Gojko Adzic

Specification by Example

How successful teams deliver the *right* software

MANNING

Table of Contents

Chapter 4. Initiating the changes. Specification by Example: How successful teams deliver the right software, ISBN: 9781617290084

4

Initiating the changes

Many ideas central to Specification by Example have been around for decades. In the late 80s, Gerald Weinberg and Donald Gause wrote about communication problems with software requirements in *Exploring Requirements: Quality Before Design.*[1] The authors suggested that the best way to check for completeness and consistency of requirements is to design black-box tests against them—effectively suggesting the duality of specifications and tests in Specification by Example. In 1986, the German army used what later became the V Model to describe ways to build acceptance tests before implementation for validation. Today, we use the same method but refer to acceptance tests as specifications with examples. Ward Cunningham applied the practices of illustrating using examples and automating validation without changing specifications on the WyCASH+ project in 1989.[2]

Unfortunately, these ideas didn't catch on at the time. Long development phases made them impractical to execute. People spent months trying to write abstract requirements for projects that would last years. Detailing everything upfront with examples would delay that even longer.

Agile development changed the way the industry thinks about software delivery phases—and shortened these phases significantly. This made Specification by Example feasible. Iteration and flow-based projects can benefit greatly from Specification by Example. With so little time to complete a delivery phase, we need to eliminate as much unnecessary work as possible. Common problems that require fixing are rework, duplicated tasks caused by miscommunication, time wasted working back from code in order to understand the system, and time spent repeatedly executing the same tests manually.

[1] Gerald M. Weinberg and Donald C. Gause, *Exploring Requirements: Quality Before Design* (Dorset House Publishing Company, 1989).
[2] http://fit.c2.com/wiki.cgi?FrameworkHistory

Chapter 4. Initiating the changes. Specification by Example: How successful teams deliver the right software, ISBN: 9781617290084

Effective delivery with short iterations or in constant flow requires removing as many expected obstacles as possible so that unexpected issues can be addressed. Adam Geras puts this more eloquently: "Quality is about being prepared for the usual so you have time to tackle the unusual." Living documentation simply makes common problems go away.

Specification by Example is the solution: a means for dealing with the usual so that we have more time to deal with the unusual within the few days or weeks of a software delivery cycle. Living documentation is now a requirement for success.

In this chapter, we'll look at how to begin changing process and team culture so you can implement Specification by Example. We'll review three team case studies that represent different ways to integrate collaboration on specifications into iterations and flow development. Finally, I present useful ideas for fitting this process into development environments that require sign-off and traceability on requirements.

How to begin changing the process

Starting a process change is never easy, especially if you're trying to fundamentally change how team members collaborate. To get over the initial resistance and build a case for further changes, most teams started by implementing a practice that improved product quality or saved time over the short term. The most common starting points were these:

- If there's already a process change going on, use it to implement key ideas of Specification by Example.
- Use the ideas of Specification by Example as inspiration for improving product quality.
- Implement functional test automation, for teams that don't have automated functional tests.
- Introduce automated executable specifications, for teams that have test automation separate from development.
- Use test-driven development (TDD) as a stepping-stone, for teams that practice it.

All of these starting points will produce benefits in the short term and lead to further improvements.

Chapter 4. Initiating the changes. Specification by Example: How successful teams deliver the right software, ISBN: 9781617290084

Implement Specification by Example as part of a wider process change

When: On greenfield projects

Four teams I interviewed implemented the core ideas of Specification by Example when moving to an agile software development process. There was no need to fight resistance to process changes or obtain management support.

Implementing Scrum, XP, or any other agile process is a shock therapy anyway, so if you can, you might as well try to implement Specification by Example at the same time.

Teams that were able to do this reported fewer problems and implemented the process more quickly than teams that started from a dysfunctional Scrum environment. This is most likely because all four of these teams had significant support as part of their agile migration (three had consultants on site, and the fourth had a team member with prior exposure to Specification by Example).

Focus on improving quality

Instead of focusing on a particular target process, the team at uSwitch (see chapter 12) decided to focus on improving product quality. They asked all members to present suggestions for improvement and used these as inspiration. They ended up implementing most of the process patterns of Specification by Example with little resistance.

From a management perspective, this is a particularly good approach if individuals on the team are likely to resist a process change. People might complain against Scrum, agile, Specification by Example, Kanban, or anything else that's process related. An open initiative to improve quality is less likely to cause complaints. David Anderson advocates focusing on quality in Kanban[3] as the first step of his recipe for success.

Start by identifying the biggest obstacle to delivering high-quality software; then solve it.

If developers and testers aren't working closely together and have a difference of opinion about whether something is of acceptable quality, it might be useful to visualize activities

[3] David Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business* (Blue Hole Press, 2010).

related to releasing a product. At LMAX, an electronic trading exchange, Jodie Parker made all the release activities visible by creating a release candidate board, which showed a high-level view of the three teams' progress. It featured the status of all the planned deliverables, the main focus of the release, a set of tasks that would have to be done before a release, and critical issues that would have to be addressed before a release. All the teams could see this information and then come up with suggestions for improving delivery flow.

Start with functional test automation

When: Applying to an existing project

The majority of teams I interviewed adopted Specification by Example by starting with functional test automation and then gradually moving from testing after development to using executable specifications to guide development. That seems to be the path of least resistance for projects that already have a lot of code and that require testers to run their verifications manually.

Several teams sought to solve the problem of bottlenecks at the testing phase, a result of testers having to constantly catch up with development. With short delivery cycles (weeks or even days), extensive manual testing is impossible. Testing then piles up at the end of one iteration and spills over into the next, disrupting flow. Functional test automation removes the bottleneck and engages testers with developers, motivating them to participate in the process change. Markus Gärtner says:

> For a tester who comes from "testing is the bottleneck" and continuous fighting against development changes, it was very, very, very appealing to provide valuable feedback even before a bug was fixed with automated tests. This is a motivating vision to work toward.

If you don't already have functional test automation, know that this is a low-hanging fruit—an easy way to start the journey to Specification by Example that provides immediate benefits.

Automating functional tests works well as a first phase of adoption of Specification by Example, for several reasons:

- It brings immediate benefit. With automated tests, the length of the testing phase is significantly reduced, as is the number of issues escaping to production.
- Effective test automation requires a collaboration of developers and testers and starts to break down the divide between these two groups.

- Legacy products rarely have a design that supports easy testing. Starting with functional test automation forces the team to address this and make the architecture more testable, as well as sort out any issues around test reliability and test environments. This prepares the ground for automating executable specifications later.
- When most testing is manual and the team works in short cycles, testers are often a bottleneck in the process. This makes it virtually impossible for them to engage in anything else. Test automation gives them time to participate in specification workshops and start looking at other activities, such as exploratory testing.
- Initial test automation enables the team to run more tests, and run them more frequently, than manual testing. This often flushes out bugs and inconsistencies, and the sudden increase in visibility helps business stakeholders see the value of test automation.
- Writing and initially automating functional tests often requires involvement of business users, who have to decide whether an inconsistency is a bug or the way the system should work. This leads to collaboration among testers, developers, and business users. It also requires the team to find ways to automate tests so business users can understand them, preparing the way for executable specifications.
- Faster feedback helps developers see the value of test automation.
- Automating functional tests helps team members understand the tools required to automate executable specifications.

Isn't this just shifting work?

A common objection to freeing up testers by getting programmers to collaborate on test automation is that programmers will have more work and this will slow down delivery of functionality. In fact, the general trend in the industry is for teams to have more programmers than testers, so moving work from testers to developers is not necessarily bad—it might remove a bottleneck in your process.

Implementing functional test automation will get the team to work closer and prepare the system for the use of executable specifications later. To get the most out of this approach, automate functional tests using a tool for executable specifications and design them well, using the ideas from chapter 9 and chapter 11. Automating tests using traditional tester record-and-replay tools won't give you the benefits you need.

Start by automating high-risk parts of the system

Working to cover all of a legacy system with automated tests is a futile effort. If you use functional test automation as a step to Specification by Example, develop enough tests to show the value of test automation and get used to the tools. After that, start implementing executable specifications for changes as they come and build up test coverage gradually.

To get the most out of initial functional test automation, focus on automating the parts of the system that are risky; these are the areas where problems can cost you a lot of money. Preventing issues there will demonstrate immediate value. Good functional test coverage will give your team more confidence. The benefits from automating parts with less risk are probably not as noteworthy.[†]

[†] See http://gojko.net/2011/02/08/test-automation-strategy-for-legacy-systems

Introduce a tool for executable specifications

When: Testers own test automation

On projects that already have functional test automation fully owned by testers, a key challenge is to break the imaginary wall between testers and developers. There's no need to prove the value of test automation or to flush out issues around test environments, but the mindset of the team has to become more collaborative.

This problem is mostly cultural (more on that soon), but sometimes it's also financial. With an expensive test automation framework such as QTP, licensed per seat, developers and business analysts are intentionally kept far from the tests. Once the attitude toward collaboration changes, a team can work together on specifications and automate the validation without changing them.

Several teams got pushed in this direction when they had a problem that couldn't be appropriately tested with their existing automation toolkit. They used this situation as a good excuse to start using one of the automation tools for executable specifications. (See the "Tools" section in the appendix for examples or additional articles on tools at http://specificationbyexample.com)

Teams discovered that using an automation tool for executable specifications got developers more engaged in test automation and provided business users with a greater understanding of the tests.

Developers then became more engaged in test automation and began running tests on their machines; they were able to see the value of quick feedback from functional tests. Business users could understand automated tests using a tool for executable specifications and participate in specifying related acceptance criteria. After that, changing the process to design executable specifications and tests up front was relatively simple.

When working with a large insurance provider, Rob Park's team used a proof of insurance PDF as an excuse to introduce a tool for automating executable specifications and move functional testing to an earlier stage in the development cycle. Park says:

> QTP wasn't able to test it—it could verify that the window would pop up without an error message, but that's it. I wanted to be able to have the developers run tests in the first place on their machines, which was one of the limitations of QTP [due to cost per seat]. I went with JBehave. We kind of threw it all in all at once, and it really just took off in a week. We can now let these acceptance tests themselves drive the design of the underlying controller.

At Weyerhaeuser, Pierre Veragen and the team worked with a custom test automation tool that recorded tests through the user interface. The cost of maintenance was high. After a change that broke many tests, he was able to justify moving to FitNesse after estimating that rewriting existing tests in the new tool would take less time than re-recording all the broken tests. Moving to FitNesse allowed the team to work more closely with engineers on executable specifications and sparked the move to Specification by Example.

Use test-driven development as a stepping stone

When: Developers have a good understanding of TDD

Another common strategy for adopting Specification by Example is to grow the process from (unit) test-driven development, especially when working on a greenfield project.

Test-driven development practices are much better documented and understood in the community than Specification by Example. If a team already has good TDD practices in place, there's probably no need to demonstrate the value of automated tests or change the design to make their software more testable. Executable specifications can be seen as an extension of test-driven development to business rules. (The term *acceptance-test-driven development* is a popular synonym for Specification by Example.)

At ePlan Services, Lisa Crispin used this approach when they were first implementing Specification by Example:

> I couldn't get people interested in acceptance tests. On Mike Cohn's suggestion, I just picked a story, went to the developer working on a story, and asked, "Could we pair up writing a test on this story?" Developers would see how easy it is. In the next sprint I picked a different story and a different person. We right away found a bug, where he didn't really understand the requirement. So the developers immediately saw the value.

When a team has a good understanding of TDD, making the case for executable specifications is easy: Explain them as tests for business functionality.

How to begin changing the team culture

For the most part, implementing Specification by Example involves a cultural change—getting people to think about collaborating on requirements and changing the way business people, developers, and testers contribute to specifications. Here are some helpful ideas for changing your team culture.

Avoid "agile" terminology

When: Working in an environment that's resistant to change

Agile software development methods are plagued with terminology and buzzwords. *Scrums*, *stand-ups*, *user stories*, *backlogs*, *masters*, *pair programming*, and other terms like these are easily misunderstood and cause confusion. To some, they can even be overwhelming and scary. Anxiety caused by jargon is one of the biggest reasons why people push back and resist any change to their process—or passively wait for it to fail. In my experience, many business users find it hard to understand technical terms used by the development team, making it hard for them to grasp ideas related to process improvement and engage with the team.

It's entirely possible to implement Specification by Example without using technical terminology. If you work in an environment that's resistant to change, avoid jargon when you start out.

Don't refer to user stories, acceptance tests, or executable specifications—implement Specification by Example without offering definitions. This will provide people who want to oppose you with less ammunition. Explain Specification by Example as the process of gathering examples to clarify requirements, deriving tests, and automating them. Everything else will be left to discovery.

Adam Knight implemented most of the key elements of Specification by Example at RainStor without making a big deal about it. The process grew without up-front planning, and Knight says that nobody else in the company knows about Specification by Example. "People aren't really aware of anything specific," he said. For his team, it's just a process they created themselves.

Pierre Veragen used a similar approach to improve one team's software process at Weyerhaeuser. The team maintains a legacy application with over a million lines of code. They were resistant to implementing anything with the name agile. Without using any big words, Veragen suggested automating tests below the user interface to make testing more efficient. When the team got their heads around that, he then suggested that developers start running tests on their machines to get faster feedback and align testing and development. With some hand holding and monitoring, Veragen ultimately got the team members to stop thinking about testing as something that comes after development. This took about six months, mostly because the automated test suite had to become big enough for developers to start seeing test failures when they introduced problems into the code. Veragen commented:

> People working in engineering realized that failing tests on their machine were actually pointing to problems in their code. When this happened to a developer, he got the idea and stopped questioning why he had to run the tests.

To implement a process change without technical terminology, make the problems visible and gently push people in the right direction to solve them. When the team comes up with a solution to a problem, even with some help, they'll have a sense of ownership and commit to following through on process changes.

Ensure you have management support

Most teams significantly changed the way they worked while implementing Specification by Example. For many, this meant changing the way they approached specifications, development, and testing—and learning how to collaborate better within a team and with external stakeholders.

Many people got confused when roles started changing. Testers had to get much more involved in analysis. Developers had to get more involved in testing. Analysts had to change the way they collected and communicated requirements. Business users had to take a much more active role in preparing specifications. Such big changes require management support; otherwise, they're destined to fail. Clare McLennan had this to say:

> You need to get the management support for the project, especially if you have an existing system already, because it's going to take quite a bit of time to get to a point where you actually have a good, stable test system. You've got to go through all the iterations, see where it's not stable or where it gives you strange answers, fix that, and repeat again. If you spend time over a year or so, you'll get an invaluable system. If you don't, or if you think it's a quick fix to get UI tests with click, click, click, then you'll get something unmaintainable and with little value.

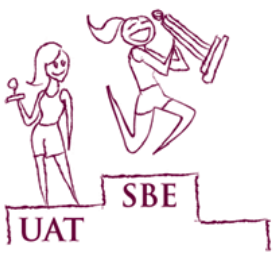
At the beginning, automation of executable specifications was a challenge for many teams, because it's conceptually different from what both testers and developers were used to when automating tests (more on this in chapter 9). Teams had to learn how to use a new tool, find a good way to design executable specifications, and structure their living documentation. For the first several months, the productivity of the development team drops before it increases. This also requires management understanding, approval, and support.

Without management buy-in and support, the chances of success with a process change are slim.

If management responds with pressure rather than support, people will fall back into their old ways of doing things and start protecting their position rather than collaborating. Sharing the success stories and benefits outlined in chapter 1 should help with getting that support, but if that fails, it's better to look at less-ambitious ways to improve the process or take smaller steps.

Sell Specification by Example as a better way to do acceptance testing

Several teams, including the ones working in strictly regulated environments, got to the point where user acceptance testing as a phase in software delivery was no longer needed. (Some companies call this phase customer acceptance testing or business acceptance testing.) This doesn't mean that they weren't testing for user acceptance. Specifying and checking the acceptance criteria is different from user acceptance testing as a software delivery phase. It's so important that it shouldn't be left to the end. Executable specifications and frequent validation make development teams check for user acceptance continuously. The product doesn't get delivered to the users unless all their acceptance tests pass.



Once the executable specifications are comprehensive enough and validated frequently, the trust between a development team and its customers increases to the level that verifying software manually after delivery becomes unnecessary. (This doesn't mean that testers shouldn't perform exploratory testing before delivery.)

I expect that most teams will be able to justify the cost of implementing Specification by Example on the basis of avoiding late acceptance testing. Changing the process so that a team can get there faster should have measurable financial benefits, which can then justify an investment in process change.

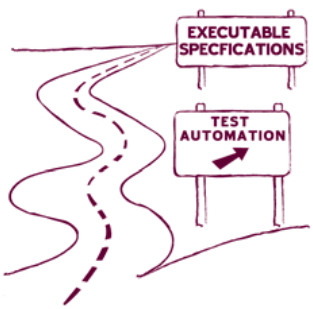
Short iterations or flow-based development significantly increase the frequency of potential releases. Let's say that you want to have 12 releases over the next 12 months (most of the teams I interviewed would do twice that figure), and that the user acceptance testing takes an average of 3 days. This means that over the next year you'll spend 36 days in user acceptance testing, assuming the best-case scenario: You never catch any problems and software is always accepted (in which case, why test it for 3 days?). More realistically, acceptance testing at the end, rework, and retesting will take at least 2 months over a 12-month period.

If you begin by collaborating to specify the acceptance criteria and automate the validation, you won't have to waste time with manual testing and rework. There's a cost to pay for the automation, but Specification by Example can reduce time-to-market significantly.

Specification by Example has many other benefits, but this is the easiest one to present to business stakeholders and the easiest to quantify. If you need to sell this process change to your business stakeholders, try selling it as a way to get to the market two months earlier every year.

Don't make test automation the end goal

One of the most common early mistakes made by the teams I interviewed was setting functional test automation as the end goal of the process change. Business users generally think of functional test automation as something to do with testing and hence something they don't need to get involved in. Unless developers understand that automated tests need to be human readable to improve communication, they'll automate them in a way that minimizes development effort.



When teams focused only on test automation, they didn't get better collaboration.

This approach often leads to tests that are too technical and that are scripts rather than specifications, a common failure pattern (see "Scripts aren't specifications" in chapter 8). In the long term, tests automated in this way became an impediment to change, not a facilitator.

If you use functional test automation as a step toward Specification by Example, make sure that everyone on the team is aware of the end goal. When functional test automation takes hold, it's time to move on.

Don't focus on a tool

Three people I interviewed started the journey by selecting a tool they wanted to use. Developers had heard about FitNesse or Cucumber and decided to try it out on their project. I've been guilty of that myself as well; this approach has a small chance of success.

Specification by Example isn't programmer centric, and programmers using a tool in isolation aren't going to get far.

This approach often ends up with programmers using a nontechnical tool, intended for executable specifications, to manage technical, developer-oriented tests. This is a waste of time.

Of the three cases where developers focused on using a particular tool, only Ian Cooper's team at Beazley succeeded in creating a good process. They pushed hard to involve testers and business analysts and then adjusted the way they wrote and organized tests to support that. They were also critical of the benefits they were getting from the tool and looked for easier ways to get those benefits.

In the other two cases, teams focused on a tool, not on high-level collaboration and process changes. They ended up wasting a lot of time and effort building a suite of technical tests that business users and testers weren't able to use. They paid a big price in terms of effort and time spent on test maintenance without any of the benefits of Specification by Example.

Keep one person on legacy scripts during migration

When: Introducing functional automation to legacy systems

Rewriting functional tests and automating them using a new tool takes time. Until the new validation system grows, any existing tests might need to be maintained and kept up-to-date. A good way to address this issue is to delegate that work to a single person and write off that person's time while planning for the immediate future.

James Shore and Shane Warden describe a pattern for process change on legacy projects called "the Batman" in *The Art of Agile Development*.[4] The Batman is a dedicated person who jumps in to solve urgent issues and resolve important bugs, while the rest of the team keeps on working on new functionality. Markus Gärtner used this approach to gradually move a set of tests to an automation tool for executable specifications. He detailed his experience:

> When we transitioned our tests from shell-based scripts to tests based on FitNesse, we started with two members working on the new stuff, while three team members maintained the old legacy test scripts. Over time, we got more and more testers involved in the new approach. First three, then another one, and finally we were able to throw the old scripts away completely.
>
> The underlying idea was the Batman, who dove in to solve problems. I remember that some of my colleagues even bought a toy car from Hot Wheels—the Batmobile—and gave that to our Batman at that time. Initially, I had the idea to rotate the Batman but never applied this, since my colleagues were knowledge silos at that time. We changed that with the new approach, and I try to rotate the Batman role so that everyone gets to deal with the old stuff and the new stuff during the transition. Getting everyone's buy-in is crucial.

By delegating the work required to update legacy items to a single person, teams are able to move more quickly toward the goal of migrating to a new process.

This idea is similar to Alistair Cockburn's "Sacrifice One Person" strategy,[5] where one person is assigned to handle distracting tasks and the rest of the team moves forward at full speed.

---

[4] James Shore and Shane Warden, *The Art of Agile Development* (O'Reilly Media, 2007).
[5] http://alistair.cockburn.us/Sacrifice+one+person+strategy

Track who is running—and not running—automated checks

When: Developers are reluctant to participate

In cases where developers came from a more structured process background—where programmers write code and testers test it—teams had problems getting programmers to participate in the process. This has to change for Specification by Example to work.

Pierre Veragen had a unique solution to get programmers involved in the process. He created a simple centralized reporting system that told him when and where the executable specifications were checked:

> In the fixture code I put a little thing that told me when people were running tests on their machines. That group was pretty shy. I used this to find out when people aren't running the tests to go and talk to them and see what's wrong and whether they had any problems. The idea was to get a more objective feedback than "Yes, it worked fine."

By tracking who isn't running tests before committing, he was able to focus his efforts on the team members who had issues or required assistance. Veragen says that all the programmers knew about the process from the start, so that instead of monitoring the actual test results, he tracked only whether someone executed tests.

Monitor when tests are being run, so programmers will run automated checks.

This is an interesting approach for larger teams, where a coach cannot work with all the members all the time. I expect the effects to be similar to publishing the locations of speed cameras on highways—programmers will know that someone is looking, and they'll take more care to run the checks.

How teams integrated collaboration into flows and iterations

Understanding how to fit collaboration into a delivery cycle is one of the biggest challenges when teams start implementing Specification by Example.

Differences between Specification by Example and Waterfall analysis

Many people I meet at conferences misunderstand and think that incrementally building up a documentation system means going back to the Waterfall ideas of big up-front analysis. During his presentation "How to Sell BDD to the business"[†] in November 2009, Dan North said that BDD is effectively V-Model compressed into two weeks. This isn't an entirely accurate description, but it's a good start.

There are a few crucial differences between the Waterfall analysis approach and what Specification by Example is trying to achieve. It's important to understand these underlying principles because that will help you fit the practices into your process, whatever it is. These are the key factors differentiating Specification by Example from more established analysis:

- Provide fast feedback and focus through quick turnaround; get small chunks of software done efficiently, instead of trying to handle large chunks at once.
- Emphasize effective, efficient communication instead of long, boring documents.
- Establish shared ownership so specifications aren't handed over or thrown over imaginary walls to become code or tests.
- Integrate cross-functional teams where testers, analysts, and developers work together to build the right specification of the system instead of working alone.

[†] http://skillsmatter.com/podcast/agile-testing/how-to-sell-bdd-to-the-business

There's no universal solution for a process change, and each team will need to decide how best to extend their way of delivering software. Following are some nice representative examples that will help you get started. I've chosen three good case studies, one for each popular process. The Global Talent Management team works according to a flow-based Kanban framework. The Sierra team delivers software using an Extreme Programming iteration-based process. The Sky Network Services group runs iterations based on Scrum.

Global Talent Management team at Ultimate Software

The Global Talent Management team at Ultimate Software is one of 16 teams working on their human resources management system. The team consists of a product owner, a user experience expert, four testers, and ten developers. When I interviewed Scott Berger and Maykel Suarez, who were part of this team, their project had been under way for eight months. The team uses a Kanban flow process.

Because the product analyst (a combination of an analyst and a product owner) is busy, the team tries to use his time efficiently when collaborating on specifications. The product analyst explains a story at a high level with "story points" (in their case, story points aren't estimates of complexity but bullet-list items that explain stories). They avoid technology-specific language as much as possible when writing story points. The story is then added to the Kanban board as part of the backlog.

In a daily meeting, limited to 30 minutes, the lead engineers meet with the product analyst and anyone else on the team who is interested in the backlog. They quickly run through the stories in the backlog, checking that each story is sliced correctly, that it makes sense, and that it's implementable in four days or less—as a team, they imposed this deadline for each story. During the meeting, they also clarify any open questions and look at story dependencies.

The story then goes into a queue for writing specifications with examples, which are used as acceptance tests. A team member who will work on implementing the story pairs up with someone who will focus on the testing to write the outlines of these specifications. (They don't have formal tester or developer roles in the team, but I'll use those roles to refer to this pair of people in this section to make it clearer.) Berger explained:

> By leveraging this pair, we're able to cut down on the necessary tests required for this story, because a developer has more intimate knowledge of the code. This has proven quite successful, because these pairs generally root out any inconsistencies and design flaws that exist and aren't caught in the initial story review.

After they define an outline, the tester usually completes the scenarios in a Given-When-Then format. The pair and the product analyst meet and review the scenarios in an in-depth Story Knowledge and Information Transfer (SKIT) session. This addresses the risk that a programmer and a tester won't be able to come up with good specifications without an analyst. When the product analyst approves the scenarios, the team regards them as the requirements. Other than minor text alterations, no further changes are allowed to these requirements until the story is delivered.

The developer will then automate the scenarios, usually prior to implementing production code. This allows the tester to spend more time completing exploratory testing. The tester might also work on automation, but that's no longer his focus. Berger said that this collaboration allows them to work productively:

> Developers who intimately know the code are much quicker in terms of automating, and actually write code that allows them to query their objects (instead of automating through the user interface), and derive a greater benefit while developing, since many more error conditions and combinations are explicitly described. Since we are now testing under the GUI, our test execution is much quicker and more stable. Testers are free to spend more time exercising the code and providing feedback.

All the executable specifications have to pass for the development stage to finish. All the tests are executed again during the run-tests phase, this time integrated with the work of other teams. While waiting for approval phase, the team does a quick product demo for the product analyst and obtains a sign-off.

According to Berger, this process leads to results that are of exceptionally high quality:

> By working closely with our product analysts, and using our tests as a basis for requirements, we are able to achieve an exceptionally high level of quality. The business has collected metrics, and one specifically that I think adequately provides an insight into our efforts in terms of our commitment to quality is Defect Detection Efficiency (DDE). Upon measuring the Global Talent Management team, it was determined that our DDE is 99% [for Q1-Q3 2010]!

Sierra team at BNP Paribas

The Sierra team at BNP Paribas builds a back-office reference data management and distribution system. The team consists of eight developers, two business analysts, and a project manager. Because there are no dedicated testers, everyone on the team is responsible for testing. Their stakeholders are off-site business users. Change requests typically require a lot of analysis and work with the stakeholders.

The project has been under way for about five years, so it's reasonably mature, and business analysts have many existing executable specifications to use as examples. Andrew Jackman, who was a member of this team when I interviewed him, says this is one of the rare examples in the financial services industry where Extreme Programming is applied almost by the book.

Their development process starts with the project manager, who selects the stories for an iteration in advance. The business analysts work with remote stakeholders to prepare the acceptance criteria in detail before an iteration starts. They use examples of existing specifications to drive structure of the new ones. If the new specifications are significantly different from any existing ones, a pair of developers will review the tests to provide early feedback and ensure the tests can be automated.

Their iterations start every second Wednesday. When an iteration starts, the whole team gets together for a planning meeting and reviews the upcoming stories in the order of priority. The goal is to ensure that all the developers understand what the story is about, to estimate the stories, and to check for technical dependencies that would make a different order of delivery better. They'll also break down the story into tasks for development. By the time a story is reviewed in a planning meeting, the acceptance criteria for that story are generally already well specified.

The team occasionally discovers that a story wasn't well understood. When they were running weeklong iterations, such stories could disrupt the flow, but two-week iterations allow them to handle such cases without much interruption to the overall process.

The stories are then implemented by pairs of developers. After a pair of developers implements a story and all the related tests pass, a business analyst will spend some time doing exploratory testing. If the analyst discovers unexpected behavior, or realizes that the team has not understood all the effects of a story on the existing system, he will extend the specification with relevant examples and send the story back to the developers.

Sky Network Services

The Sky Network Services (SNS) group at the British Sky Broadcasting Company maintains a system for broadband provisioning. The group consists of six teams, and each team has five to six developers and one or two testers. The entire group shares six business analysts. Because the teams maintain separate functional components that differ in terms of maturity, each team has a slightly different process.

The entire group runs a process based on Scrum in two-week sprints. One week before a sprint officially starts, they organize a preplanning coordination meeting attended by two or three people from each team. The goal of that meeting is to prioritize the stories. By the time they meet, the business analysts have already collected and specified some high-level acceptance criteria for each story. After the meeting, testers will start writing the specifications with examples, typically collaborating with a business analyst. Before the sprint officially starts, each team will have at least one or two stories with detailed specifications with examples ready for automation.

The iterations start every second Wednesday, with a cross-team planning meeting to inform everyone about overall progress and the business goals of the upcoming iteration. The teams then go into individual planning meetings. Some teams spend 15 minutes going through the stories briefly; others spend a few hours going over the details. Rakesh Patel, a developer who works on the project, says that this is mostly driven by the maturity of the underlying component:

> At the moment we're working on a component that's been there for a long time and adding more messages to it. There is not really much need for the whole team to know what that involves; we can wait until we pick up the story card. Some other teams are building a new GUI, completely new functionality, and then it might be more appropriate for the whole team to sit down and discuss this in depth and try to work out what needs to be done. At that point we might also discuss nonfunctional requirements, architecture, etc.

After the planning meeting, the developers start working on the stories that already have specifications with examples. Business analysts and testers work on completing the acceptance criteria for all the stories planned for the iteration. Once they finish the specifications for a story, they'll meet with a developer designated to be the "story champion" (see the sidebar) and go through the tests. If everyone thinks they have enough information to complete a story, the story card gets a blue sticker, which means that it's ready for development.

Once development is completed, the business analysts will review the specifications again and put a red sticker on the card. Testers will then run additional tests on the story and add a green sticker when the tests pass. Some stories will involve their support team, database administrators, or system administrators, who review the story after the testers. Database administrators put a gold star on the story card, and system administrators add a silver star when they've reviewed the results. Stickers ensure that everyone who needs to be involved in a story knows about it.

Instead of big specification meetings, the teams at SNS organize a flow process. They still have a two-phase specification process, with business analysts and testers preparing all the examples upfront and then reviewing them with a developer later. This gives developers more time to focus on development work. Instead of involving all the developers in the review to ensure they understand a story, the story champion is effectively responsible for transferring the information while pairing with other developers.

The previous three examples show how teams can fit collaboration into short iterations and even flow-based processes, demonstrating that there's no generic and universally applicable approach to structuring the process. All the teams successfully integrated collaboration into their short release cycles but used different approaches depending on the team structure, availability of their business users, and complexity of changes coming into the delivery pipeline. For some nice ideas on designing a process to fit your team, see "Choosing a collaboration model" in chapter 6. For more examples, see the case studies in part 3.

Story champion

SNS teams use story champions to ensure an efficient transfer of knowledge while switching pairs of developers who work on a story. Kumaran Sivapathasuntharam, a business analyst working on the project, explains this idea:

> The story is assigned to a particular developer who stays with the story until it's completed. This ensures that there's one point of contact for a story—so if you have an issue with a story, you can talk to the story champion. One person can stay on the story from start to finish so an entire pair doesn't get stuck on it, and they can keep changing the pairs but still having the continuity throughout.

The Global Talent Management team at Ultimate Software has a similar role, called story sponsor. According to Maykel Suarez, the sponsor is responsible for communication with other teams, tracking the progress on the Kanban board, checking status at stand-up meetings, and eliminating roadblocks.

Dealing with sign-off and traceability

For some teams, a big problem with little or no documentation on agile projects is the lack of requirements. This makes sign-offs on requirements or deliverables difficult. As a whole, the software development industry is much less concerned with sign-offs than it was 10 years ago. In some cases, sign-offs are still required because of regulatory constraints or commercial arrangements.

Specification by Example provides artifacts around requirements—living documentation—that can be used for traceability. This enables agile processes to be applied to regulated industries. Bas Vodde and Craig Larman wrote about the first agile development project in the U.S. nuclear industry[6] in *Practices for Scaling Lean and Agile*.[7] The team working on that project used executable specifications to ensure full traceability of requirements, which is critical in nuclear and other safety-critical domains. On the other hand, because of a highly dynamic, iterative, and collaborative approach to building these artifacts, up-front sign-off is practically impossible. Here are some ideas on how to deal with sign-off and traceability constraints.

Keep executable specifications in a version control system

Several people I interviewed said that keeping executable specifications in the same version control system as the product source code is one of the most important practices for a successful process implementation.

Many automation tools work with executable specifications in plain-text files, so they work nicely with version control systems. This allows you to easily tag and branch the specifications along with the source code. It gives you a current and correct version of tests that can be used to validate any version of your product.

Version control systems are great for traceability because they allow you to find who changed any file, when, and why in an instant. If you store your executable specifications in a version control system, you'll get traceability on requirements and specifications for free. With Specification by Example, the executable specifications will be directly linked to the programming language code (through the automation layer), which means that it will be relatively straightforward to prove code traceability as well.

Executable specifications in a version control system are also less likely to disappear than those stored in a separate requirements or test tool.

Get sign-off on exported living documentation

When: Signing off iteration by iteration

Specification by Example should help build trust between project sponsors and the delivery team and remove the need for sign-off. If you do need to get requirements signed off for commercial or political reasons, you can use the living documentation system for that.

[6] I would have loved to include that case study in this book as well, but unfortunately I couldn't get in touch with anyone willing to talk about it.

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

If you need to get sign-off on specifications before starting to implement functionality, and it's possible to organize this for each iteration, then you can create a Word or a PDF document from the executable specifications planned for the next iteration and get sign-off on that.

Some automation tools, such as Cucumber, support exporting to PDF directly, and this might help you with the process.

Get sign-off on scope, not specifications

When: Signing off longer milestones

If you need to get sign-off on batches of software larger than what a single iteration can deliver, try to get sign-off on scope and not on detailed specifications. For example, obtain sign-off on user stories or use cases.

Rob Park applied this approach when working with a large U.S. insurance provider. His team kept a Waterfall approval process for sign-off but significantly cut down on the material that required it. Park explained:

> There is kind of a bigger process outside of everything we have under our control. The business analysts work on Word documents using a template, but they cut the template down from eight pages to two. There is a story card approval process where the people who pay for the project actually sign off on these story cards before anyone except the business analyst actually sees them. So they have this Waterfall process in the company at a high level, but once it gets into the team it becomes different.

Word documents are used in this case purely because contractual obligations dictate that there be paperwork before a story makes it into development. The team uses executable specifications as their only source of requirements after the scope is approved.

Get sign-off on "slimmed down use cases"

When: Regulatory sign-off requires details

Getting sign-off on scope might not work in a heavily regulated environment. Mike Vogel of Knowledgent Group worked on a project for the pharmaceutical industry using a process based on Scrum and XP extended to satisfy regulatory requirements. His team used use cases, because the standards of a regulated system can't be met with user stories alone.

The team used slimmed-down use cases (they called them "structured stories") so that initial capture and ongoing evolution wasn't a big problem. Those use cases would avoid most details about data and decisions (these were extracted into separate data sections). Vogel explained that approach:

> In a use case you would have a nickname for a piece of data, something that the customer understands as part of the domain language. The data section describing that gives the structure and the rules describing the data—not examples. Examples are in the [acceptance] tests/requirements-by-example where we walk the use case building examples that cover and show variations of all our named chunks of data. You vary your examples of the chunks of data that they have factored out.

Get sign-off on "lighter" use cases—without examples.

Vogel's team built a requirements document with those lightweight use cases but without any examples. The result was a document less than 100 pages long for a large project "with all the regulatory required boilerplate," according to Vogel. Throughout the project, they collaborated with the customer to specify the use cases and examples:

> We sit with the customer in the team room and try to work out one use case at a time along with the examples. Discussions are about detailed examples. We end up putting in some details after that and the customer reviews it.

This approach allows the team to get a sign-off on something that closely resembles traditional specifications without overprescribing them. Details come from an iterative and collaborative process; the backlog the customer works with is based on the high-level use cases and the details come later.

Introduce use case realizations

When: All details are required for sign-off

Matthew Steer worked on several projects based on a structured process (Rational Unified Process). The process required full sign-off on all the details, and specifications were captured as use cases. In addition to use cases, Steer and his team introduced use case realizations that effectively illustrated use cases with examples. This allowed them to use Specification by Example in a structured process. Steer says:

> "The requirements were captured as use cases and supplementary specifications for nonfunctional requirements—quite traditional, by-the-book capture. With use cases we produced use case realizations, examples and scenarios that use cases would fulfill. We created tables with lots of parameters and hooked up data from there and then had flows to show how the use case would be realized. Use case realization was a working version of what that meant to the business, using real scenarios."

Adding details such as use case realizations is a good idea to get Specification by Example into a formal process—under the radar of the methodology police. It can also help to implement the ideas of Specification by Example when commercial contracts require sign-off on requirements but still allow for later variability in detail.

Steer's team, as well as others mentioned in the previous section, used examples—even if disguised as use-case realizations—instead of only using use cases or tests made against more generic requirements. This made their delivery process more effective.

Technically, a living documentation system instantly provides traceability in requirements changes, because teams use version control systems to store their executable specifications. Iterative development is generally at odds with up-front sign-offs, but you can use the tips from this section to deal with that while the process changes and the delivery team gains trust from the business users. The visibility that a living documentation system provides, along with collaboration on specifications, should help to eliminate the need for sign-offs.

Warning signs

You can track your progress to check if you're implementing Specification by Example properly. As with any metrics, make sure that metrics themselves don't become a goal; otherwise, you'll locally optimize a process to get the numbers right but hurt the long-term results. Use these as measurements to check if the process needs adjustment.

Watch out for tests that change frequently

At XPDay 2009, Mark Striebeck talked about what Google is doing to advance its testing practices.[8] One of the ideas that impressed me was how they measure whether a (unit) test is good or not. When a test breaks, they track changes in the source code until the test starts passing again. If the underlying code was changed, they consider the test to be a good one. If the test changed and the code did not change, they consider it to be bad. By collecting these statistics, they hope to analyze unit test patterns and identify what makes a test good or bad.

I believe that the same criteria can be applied to executable specifications. If a validation fails and you change the code, that means you found and fixed a problem. If a validation fails and you have to change the specification, that means it wasn't written properly.

Business rules should be much more stable than the technology that implements them. Watch out for executable specifications that change frequently. Look for ways to write them better.

You can also measure the time your team spends on refactoring specifications and related automation code in order to keep it under control. If you spend a significant portion of your iteration doing this, look for better ways to automate tests (see chapter 9 for some good tips).

Watch out for boomerangs

Another good metric to check if you're doing something wrong is checking for the presence of boomerangs. A *boomerang* is a story or a product backlog item that comes back into the process less than a month after it was released. The team thought it was done, but it needs rework. But it's not a boomerang when the business later extends existing requirements to incorporate innovation as the product evolves.

Once you implement Specification by Example, the number of boomerangs should be reduced significantly until they're a rare occurrence. Collaboration on specifications and better alignment of testing and development should eliminate wasteful rework caused by misunderstanding. Reviewing your boomerang trends over several months will show you how much you've improved. If the rate doesn't drop, it means that there's something wrong with the way you implemented the process.

Tracking boomerangs doesn't take a lot of time, usually a few minutes every iteration, but it can help a lot when the time comes to challenge or prove that Specification by Example is working. In larger companies, it can also provide compelling evidence that it's worth doing with other teams. For more complex statistics, you can also track

[8] http://gojko.net/2009/12/07/improving-testing-practices-at-google

the time spent on boomerangs, because this figure directly translates into wasted development/testing time and money. If people complain about the time spent on automating executable specifications as unnecessary overhead, compare that to the time they spent working on boomerangs several months earlier. This should be more than enough to build a business case for Specification by Example.

Once the number of boomerangs goes down and they occur relatively rarely, you can stop tracking them. If a boomerang occurs, try to understand where it's coming from. One of my clients had many boomerangs coming from their financial department. This pointed to a communication problem with that particular part of the company; as a result, they looked for better ways to engage the department.

Tracking boomerangs is also a good way to build a business case for introducing Specification by Example. It can help a team pinpoint the waste caused by vague requirements and functional gaps in specifications.

Watch out for organizational misalignment

Many teams started implementing Specification by Example as a way to better align their activities with iterations. After you become familiar with executable specifications and the automation code becomes stable, you should be able to implement a story and completely finish testing it (including manual exploratory testing) inside the same iteration. If your testers are lagging behind development, you're doing something wrong. A similar warning sign is misaligned analysis. Some teams start analysis ahead of the relevant iteration, but they still have regular intervals and flow. Analyzing too much up front, analyzing things that won't be implemented immediately, or being late with analysis when details are needed are signs that the process is wrong.

Watch out for just-in-case code

In *Lean Software Development*[9] Mary and Tom Poppendieck wrote that the biggest source of waste in software development is just-in-case code—software that was written without being needed. I'm not sure whether it's the biggest source of waste, but I've certainly seen lots of money, time, and effort wasted on things that nobody needed or asked for. Specification by Example significantly reduces this problem because it helps us build a shared understanding of what we need to deliver. Jodie Parker says that the conversations and collaboration on specifications helped her team achieve just that:

[9] Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit* (Addison-Wesley Professional, 2003).

> When the developers got a story card, they'd very much want to deliver everything within it, to make it technically as fabulous as possible, even though the steer was "do the minimal thing possible to get the value given." It's got to be efficient, but we can always bring in the stories later to refine it. This was addressed using conversations and continually working out if we're able to draw the business model that we're trying to achieve. By domain modeling you can very easily break down this into tasks. Those tasks are then the only thing you can do. Because the tasks are small, you can go off on one of them, but if you do it's very easily spotted by the rest of the team and the team would speak up. When someone's been on a task for several days, we'd have that conversation in the standup.

Watch out for people who implement more than what was agreed on and specified with examples. Another good way to avoid just-in-case code is by discussing not only what you want to deliver but also what's out of scope.

Watch out for shotgun surgery

Shotgun surgery is a classic programming antipattern (also called *code smell*) that occurs when a small change to one class requires cascading changes in several related classes. This telling sign can be applied to living documentation; if a single change in production code requires you to change many executable specifications, you're doing something wrong. Organize your living documentation so that one small change in code leads to one small change in tests (see "Listen to your living documentation" in chapter 11 for some good tips on how to do so). This is one of the key steps to reducing maintenance costs of automation over the long term.

Remember

- Specification by Example is a good way to provide development teams with just-in-time specifications, so it's a key factor for success with short iterations or flow-based development.
- Handle small chunks of software efficiently to enforce quick turnaround time and feedback.
- Emphasize effective, efficient communication instead of long, boring documents.
- Integrate cross-functional teams where testers, analysts, and developers work together to build the right specification of the system.
- Plan for automation overhead upfront.