owner on finding a way to get in touch with stakeholders directly without interfering with the product owner's stakeholder management responsibilities.

Undertake detailed preparation and review up front

When: Remote stakeholders

Teams with remote stakeholders should have at least one person working on preparing detailed examples ahead of the team.

In the teams I interviewed, the person who worked ahead of the team was typically a business analyst or a tester. They worked with the stakeholders to analyze the requirements, agree on the structure of the examples, and capture the values for the most important cases. Teams working with vague requirements that needed a lot of analysis and clarification also had one person working ahead of the team.

In most teams, a developer also reviewed the initial examples early to provide technical feedback. This guaranteed that the team detected most functional gaps and questions early on. Stakeholders could then answer these questions up front, so the team wouldn't get stuck when collaboratively reviewing a story.

Many teams failed to implement this step when they started out, especially if they based their process on time-bound iterations. It seems logical that everything related to a particular story should be done within a single sprint or a single iteration. If the domain is complex, time-boxing both the specification and the development effort to a single iteration can cause developers to get stuck frequently.

The Sierra team at BNP Paribas tried to time-box everything in the same iteration, but they found this approach didn't allow them to work efficiently. Instead, their business analyst started working one step ahead of the rest of the team. Andrew Jackman says:

> Our project manager who's effectively the product owner will have prepared in advance the stories that he wants us to play. He and the business analyst already had them for the next iteration up on the board, and the business analyst went though preparing the acceptance tests. We used to not do this, but when developers tried to write a [acceptance] test we suddenly asked questions and found out that we were missing analysis.

Putting the initial examples together ahead of an iteration also enabled the team members to be better prepared for the collaborative discussion. Ian Cooper's team at Beazley uses this approach. Their business analysts and stakeholders are based in the United States, but the development team is in the UK. He says:

> Given the nature of the product and the fact that we're serving U.S. customers, time zones became an issue and there is no real easy access to the customer. Business analysts are proxies and they often took questions to answer later. Developers knew a lot about the domain, so analysts and developers were running the show. Testers didn't really participate.
>
> We found it easier to get the analyst to do the first pass through what was required and then come to the meeting. Testers will quite often run through all possible scenarios and ask about edge cases. Testers are given more time to read through stuff and understand it, to think about what the issues might be. This enables them to participate much better.

If stakeholders can't participate in the collaboration for specifications, then the risk that the delivery team will misunderstand their goals increases substantially. To reduce risk, teams with remote users performed more analysis up front than the teams who had direct access to their business users. Doing so requires an analyst to work upstream with the business users and stakeholders, so other team members may need to take over some of the analysts' downstream tasks.

If you decide to start analysis before the iteration, be sure that this responsibility is assigned to a dedicated team member to avoid dragging the entire team into it; this defeats the point of iteration scope.

Have team members review stories early

When: Analysts/domain experts are a bottleneck

If analysts or subject matter experts cause a bottleneck in the process, they won't be able to conduct much analysis before the relevant iteration. This might not be a problem if the stakeholders are readily available to answer questions or if the product is mature; functional gaps won't appear late in development.

On the other hand, if the team finds that they don't have enough information to write the executable specifications, someone has to provide analysis earlier. That someone doesn't necessarily have to be a business analyst or a subject matter expert. It could be a tester or a developer.

Developers and testers can help to take the load off domain experts (when they're causing a bottleneck) and do a first-pass review to spot the common problems. This increases the overall throughput of the team and also helps to build cross-functional teams.

Clare McLennan worked on a web advertising project where the stakeholders were in Germany and the team was in New Zealand—almost 12 hours apart. The testers played the role of local analysts. They couldn't make decisions for the customers, so they worked ahead of the team. McLennan says:

> To avoid the time zone problems we had to make sure that we have a handle on a story. If the testers read through it and it makes sense to them, they interrupt a programmer to make sure that it makes sense to them as well.

For the Global Talent Management team at Ultimate Software, the product owner is busy so the rest of the team helps with analysis work. A "cell" consisting of two developers and a tester reviews each story early on to prepare for the meeting with the product owner, identifying any open questions. Maykel Suarez says that this approach helped them use everyone's time more efficiently:

> The bigger team, around 17 people, put a lot of pressure on decision-making. The solution was to create cells. Now a cell (one tester, two developers) is able to make decisions more quickly. The flow process allowed working on those preparation meetings in chunks smaller than two-week iterations, usually just two-three stories. So, having three people in a meeting for 15–30 minutes every 3–5 days didn't seem like a waste of time or resources.

Prepare only initial examples

When: Stakeholders are readily available

The teams with stakeholders who are readily available to answer questions didn't spend too much time preparing detailed examples up front. They still found it useful to identify some initial examples, to get the basic structure in place before the discussion.

Identifying initial examples gets the basic structure in place and helps discussions run more efficiently.

André Brissette often uses examples provided by the external customers to start the work on specifications for the Talia project at Pyxis Technologies. He's the business stakeholder for the development team and also works with external customers. When the customers propose new functionality, they send examples of how the system would work; those examples become part of the future specification.

The team at uSwitch works in the same location as their stakeholders, so they don't need a lot of up-front preparation. Anyone on the team can suggest a new story during a stand-up meeting; the person who makes the suggestion often prepares basic examples beforehand.

Having initial examples ready from the beginning helps run the discussion more efficiently because the team doesn't have to experiment with the best structure of examples to illustrate a requirement or identify key attributes. They can instead focus on understanding the initial examples and extending them.

Don't hinder discussion by overpreparing

A preparatory phase shouldn't replace collaboration. It should just make the collaboration more effective. Some teams prepared too much information up front because testers approached the executable specifications from the perspective of combinatorial functional regression checking. They specified every possible combination of input arguments in the tests.

Complex specifications are hard to understand, so most people won't be able to identify functional gaps and inconsistencies in such specifications.

With complex specifications, the effect of up-front analysis will be similar to what occurs when traditional requirements are handed down from analysts to developers. Instead of collaborating to build a shared understanding, developers just take the requirements, leading to misunderstanding and a higher probability that functional gaps won't be identified until late in the process.

Jodie Parker's team at LMAX took preparation too far and ended up with examples that seemed complete. This made them skip the discussions—and resulted in functional gaps in specifications. Parker advises preparing "just enough" examples up front:

> Because we were all very new to the process, at first our developers said that it's not enough information to work on. Then the business analysts completely prescribed very much everything and our hands were tied. When the time came to do any development on the cards, there was no creativity, no way to do a simpler solution, because it was too prescribed.

> If you read a card and say, "OK, I completely understand that," you just go off and work, and you could have made a million and one assumptions. If you read a card and there is enough of "I'm not quite sure," it pushes you to have a conversation, drawing it out at the start of an iteration, and then talking about different implementations and their effects. The testers would then consider how this impacts tests. BAs could think about what's also coming up soon and see how that fits in. "Just enough" means that your developer, BA, and QA are standing against a board and really discussing how this needs to work.

Whether you decide to have someone work one week ahead to prepare initial examples or hold an introductory meeting to identify open questions, remember that the goal is to prepare for the discussion later, not replace it.

Choosing a collaboration model

I don't think there's a one-size-fits-all heuristic that will help you choose the best model for your team, including the balance between individual up-front work and more hands-on collaboration. After comparing the teams who had similar processes, I suggest basing your decision on the following criteria:

- How mature is the product?
- How much domain knowledge is there in the team?
- How much analysis do typical changes require?
- How close are the business users to the development team? Are they readily available to discuss and verify examples?
- Where's the bottleneck in the process?

Immature products require big workshops and lots of up-front analysis. With immature products, it's important to get testers and developers to contribute to specifications more actively, because the underlying system is changing frequently and they have insights that the business users won't have.

Mature products might allow for less analysis up front and other models of collaboration. A mature product probably means that there will be few surprises. Business analysts and product owners most likely have a good idea of what the technology can give them, and they can do a good job of preparing examples up front.

If the team is relatively new or if testers and developers don't have a solid understanding of the business domain, it's worth running big workshops. All-hands workshops are a great way to efficiently transfer the knowledge about the business domain to the entire team. Once the team understands the business domain better, smaller and more focused discussions might be sufficient.

If typical changes require a lot of analysis, then someone in an analyst role should work ahead of the team to prepare detailed examples with stakeholders. Otherwise, any discussion during the workshops will end quickly and with too many open questions. If relatively small and well-understood features normally come into development, preparing some basic examples up front to make the discussion run more smoothly might be sufficient.

Teams with remote business users typically have to do more work up front than those with business users who are readily available to answer open questions. If the business users aren't available for specification workshops at all, most questions and functional gaps have to be identified and addressed up front.

Finally, there's no point in overloading team members who are already a bottleneck in the process. The teams where testing is a bottleneck should get developers and business analysts much more engaged in up-front work. Likewise, the teams where business analysts or subject matter experts are the bottleneck should get testers to help with up-front analysis.

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.