

GOJKO ADZIC

How successful teams deliver
the right software



SPECIFICATION BY EXAMPLE

MEAP

 MANNING



**MEAP Edition
Manning Early Access Program
Specification by Example final version**

Copyright 2011 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

Introduction

Part One: Getting Started

Chapter 1: Key benefits

Chapter 2: Key process patterns

Chapter 3: Living documentation

Chapter 4: Initiating the changes

Part Two: Key Process Patterns

Chapter 5: Deriving scope from goals

Chapter 6: Specifying collaboratively

Chapter 7: Illustrating using examples

Chapter 8: Refining the specification

Chapter 9: Automating validation without changing specifications

Chapter 10: Validating frequently

Chapter 11: Evolving a documentation system

Part Three: Case Studies: How companies have implemented Specification by Example

Chapter 12: uSwitch

Chapter 13: Rainstor

Chapter 14: Iowa Student Loan

Chapter 15: Sabre Airline Solutions

Chapter 16: ePlan Services

Chapter 17: Songkick

Chapter 18: Concluding thoughts

Appendix A: Resources

Introduction

The book you hold in your hands, or on your screen, is the result of a series of studies of how teams all over the world specify, develop and deliver the right software, without defects, in very short cycles. It presents the collective knowledge of about fifty projects, ranging from public web sites to internal back-office systems.

These projects involved diverse teams, from small ones working in the same office to groups spread across different continents, working in a range of processes including Extreme Programming, Scrum, Kanban and similar methods (often bundled together under the names "Agile" and "Lean"). They have one thing in common — they all got the practices of collaborating on specifications and tests right, and they got big benefits out of that.

Specification by Example

Different teams use different names for their ways of dealing with specifications and tests, yet they all share a common set of core principles and ideas, which I hold to be essentially the same thing. Some of the names that the teams used for these practices are:

Agile acceptance testing

Acceptance-test driven development

Example-driven development

Story-testing

Behavior-driven development

Specification by example

The fact that the same thing has so many names reflects the fact that there is a lot of innovation in this field at the moment. It also reflects the fact that the practices described in this book impact the ways teams approach specifications, development and testing. To be consistent, I had to choose one name. I settled on Specification by Example and I will use that in the rest of the book. I explain this choice in detail in the "A few words on the terminology" section later in this introduction.

In the real world

I present this topic through case studies and interviews. I do it this way so that you can see that there are real teams out there right now, doing this and getting big benefits out of that. Specification by Example is not a dark art. You don't have to be a high priest of the Church of Scientology or a Freemason to implement it, although some popular media might make you think that.

Almost everything in this book is from the real world, real teams and real experiences. A very small number of practices are presented as suggestions without being backed by a case study. These are ideas that I think will be important for the future, and they are clearly introduced as such.

I am certain that the studies I conducted leading to this book and my conclusions will be dismissed for not being a serious scientific research by those skeptics who claim that agile development does not work and that the industry should go back to "real software engineering."³ That is fine. The resources available to me for this book project are minute compared to what would be required for a serious scientific research. Even with those resources, I am not a scientist, nor do I intend to present myself as such. I am a practitioner.

Who should read this book?

If you are a practitioner, like me, and your bread and butter comes from making or helping software go live, this book has a lot to offer. I primarily wrote this book for teams that have tried to implement an agile process and ran into problems which manifest themselves as poor quality, rework and missed customer expectations. (Yes, these are problems, and plainly iterating is a workaround and not a solution.) Specification by Example, agile acceptance testing, behavior driven development, and all the other alternative names for the same thing, solve these problems. This book will help you get started with those practices and learn how to contribute better to your team, regardless of whether you qualify yourself as a tester, developer, analyst or product owner.

A few years ago, most people I met at conferences had not heard of these ideas. Most people I meet now are somewhat aware of these practices, but many failed to implement them properly. There is very little literature on problems that teams face while implementing agile development in general, so every discouraged team thinks that they are unique and that somehow the ideas don't work in their "real world". They seem surprised how I can guess three or four of their biggest problems after just five minutes of listening to them. They are often completely astonished that there are many other teams out there that have the same issues.

If you work in such a team, the first thing that this book will do for you is show you that you are not alone. The teams I interviewed for this book are not perfect — they had tons of issues as well. Instead of quitting after they hit a brick wall, they decided to drive around it or tear it down. Knowing this is often encouraging enough for people to look at their problems in a different light. I hope that after reading the book you will feel the same.

If you are in the process of implementing Specification by Example, this book will provide useful advice on how to get past your current problems and learn what you can expect in the future. You will hopefully learn from the mistakes of others and avoid hitting some problems at all.

This book is also written for experienced practitioners, people with a relatively successful implementation of Specification by Example in their process. I started conducting the interviews expecting that I knew most of what is out there, looking for external confirmation. I ended it surprised by how many different ideas people implemented in their contexts, things I never thought about. I learned a lot from these examples and hopefully so will you. The practices and the ideas described here should inspire you to try alternative solutions to your problems or realize how you can improve the process of your team once you read similar stories.

What is inside?

In Part I, I introduce Specification by Example. Instead of convincing you why you should follow the principles outlined in the book, I show you -- in the true specification by example style -- examples of benefits that teams got from this process. If you are thinking whether or not to buy this book, skim over Chapter 1 and see if any of the benefits presented there would apply to your project. In Chapter 2 I introduce the key process patterns and key artifacts of Specification by Example. In Chapter 3, I explain the idea of Living Documentation in more detail. In Chapter 4, I present the most common starting points for initiating the changes to process and team culture and advice on what to watch out for when you start implementing the process.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=714>

One of my goals with this book is to create a consistent language for patterns, ideas and artifacts that teams use to implement Specification by Example. The community has a dozen names for the practice as a whole, and twice that much for various elements of it. Different people call the same thing feature files, story tests, BDD files, acceptance tests and so on. For that reason, I also introduce what I think are very good names for all the key elements in Chapter 2. Even if you are an experienced practitioner, I suggest you read this chapter to make sure that we have the same understanding of the key names, phrases and patterns in this book.

In Part II, I present the key practices that the teams from the case studies used to implement the principles of Specification by Example. Teams in different contexts do very different things, sometimes even opposing or conflicting, to get to the same effect. In addition to the practices, I document the contexts in which the teams use them to implement the underlying principles. The chapters in Part II are roughly broken down by process areas.

Software development is not static — teams and environments change and the process must follow. I present case studies showing the journeys of a few selected teams in Part III. I write about their processes, constraints and contexts, analyzing how the processes evolved. These stories will help you get started with your journey or take the next step, find ideas and discover new ways of doing things.

In the final chapter of the book I present a summary of the key things I have learned from the case studies leading to this book.

Beyond the basics

On the traditional 'Shu-ha-ri'⁴ learning model, this book is at the 'Ha' level. 'Ha' is about breaking the old rules and showing that there are many successful models. In *Bridging the Communication Gap*, I presented my model and my experience. In this book, I try very hard not to be influenced by my background. I present things from the projects I worked on only when there is an important point to make and I do not think that any of the teams featured in the book had a similar case. So in that sense, *Specification by Example* continues where *Bridging the Communication Gap* stopped.

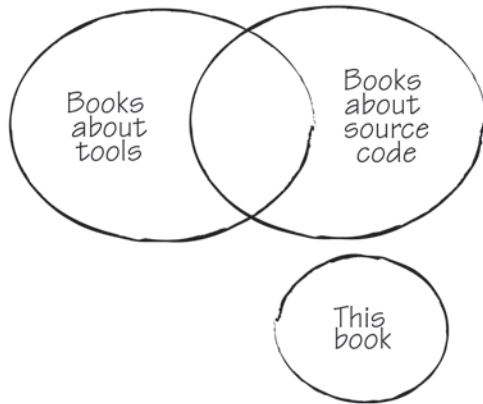
I introduce the basic principles very briefly in Chapter 2. Even if you've never heard of any of these ideas before, this should give you enough information to understand the rest of the book, but I won't go into the basics too much. I wrote about the basics of Specification by Example at length in *Bridging the Communication Gap* and have no wish to repeat myself.

Visit <http://specificationbyexample.com>, register a copy of this book and you will get the PDF of *Bridging the Communication Gap* for free if you want to go over the basics in more detail.

I do not think that I will write a follow-up on this subject on the Ri level — because that is a level beyond books. On the other hand, I believe that this book will help you move to that level. Once you start thinking that the choice of a particular tool is irrelevant, you are there.

This book has no source code and does not explain any tools

This book has no source code or instructions on how to work with a particular tool. I feel compelled to mention this upfront, because I had to explain it already several times during the publishing process (typically as an answer to the question "What do you mean? A software development book without source code? How's that possible?")



The principles and practices of Specification by Example primarily affect how people communicate in software delivery teams and how they collaborate with business users and stakeholders. I'm sure that there are many tool vendors out there who will try to sell you a technical solution for that. There are also many managers out there who would be happier to pay for their problem to go away instantly. Unfortunately for them, this is mostly a people problem, not a technical one.

Bill Gates said that "The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency". Many teams that failed with Specification by Example have just magnified their process inefficiency by automating it. Instead of focusing on a particular tool, I want to address the real reasons why teams struggle to implement these ideas. Once you get the communication and collaboration right, you will be able to choose the right tool to fit it.

A few words on the terminology

If this is your first contact with specification by example, acceptance-test driven development, agile acceptance testing, behavior-driven development or any of the other names people use for this set of practices, you have avoided years of confusion caused by misleading names. You should feel good about that and you may skip this part of the introduction. If you have already come into contact with any of those ideas, the names I use in this book might surprise you. Read on to understand why I use those names and why you should start using them as well.

While writing this book, I had the same problem we often have when writing our automated specifications. The terminology has to be consistent to make sense, but we don't necessarily see that until we write things down. Because this book is the result of a series of interviews, and many people I spoke to used different names for the same thing, it was quite hard to make the story consistent with all the different names.

I realized that the practitioners of Specification by Example, me included, have traditionally been very guilty of using technical terms to confuse both ourselves and everyone else who tries to implement these practices. Then I decided that one of my goals with this book will be to change the terminology in the community. If we want to get business users more involved, which is one of the key goals of these practices, we have to use the right names for the right things and stop confusing people.

This lesson is obvious when we write our specifications, and we know that we need to keep the naming consistent and avoid misleading terms. But we don't do this when we talk about the process. For example, when we say continuous integration in the context of specification by example, we don't really mean running integration tests. So why use that term, and then have to explain how acceptance tests are different from integration tests? Until I started using "specification workshop" as the name for a collaborative meeting about acceptance tests, it was very hard to convince business users to participate.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=714>

But a simple change in naming made the problem go away. By using better names, we can avoid a lot of completely meaningless discussions and get people started on the right path straight away.

Why Specification by Example?

I first want to explain why I chose Specification by Example (which I will sometimes abbreviate as SBE) as the overall name for the whole set of practices, as opposed to Agile Acceptance Testing, Behavior-Driven Development or Acceptance Test Driven Development.

During the Domain Driven Design Exchange 2010 conference¹ in London, Eric Evans argued that "agile" as a term has lost all meaning because anything can be called agile now. Unfortunately, he is right. I've seen too many teams that tried to implement a process that was obviously broken, but slapped the name agile on it as if that would magically make it better. This is in spite of a huge body of available literature on how to properly implement XP, Scrum and other less popular agile processes.

To get around this meaningless ambiguity and arguing whether agile works or not (and what it is), I'll avoid using the term agile in this book as much as I can. I will use it only when referring to teams that started implementing well-defined processes built on the principles outlined in the Agile Manifesto. So without being able to mention agile in every second sentence, agile acceptance testing as a name is out of the question.

The practices described here don't form a fully-fledged software development methodology. They supplement other methodologies — both iteration and flow based — to provide rigor in specifications and testing, enhance communication between various stakeholders and members of software development teams, reduce unnecessary rework and facilitate change. So I don't want to use any of the "-driven-development" names. Especially not behavior-driven-development (BDD). Don't take this as a sign that I have anything against BDD. Quite the contrary, I love BDD and consider most of what this book is about actually a central part of BDD. But BDD suffers from the naming problem as well.

What BDD actually means changes all the time. Dan North, the central authority on what BDD is and what it is not, said that BDD is a methodology at the Agile Testing, Specifications and BDD Exchange 2009². (Actually he called it "a second-generation, outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation, agile methodology") To avoid any confusion and ambiguity between what North calls BDD and what I consider BDD, I don't want to use that name. This book is about a very precise set of practices, which you can use within a range of methodologies, BDD included (if you accept that BDD is a methodology).

I also want to avoid using the word test too much. Many managers and business users unfortunately consider testing as a technical supplementary activity, not something that they want to get involved in. After all, they have dedicated testers to handle that. Specification by Example requires an active participation of stakeholders and delivery team members, including developers, testers and analysts. Without putting tests in the title, story-testing, agile acceptance testing and similar names are out.

This leaves Specification by Example as the most meaningful name with the least amount of negative baggage.

Process patterns

SBE consists of several process patterns, elements of the wider software development life cycle. The names I use for process patterns in this book are a result of several discussions at the UK Agile Testing user group meetings, Agile Alliance Functional Testing Tools mailing list and workshops. Some of them have been in use for a while; some of them will be completely new to most readers.

A popular approach in the community is to use the name of a practice or tool to describe a part of the process. Feature Injection is a good example — it is a popular name for extracting the scope of a project from the business goals. But Feature Injection is just one technique to do that, and there are alternative ways to achieve the same goal. In order to talk about what different teams do in different contexts, we need a higher-level concept that includes all those practices. A good name describes the expected outcome and clearly points to the key differentiating element of this set of practices.

In the case of Feature Injection and similar practices, the outcome is a scope for a project or a milestone. The key differentiator from the other ways of defining the scope is that we focus on the business goals. So I propose that we talk about *Deriving Scope from Goals*.

One of the biggest issues teams have with Specification by Example is who should write what and when. So we need a good name that clearly says that everyone should be involved (and that this needs to happen before the team starts programming or testing), because we want to use acceptance tests as a target for development. Test-first is a good technical name for it, but business users don't get it and it does not imply collaboration. I propose we talk about *Specifying Collaboratively* instead of test-first or writing acceptance tests.

It sounds quite normal to put every single numerical possibility into an automated functional test. Why wouldn't we do it if it is automated? But such complex tests are unusable as a communication tool, and in SBE we need to use "tests" for communication. So instead of writing functional tests, let's talk about *Illustrating using Examples* and expect the output of that to be *Key Examples* to point out that we only want enough to explain the context properly.³

Key examples are raw material, but if we just talk about acceptance testing then why not just dump complicated 50-column 100-row tables with examples into an acceptance test without any explanation? It is going to be tested by a machine anyway. With Specification by Example, the tests are for humans as well as for machines. We need to make it clear that there is a step after illustrating using examples, where we extract the minimal set of attributes and examples to specify a business rule, add a title, description and so on. I propose we call this step *Refining the Specification*.⁴

The result of this refinement is at the same time a specification, an target for development, an objective way to check acceptance and a functional regression test for later. I don't want to call this an acceptance test, because it makes it very hard to justify why this document needs to stay in domain language, be readable and easily accessible. I propose we call the result of refining a *Specification with Examples*, which immediately points to the fact that it needs to be based on examples but also contain more than just raw data. Calling this artifact a specification makes it obvious that everyone should care about it, and that it needs to be easy to understand. Apart from that, there is a completely different argument around whether these checks are there to automatically accept software or to automatically reject the code that doesn't satisfy what we need.⁵

I just don't want to spend any more time arguing with people who already paid a license for QTP that it is completely unusable for acceptance tests. As long as we talk about test automation, there is always going to be a push to use whatever horrible contraption testers already use for automation, because it is logical to managers that their teams use a single tool for test automation. Agile acceptance testing and BDD tools don't compete with QTP or things like that; they address a completely different problem. A specification shouldn't be translated into something technical just for automation. Instead of talking about test automation, let's call automating a check without distorting any information *Automating Validation Without Changing Specifications*. The fact that we need to automate validation without changing the original specification should help to avoid the horror of scripting and using technical libraries directly in test specifications. An executable specification should be unchanged from what it looked like on the whiteboard, it should not be translated to Selenium commands.

After the validation of a specification is automated, we can use it to validate the system. In effect, we get *Executable Specifications*.

We want to check all the specifications frequently to make sure that the system still does what it is supposed to do and, equally importantly, to check that the specifications still describe what the system does. If we call this regression testing, it's very hard to explain to testers why they should not go and add five million other test cases to a previously nice, small and focused specification. If we talk about continuous integration, then we get into the trouble of explaining why these tests should not always be run end-to-end and check the whole system. For some legacy systems we need to run acceptance tests against a live, deployed environment. Technical integration tests run before deployment. So let's not talk about regression testing or continuous integration, let's talk about *Validating Frequently*.

The long term pay-off from Specification by Example comes from having a reference on what the system does that is as relevant as the code itself, but much easier to read. That makes development much more efficient long term, facilitates collaboration with business users, leads to an alignment of software design and business models and just makes everyone's work much easier. But to do this, the reference really has to be relevant, it has to be maintained, and it has to be consistent internally and with code. We should not have silos of tests that use terms we had three years ago, and those we used a year ago, and so on. Going back and updating tests is a very hard thing to sell to busy teams, but going back to update documentation after a big change is expected. So let's not talk about folders filled with hundreds of tests, let's talk about *Evolving a Living Documentation System*. That makes it much easier to explain why things should be self-explanatory, why business users need access to this as well and why it has to be nicely organized so that things are easy to find.

So there it is, I chose the names not because of previous popularity but because they make sense. The names for these process patterns should create a mental model that actually points out the important things and reduces the confusion. I hope that you'll see this and adopt this new terminology as well.

1

Key Benefits

In this age of Internet, delivery speed seems to be the theme of the day in software development. A decade ago, projects lasted several years and project phases were measured in months. Today, most teams I work with deal with projects measured in months and project phases which are time-boxed to weeks or even days. Everything that smelled of long term planning simply got dropped in a realization that the future is unpredictable. Big up front software designs, detailed requirements analysis and anything else that required more time than an average project phase, were no longer viable. Good-bye code freezes and weeks of manual regression testing!

With such a high frequency of change, documentation quickly became outdated. Detailed specifications, documentation and test plans required too much effort to keep current. They were just labeled as waste.

These artifacts are collateral damage, casualties of the change to a model with fast and frequent deliveries. People who relied on them for their day-to-day work, such as business analysts or testers, often got very confused about what to do in this new world of weekly iterations. Software developers thought that they weren't affected that badly with the lack of paper documents, but this had a rippling effect on their work as well. Instead of the wastes of preparing documentation that nobody will ever read, we got the wastes of rework and just-in-case-code. Spending weeks iterating to polish the wrong product is as wasteful as was spending the same time building big plans.

In the last decade, the software development community was mostly focused on building the software the right way -- on the technical practices and ideas that ensure a high technical quality of the result. But building the product right and building the right product are two very different things. We need both to be successful.

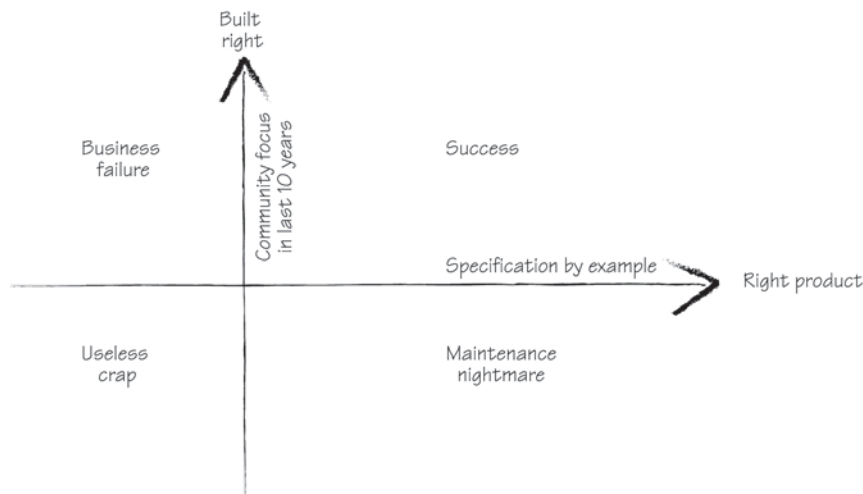


Figure 1.1 Specification by Example points us in the direction of building the right product

To build the right product effectively, our software development practices have to provide the following:

Assurance that all stakeholders and delivery team members understand what needs to be delivered, and that they all understand it the same.

Precise specifications for delivery teams to avoid wasteful rework caused by ambiguities and functional gaps.

An objective measurement of being done.

Documentation to facilitate change, both in terms of software features and team structure.

The traditional solutions to building the right product were big functional specifications, requirements documents and long testing phases. These just don't work in the world of weekly software deliveries. We need a solution that gives us a way to:

Avoid wasteful over-specifying and spending time on detailing things that will change before even being developed.

Have reliable documentation that explains what the system does so that we can change it easily.

Efficiently check that a system does what the specifications say.

Keep that documentation relevant and reliable with minimal maintenance costs.

Fit all this into short iterations and flow-based processes, so that the information on upcoming work is produced just in time.

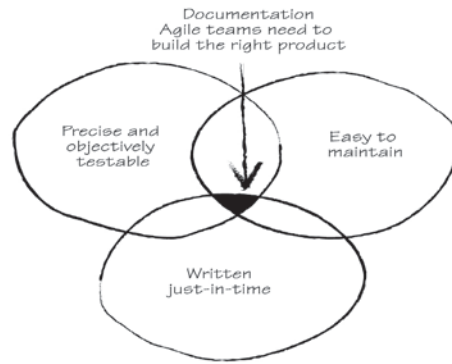


Figure 1.2 When these factors merge, we have a new solution that leads us to build the right product

Although these goals might seem conflicting at first, many teams have succeeded in fulfilling all of them. In the research leading to this book, I interviewed 30 teams that implemented around 50 projects. I looked for patterns, common practices and identified underlying principles behind these practices. Taking the smallest set of common ideas from all these projects, I can now safely define the solution: Specification by Example.

Specification by Example is a set of process patterns that help teams build the right software product. These patterns help teams write just enough documentation to facilitate change effectively in short iterations or flow based development.

I introduce the key process patterns of Specification by Example in the next chapter. In this one, I want to explain the benefits. In the true Specification by Example style, I will not build a business case for the theme of this book by explaining benefits in a theoretical introduction. Instead of that, here are 18 real world examples of teams that got big dividends from Specification by Example.

It is very hard to isolate the impact or effect of any single idea on a project. The practices described in this book work together with many other more established agile software development practices (such as test driven development, continuous integration and planning with user stories), enhancing their effectiveness. Likewise, other agile software development ideas help to support these practices.

However, when we consider a range of projects in different contexts, patterns emerge. Some of the teams I interviewed had an agile process before implementing Specification by Example, some of them

implemented the practices at the time when they were moving to an agile process. Most of the teams used iteration based processes such as Scrum and Extreme Programming (XP) or flow based processes such as Kanban, but some even used these practices in an environment that would not be considered agile by any standards. Yet most of them reported similar effects:

Implementing changes more efficiently: They had a living documentation, a reliable source of information on system functionality, which enabled them to analyze the impact of potential changes and share knowledge effectively.

Higher product quality: They defined expectations clearly and made the validation process very efficient.

Less rework: They collaborated better on specifications and ensured a shared understanding of the expectations by all team members.

Better alignment of the activities of different roles on a project: Improved collaboration led to much more regular flow of delivery.

In the next four sections, we look into each of those benefits in much more detail, with some real-world examples.

Implementing changes more efficiently

In the long term, most teams discovered that a result of implementing Specification by Example is having a living documentation — a source of information about system functionality that is as reliable as the programming language code but much easier to access and understand. This documentation system allows teams to collaboratively analyze and understand the impact of proposed changes and discuss potential solutions. It also allows them to utilize existing documentation by extending it for new requirements. This makes specifying changes and implementing them more efficient in the long term.

The long-term benefits of living documentation are the most important lesson for me from the research leading to this book and one of the most important messages I want to convey. Because of that, I cover living documentation throughout this book.

Iowa Student Loan Liquidity Corp. based in West Des Moines, Iowa, went through a fairly significant business model change in 2009. The financial market turmoil during the previous year made it nearly impossible for lenders to find funding sources for private student loans. Because of this, many lenders were forced to leave the private student loan market or change their business models. Iowa Student Loan was able to adapt. Instead of using bond proceeds to fund private student loans, it pooled funds from banks and other financial institutions.

In order to support this, they had to perform a “dramatic overhaul of a core piece of the system”, according to Tim Andersen, a software analyst and developer who works there. The team used living documentation as a primary mechanism for documenting business requirements while they were developing the software earlier. The living documentation system supported them in efficiently discovering the impact of new requirements, specifying the change and ensuring that the rest of the system still works as it was working before. Because of that, they were able to implement this fundamental change to the system and release it to production in just one month. A living documentation system was essential for this change. Andersen said:

Any system that didn't have the tests [living documentation] would halt the development and it would have been a re-write.

The Talia project team at **Pyxis Technologies** in Montreal, Canada, had a similar experience. Talia is a virtual assistant for enterprise systems, a chat robot with complex rules to communicate with employees. From the first day of development, the Talia team used Specification by Example and built up a living documentation system. After a year of development, they had to rewrite the core of the virtual agent engine from scratch. That is when the investment in living documentation really paid off. André Brissette, the Talia product director, said:

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=714>

Without that, any major refactoring would be a suicide.

Their living documentation system gave the team confidence that the new system works the same as the old one when the change was complete. It also enabled Brissette to manage the project better and gave him visibility of the progress.

The team at **Songkick.com**, a consumer website about live music based in London, UK, used a living documentation system to facilitate the changes while redeveloping activity feeds on the web site. They realized that the feeds were implemented in a way that would not scale to the required capacity. Phil Cowans, the CTO of Songkick, estimates that the team saved at least 50% of the time for implementing that change because they had a living documentation system. Cowans said:

Because we had such a good coverage and we really trusted the tests [in the living documentation system], we felt very confident making big changes to the infrastructure rapidly. We knew that the functionality wouldn't change or if it did change it would be picked up by a test.

The development team at **ePlan Services**, a pension service provider based in Denver, Colorado, US, has been using the ideas of Specification by Example since 2003. They build and maintain a financial services application with lots of stakeholders, complex business rules and complex compliance requirements. Three years after starting the project, a manager with unique knowledge about the legacy parts of the system decided to move to India. According to Lisa Crispin, a tester working for ePlan Services and author of Agile Testing, the team worked hard to learn what the manager knew and build it into living documentation.

A living documentation system enabled them to capture the specialist knowledge about their business processes and make it instantly available to all the team members. They eliminated a bottleneck in knowledge transfer which enabled them to efficiently support and extend the application.

The Central Patient Administration project team at the **IHC Group** in Oostkamp, Belgium, implemented a living documentation system with similar results. The project, which started as a re-write of a legacy mainframe system, has been going on since 2000. Pascal Mestdach, a solution architect on the project, said that the team benefited greatly from it:

There were just a few people who knew what some functionality on the legacy system did — that became much clearer now because the team has a growing suite of tests [living documentation] against that functionality and it describes what it does. Also questions can be answered when a specialist is on holiday. It's more clear to other developers what a piece of software is doing. And it is tested.

A living documentation system helps delivery teams share knowledge and deal with staff changes. It also enables businesses to react to market changes more efficiently.

Higher product quality

Specification by Example improves the collaboration between delivery team members, helps them engage better with their business users and provides clear objective targets for delivery. That typically leads to a big improvement in product quality.

Two case studies stand out when it comes to higher quality. Wes Williams, an agile coach from **Sabre Holdings**, and Andrew Jackman, a consultant developer who worked on the Sierra project at **BNP Paribas** in London, both told me stories of projects that failed several times before but succeeded with Specification by Example. The approach described in this book helped their teams conquer the complexity of the business domains that was previously unmanageable and ensure high quality of deliveries.

Wes Williams worked on a two year airline flight booking project where most complexity came from business scenarios involving global distribution and the fact that the processes are very data driven. The

©Manning Publications Co. Please post comments or corrections to the Author Online forum:

<http://www.manning-sandbox.com/forum.jspa?forumID=714>

project involved thirty developers working in three teams on two continents. The first two attempts to build the system failed, according to Williams, but this attempt succeeded. Williams said:

We went live with a large customer [a big airline] with very few issues and had only one severity 1 issue during [business acceptance] testing, related to fail-over.

Williams estimates that Specification by Example was one of the key pieces for the success of this project. In addition to higher quality, it also facilitated building trust between developers and testers.

The Sierra project at BNP Paribas is another great example of how Specification by Example leads to high quality products. Sierra is a data repository for bonds, receiving information from several internal systems, rating agencies and other external sources, consolidating the data and distributing it to various systems inside the bank. The complexity in this project is mostly driven by data and communication. Different systems and organizations use the same names for different meanings, which caused a lot of misunderstanding.

The first two attempts to implement this system failed, according to Channing Walton, one of the developers on the team that helped make the third attempt a success. Specification by Example enabled the team to tackle the complexity and ensure a shared understanding. The end result is very impressive in terms of product quality. The project has been live since 2005 “with no major incidents in production”, according to Andrew Jackman, a consultant developer on the Sierra project.

Most people working on the Sierra project at the moment were not there when the project started, but the level of quality is still very high.

A similar thing happened on a car leasing system project for a major European bank developed by **Bekk Consulting**. According to Aslak Hellesøy, a member of the original team and the author of the Cucumber automation tool for executable specifications, they had only five bugs reported in the two years since the system went live, although the software is now maintained by a completely new team.

Lance Walton worked as a process consultant for a branch of a large Swiss bank in London on a project to develop an order management system that “failed to start several times before”. The project was implemented in an environment where systems were assumed to require a support team at least as big as the development team, according to Walton.

His team used Specification by example and delivered a system to production nine months after the project started, passing the business acceptance testing in a single day and with no bugs reported for six months after that. According to Walton, it required no additional support staff, the cost was much smaller than predicted, and the team delivered earlier. For comparison, the team next to them had ten times more people doing support than development. Walton said:

At the moment the team is still releasing every week and the users are always happy with it. From the point of quality, it is superb.

The techniques of Specification by Example work for brown-field projects as well as for green-field ones. It takes time to build up trusted documentation and clean up legacy systems, but teams see benefits such as confidence in new deliverables relatively quickly.

A good example of that is the foreign exchange cash management system at **JP Morgan Chase** in London. Martin Jackson, a test automation consultant on that project, said that the business analysts expected the project to be late, but it was delivered two weeks early. High product quality enabled them to successfully complete the business acceptance testing phase in a week instead of four weeks as originally planned. Jackson said:

We deployed it and it worked. The business reported back to the board as the best UAT experience they ever had.

Specification by Example also enabled Jackson's team to quickly implement "quite a significant technical change" late in the project development, improving the precision of calculations. Jackson said:

All the functionality covered by the Fitnesse suite [living documentation] went through the whole of system test, whole of UAT and live to production without a single defect. There were several errors outside of the core calculation components that were captured during system testing. What made the UAT experience so good for the business was that when calculation errors appeared we were all pretty certain that the root cause was going to be upstream from the calculation code itself. As a result of the FitNesse suite it was easier to diagnose the source of defects and hence the cleaner and faster delivery through to production.

The software development team at **Weyerhaeuser** in Denver, US, writes and maintains several engineering applications and a calculation engine for wooden frames. Before applying Specification by Example, the construction engineers did not really get involved in software development although the team was dealing with very complex scientific calculation formulas and rules. This caused a lot of quality issues and delays, further complicated by the fact that the engine is used by several applications. The hardening phase before releasing would drag on forever and a release would rarely go out without problems according to Pierre Veragen, the SQA lead on the project.

After they implemented Specification by Example, the team now collaborates on specifications with structural engineers and automates the resulting validations. When a change request comes in, the testers work together with structural engineers on capturing the expected calculation results and record them as specifications with examples before the development starts. The engineer who would be approving a change later works directly on writing the specifications and tests.

The main benefit of the new approach, according to Veragen, is that they can make changes with confidence. When this was written, they had more than 30.000 checks in their living documentation system. According to Veragen, they haven't noticed any big bugs in the past several years and have now stopped tracking bugs. He said:

We don't need the [bug count] metrics because we know it's not coming back...engineers love the test first approach and the fact that they have direct access to automated tests.

Lance Walton worked on a credit risk management application for a branch of a large French bank in London. The project originally started with external consultants helping the team adopt Extreme Programming (XP) practices, but they did not adopt any of the Specification by Example ideas (although XP includes "customer tests", which is closely related to executable specifications). After six months, Walton joined the project and found the quality of the code very low. Although the team was delivering every two weeks, the code was written in a way that made validation very complicated. Developers would normally test only the most recently implemented features. As the system was growing, this became less and less satisfactory.

"When a release happened, people would sit around nervously, making sure that everything was still running and we'd expect a few issues to come up within hours", said Walton. After they implemented Specification by Example, the quality and confidence in the product significantly improved. Walton said:

We were pretty confident that we could release without any issues. We got to the point where we would quite happily deploy and go out for lunch without sticking around to see if it was OK.

A completely opposite example is the recent web site rewrite project at the **Trader Media Group** in Newton-le-Willows and Wimbledon, United Kingdom. The team was collaborating on specifications and

automating the validation at first. They stopped doing that because of the management pressure to seemingly deliver more functionality earlier, expecting that they will deliver faster. “We noticed that the quality took a nose dive”, said Stuart Taylor, the test team leader. “Where before it was quite hard for us [testers] to find defects, later we found that one story could produce four, five defects”.

Not just for agile teams

Collaborating on specifications is not something that only agile teams can benefit from. In *Bridging the Communication Gap*, I suggested that a similar practice could be applied to more traditional structured processes, but without any data to back that up.

Matthew Steer, a senior test consultant at the Sopra Group, helped a major telecommunication company with a third party off-shore software delivery partner implement these practices. The main reason for the change was the realization that their projects were suffering from poorly defined requirements. Steer compared the delivery in the year when these ideas were implemented to the costs of delivering software the previous year. Unsurprisingly, with a waterfall approach these projects did not get to a zero defect level, but the changes “increased upstream defect detection and reduced downstream rework and costs”, according to Steer. He said:

We were able to demonstrate the effectiveness of this approach by catching many more defects earlier in the life cycle that were traditionally found at later phases. The volumes of defects at the end of the lifecycle significantly reduced and the pile increased at the early phases of the lifecycle.

The end result was, according to Steer, saving over 1.7 million GBP on delivery costs in 2007 alone.

Less rework

Frequent releases in general promote quick feedback, so that development teams can find out that they have implemented something wrongly quicker and fix it sooner. But iterating quickly does not guarantee that a wrong thing will not be implemented in the first place.

Very often, teams have three or four stabs at implementing a feature, and developers claim that customers don't really know what they want until they get something to play with. I disagree. With Specification by Example, teams generally hit the target in the first attempt. This saves a lot of time in the long term and enables the delivery process to be more predictable and reliable.

The **Sky Network Services (SNS)** group at British Sky Broadcasting Corporation in London, UK is responsible for broadband and telephony provisioning software with a lot of business workflow and integration complexity. The group consists of six teams. They have been using Specification by Example for several years. According to Rakesh Patel, a senior agile Java developer there, the group has a very good reputation within Sky. Patel summarized this as “we do tend to deliver when we say we do”. He briefly worked with a different organization before going back to Sky and compared the two teams:

Every time they [developers in the other organization] give software to testers towards the end of the sprint testers find something wrong and it always comes back to the developers. But here [at Sky] we don't have that much churn. If we have an issue, we have an issue to make a test go green during development — it either does or it doesn't. We can raise it there and then.

Several other teams noticed a significant reduction of rework, including the one developing an aggregation application for a large US insurance provider. Their application presents a unified user interface on top of a host of mainframe and web based services. In addition to that complexity, the project has many stakeholders spread across different US states. The project was initially suffering from many functional gaps in requirements, according to Rob Park, an agile coach at LeanDog who helped the